Chapitre V

Théorie de la Complexité

V.1. Introduction

Il existe des problèmes « faciles » comme : « Trouver la plus petite chaîne simple entre les sommets x et y » et des problèmes « difficiles » comme : « Trouver la plus grande chaîne simple entre les sommets x et y ».

Informellement, un problème est dit facile si on connaît, au moins, un algorithme efficace (complexité polynomiale ou inférieure) pour le résoudre, sinon il est considéré comme étant un problème plutôt difficile (tous les algorithmes connus ont une complexité exponentielle ou supérieure).

La théorie de la complexité étudie de manière plus formelle la difficulté liée à la résolution des problèmes et les relations existantes entre eux.

V.2. Modélisation de problèmes

Pour simplifié l'étude de la difficulté des problèmes, on s'intéresse à des formes particulières de problèmes que l'on appelle « Problème de décision ».

Informellement, un problème de décision est un problème pour lequel la solution se résume à répondre « oui » ou « non ». Par exemple, le problème P1 de l'existence d'un chemin entre deux sommets x et y dans un graphe orienté G, est un problème de décision car la solution au problème se résume à répondre « oui il existe bien un tel chemin » ou alors « non, il n'existe par de chemin entre x et y dans G ». Par contre le problème P2 qui consiste à trouver un chemin entre les sommets x et y dans un graphe orienté G, n'est pas considéré comme étant un problème de décision, car il ne s'agit pas juste de dire s'il existe ou non un tel chemin, mais il faudrait le trouver (s'il existe) et lister par exemple les nœuds qui le forment. On dit que P2 est un problème général. Par contre on peut remarquer que si on sait résoudre le problème P1 avec un algorithme donné, on pourra aussi résoudre P2 avec un algorithme proche (ayant des complexités voisines), car on aura pratiquement les mêmes étapes à faire pour résoudre les deux problèmes. P1 et P2 ont apparemment la même difficulté.

Il est souvent assez facile de passer d'un problème général à un problème de décision ayant la même classe de complexité. Une fois qu'on étudie la difficulté du problème de décision (à l'aide de la théorie de la complexité) on pourra alors connaître le degré de difficulté du problème général initial.

Voici quelques exemples de problèmes généraux avec leurs équivalents sous formes de problèmes de décision :

HAM : « Le problème général qui étant donné un graphe non orienté G, consiste à trouver un cycle hamiltonien[1] (s'il existe) »
HAMD : « Le problème de décision qui étant donné un graphe non orienté G, consiste à décider si oui ou non, G contient un cycle hamiltonien »

PVC : « Le problème général du voyageur de commerce, qui étant donné un graphe non orienté complet et étiqueté avec des poids strictement positifs, consiste à déterminer le cycle hamiltonien de poids minimal »
PVCD : « Le problème de décision équivalent, qui étant donné un entier m et un graphe non orienté complet et étiqueté avec des poids strictement positifs, consiste à vérifier s'il existe un cycle hamiltonien de poids inférieur ou égal à m »

Plus formellement, un problème de décision peut être défini de la manière suivante :
Soit Y l'ensemble de toutes les instances du problème et soit X le sous-ensemble de Y formé uniquement des instances positives, c-a-d les instances pour lesquelles la réponse est « oui ». Les instances négatives (celles pour lesquelles la réponse est « non ») appartiennent donc $Y - X$.
Le problème de décision peut alors être noté ($X \subseteq Y$) et consiste à décider, pour une instance donnée y de Y, si oui ou non, $y \in X$?

Par exemple pour le problème de décision HAMD, Y représente l'ensemble de tous les graphes non orientés, alors que X représente l'ensemble de tous les graphes non orientés renfermant un ou plusieurs cycles hamiltoniens.

V.3. Modélisation des traitements

Quand on veut résoudre une instance d'un problème de décision donné, on écrit un programme qui va manipuler les données de l'instance sous forme de structure de données.

Donc toute instance de problème finira par être « codée » sous forme d'une suite de bits $\{0|1\}^*$. Cette représentation binaire des données du problème peut être vue comme un mot binaire sur l'alphabet $\Sigma = \{0,1\}$. L'ensemble des instances positives X du problème de décision représente alors (après codification) un langage L ($L \subseteq \Sigma^*$), alors que les instances négatives (l'ensemble $Y - X$) représente alors le complémentaire de L dans Σ^* ($\overline{L}$).

Donc tout algorithme qui vérifie si une instance y est positive ou non, peut être vu comme étant un automate reconnaissant un mot w du langage L.

Parmi les automates utilisés dans la théorie des langages, ceux de type 0 (les machines de Turing) ont assez de pouvoir d'expression pour être associés aux

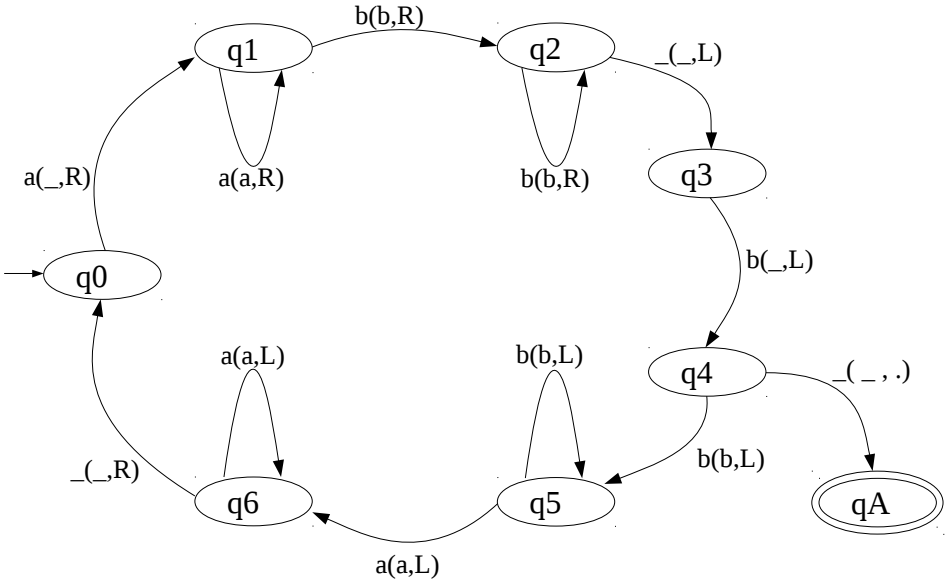
1 Un cycle hamiltonien est un cycle simple qui passe par tous les sommets du graphe

problèmes calculables (donc les langages de programmation).

De ce fait, tout algorithme de résolution d'un problème de décision ($X \subseteq Y$), n'est rien d'autre qu'une machine de Turing reconnaissant un certain langage L formé par la représentation binaire de ces instances positives.

Dans une machine de Turing, le ruban est en lecture/écriture et on peut se déplacer dans les deux sens à droite et à gauche. A chaque état de l'automate, l'action à entreprendre dépend du symbole en cours d'analyse. A la fin (lorsque tous les symboles du mot en entrée auront été lus, si l'automate se trouve dans un des états accepteurs, le mot est accepté (appartient au langage) sinon le mot est rejeté.

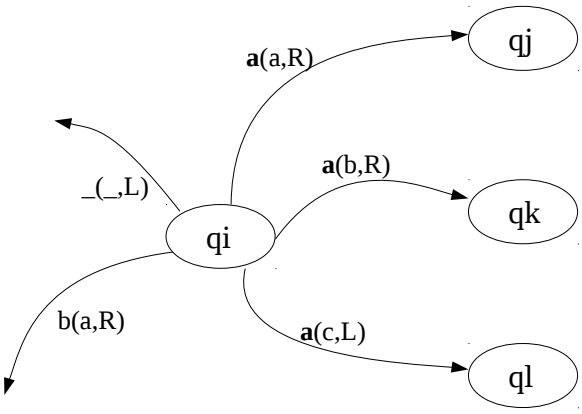
Voici un exemple d'une machine de Turing reconnaissant le langage $\{a^n b^n\}$:
Les symboles formant l'alphabet (Σ) sont 'a' et 'b'. Les cases du ruban contiennent le mot à analyser et le symbole vide '_' partout ailleurs.
Les transitions entre états sont représentées par le triplet : SymC (SymR , Dir) où SymC représente le symbole courant sur le ruban, SymR le symbole de remplacement (à écrire à la place du symbole courant) et Dir représente la direction du déplacement de la tête de lecture/écriture (Left 'L', Right 'R', ou alors rester sur place '.')
L'état initial est 'q0' et il existe un seul état accepteur dans cet exemple 'qA'.



Par exemple au niveau de l'état 'q1', si le symbole courant est un 'a' on reste dans l'état 'q1' et on écrit un 'a' tout en déplaçant la tête de lecture/écriture à droite ('R'). Si par contre le symbole courant est un 'b' on change d'état ('q2') et on se déplace à droite aussi en écrivant le même symbole 'b'.
Au niveau de l'état 'q3', si le symbole courant est un 'b' on l'efface (écriture de '_' à la place) et on se déplace à gauche ('L').

Dans une machine de Turing, le nombre de transitions effectuées pour analyser un mot, donne une indication sur la complexité en temps. Chaque transition est une étape dans le processus de d'analyse. Par contre le nombre de cases utilisées dans le ruban donne une indication sur la complexité en espace mémoire.

Une machine de Turing est dite « non déterministe » s'il existe pour un état donné, plusieurs transitions différentes étiquetées par un même symbole :



Dans l'exemple de la figure ci-dessus, quand on est dans l'état 'qi', si le symbole courant est un 'a', il y a trois transitions différentes qui peuvent être activées en même temps, soit on va vers l'état 'qj' soit vers l'état 'qk' soit vers l'état 'ql', d'où l'indéterminisme produit par l'existence d'alternatives dans le déroulement de l'exécution.

En fait, lors de l'analyse d'un mot en entrée, une machine de Turing non déterministe est supposée fonctionner de la manière suivante :
A chaque fois qu'on se trouve dans état avec plusieurs alternatives possibles, la machine est dupliquée par clonage en autant de machines que d'alternatives possibles. Chaque clone continue l'analyse, en parallèle avec les autres, sur une des transitions activables (c-a-d chaque machine traitera une alternative de l'exécution). Si une des machines s'arrête sur un état accepteur, le mot est accepté et les autres machines abandonnent leurs analyses (elles s'arrêtent). Si par contre, toutes les machines s'arrêtent sur des états non accepteurs, le mot est rejeté. Sinon, dans tous les autres cas, l'analyse continue (en parallèle) avec toutes les machines qui ne se sont pas encore arrêtées.

Il est démontré que tout langage accepté par une machine de Turing non déterministe peut l'être aussi par une machine déterministe. Il suffit juste d'explorer séquentiellement, en largeur par exemple, toutes les alternatives durant l'analyse.

Les langages de programmation existants sont bien sûr modélisés par des machines de Turing déterministes, mais on peut aussi définir (théoriquement) la

notion d'algorithme non déterministe, qui dispose d'une instruction un peu particulière 'choix(....)' qui permet de choisir, comme par magie, la bonne alternative à suivre pour trouver une solution au problème. Toutes les autres instructions sont celles que l'on rencontre dans les algorithmes déterministes usuels.

V.4. Classification des problèmes

Selon la difficulté de résolution des problèmes de décision, on peut les classer dans l'une des classes suivantes

Classe P :
C'est l'ensemble des problèmes de décision pouvant être résolus en temps polynomial, par des algorithmes déterministes (machines de Turing déterministes).
Tout problème de décision pour lequel il existe un algorithme de résolution général ayant une complexité polynomiale, est classé dans P. En fait il suffit d'exhiber une machine de Turing déterministe capable de décider le langage modélisant le problème de décision en un nombre d'étapes polynomial en fonction de la longueur du mot à analyser.

Classe NP :
C'est l'ensemble des problèmes de décision pouvant être résolus en temps polynomial, par des algorithmes non déterministes (machines de Turing non déterministes).
Tout problème de décision pour lequel il existe un algorithme de résolution général non déterministe polynomial, est classé dans NP. En fait il suffit d'exhiber une machine de Turing non déterministe capable d'accepter le langage modélisant le problème de décision en un nombre d'étapes polynomial en fonction de la longueur du mot à analyser.

Il est clair que tout les problèmes dans P sont aussi dans NP.

On peut aussi définir la classe NP sans faire référence aux machines de Turing non déterministes, en utilisant les systèmes de preuve.

Un système de preuve pour un problème de décision ($A \subseteq B$), est un ensemble Q_A (espace de preuves) et un sous-ensemble F_A de ($A \times Q_A$), où pour toutes instance x de B, on a : $x \in A \Leftrightarrow \exists\, q \in Q_A$, tel que $\langle x,q \rangle \in F_A$

La classe NP peut alors aussi être définie comme étant la classe de tous les problèmes de décision A qui admettent un système de preuve $\{Q_A, F_A\}$ vérifiant :

1. $\exists$ un polynôme p(n) tel que $\forall\, a \in A, \exists\, q \in Q_A$ de taille $\leq p(|a|)$ et $\langle a,q \rangle \in F_A$
2. Le problème de décision ($F_A \subseteq A \times Q_A$) doit être dans P

Exemple : Montrons, à l'aide d'un système de preuve, que HAMD est dans NP.

Soient Y l'ensemble des graphes non orientés et X le sous-ensemble de Y formé par les graphes non orientés contenant au moins un cycle hamiltonien.
Le problème de décision HAMD est donc noté ($X \subseteq Y$).
Soit Q_x l'ensemble de toutes les séquences de sommets.
Soit F_x l'ensemble des couples <G,q> tel que $G \in X$ et $q \in Q_x$ et pour lesquels la séquence de sommets q représente un cycle hamiltonien dans G.
Les ensembles F_x et Q_x représentent donc un système de preuve pour le problème de décision HAMD.
(i) On remarque d'une part que la longueur (en nombre de sommets) de tout cycle hamiltonien q d'un graphe G contenant n nœud est de n+1 sommets, donc il existe bien un polynôme p(n) = n+1 donnant la taille de q.
(ii) D'autre part on peut vérifier en temps polynomial pour un graphe G et une séquence de sommet q, si cette dernière représente bien un cycle hamiltonien. Il suffit pour cela de parcourir la séquence q (O(n)) pour vérifier que :

1. chaque sommet est présent une seule fois (à part le premier et le dernier) (O(n))
2. pour chaque couple de sommets successifs dans q, il existe bien une arête qui les relie dans G (O(1) pour une représentation matricielle et O(n) pour une représentation par liste)

Cet algorithme est donc bien polynomial (au max $O(n^2)$) ce qui implique que le problème de décision ($F_x \subseteq (X \times Q_x)$) est bien dans P.
D'après (i) et (ii) on déduit que le problème de décision HAMD est dans NP.

Ainsi, montrer qu'un problème de décision donné est dans NP, revient principalement à trouver un algorithme déterministe polynomial permettant de vérifier une solution donnée.

Théorème : $P \subseteq NP$ (trivial)

Par contre on ne sait pas si P = NP ou alors $P \subset NP$ (c'est toujours un problème ouvert).

Pratiquement personne ne croit vraiment que P=NP mais on n'arrive toujours pas à démontrer l'inclusion stricte.

En d'autres termes, on n'arrive pas à montrer une quelconque utilité du non-déterminisme dans la diminution de la difficulté des problèmes (de NP vers P). (l'instruction magique 'choix' dans les algorithmes non déterministes, est-elle vraiment nécessaire pour pouvoir résoudre un problème donné en temps polynomial ?)

Réduction de Turing

Soient A et B deux problèmes quelconques (pas forcément des problèmes de

décision), on dit que A est « polynomialement Turing-réductible » en B (et on note $A \leq_T^p B$) si en prenant comme hypothèse que l'on peut résoudre n'importe quelle instance de B en temps unité, on trouve un algorithme pour résoudre toute instances de A en temps polynomial.

La relation $\leq_T^p$ est un préordre (réflexive, transitive mais pas antisymétrique).

Si $A \leq_T^p B$ et $B \leq_T^p A$ alors on dit que A et B sont polynomialement Turing-équivalents (et on note $A \equiv_T^p B$).

Théorème : Si $A \leq_T^p B$ et si B peut être résolu en temps polynomial, alors A est aussi résoluble en temps polynomial.

Conséquence : Si $A \leq_T^p B$ et si B est dans P, alors A est aussi dans P.

Théorème : $HAM \equiv_T^p HAMD$

Réduction par association

Soient A et B deux problèmes de décision, définis respectivement sur des ensembles d'instance I_A et I_B, on dit que A est « polynomialement réductible par association » à B (et on note $A \leq_m^p B$) si existe une fonction f : I_A vers I_B, calculable en temps polynomial et tel que : $\forall x \in I_A, (x \in A \Leftrightarrow f(x) \in B)$

Si $A \leq_m^p B$ et $B \leq_m^p A$ alors on dit que A et B sont polynomialement équivalents par association (et on note $A \equiv_m^p B$).

Théorème : Soient A et B deux problèmes de décision, Si $A \leq_m^p B$ alors $A \leq_T^p B$.

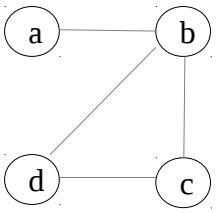
Corollaire : Si $A \leq_m^p B$ et si B est dans P, alors A est aussi dans P.

Théorème : $HAMD \leq_m^p PVCD$

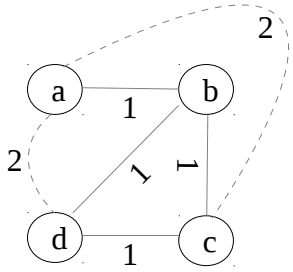
Preuve : Il faut trouver une fonction f polynomiale qui transforme toute instance de HAMD (existe t-il un cycle hamiltonien dans un graphe non orienté G quelconque) vers une instance de PVCD (existe t-il un cycle hamiltonien de poids ≤ à un entier donné m, dans un graphe non orienté, étiqueté et complet G'). De plus cette transformation doit aussi vérifier :

1. Si l'instance x de HAMD est positive alors l'instance transformée f(x) de PVCD doit aussi être positive.
2. Si l'instance transformée f(x) de PVCD est positive alors l'instance initiale x de HAMD doit aussi être positive.

Remarquons d'abord qu'il existe des graphes qui possèdent des cycles hamiltoniens et il existe aussi des graphes qui ne possèdent pas de cycles hamiltoniens. Par contre pour un graphe complet (où il existe une arête entre chaque couple de sommets du graphe) le problème de l'existence d'un cycle hamiltonien ne se pose pas, car on sait qu'il existe (n-1) ! / 2 cycles hamiltoniens différents dans un graphe complet de n sommets.
De plus dans un graphe contenant n sommets, tout cycle hamiltonien est forcément formé de n arêtes.



Graphe G
(ne contient pas de cycle hamiltonien)



Graphe G'
(tous les cycles hamiltoniens ont un poids > 4)

Considérons la transformation f suivante :
Tout graphe non orienté G se transforme en un graphe non orienté étiqueté et complet G', en gardant les mêmes sommets et les mêmes arêtes initiales de G puis on complète avec de nouvelles arêtes pour rendre le graphe complet.
Les arêtes initiales seront étiquetées par l'entier 1 et les arêtes rajoutées seront étiquetées par l'entier 2.
Ainsi s'il existait un cycle hamiltonien dans le graphe initial G (contenant n sommets), alors il existera un cycle hamiltonien formé uniquement par les arêtes initiales de G (portant un poids de 1 dans le graphe transformé G') et donc, son poids total serait exactement de n (la somme des poids des n arêtes).
Par contre s'il n'existait pas de cycle hamiltonien dans le graphe initial G, tous les cycles hamiltoniens du graphe transformé G' utiliseraient au moins une arête rajoutée (avec un poids de 2) et dans ce cas le poids du cycle sera forcément supérieur à n.
Donc avec cette transformation, on est sûr que si l'instance initiale de HAMD etait positive (le graphe non orienté quelconque G formé de n sommets contenait un cycle hamiltonien), alors l'instance transformée f(x) de PVCD sera aussi positive (le graphe G' avec m = n) et inversement si le graphe G' contient un cycle hamiltonien d epoids inférieur ou égal à n (en fait égal exactement à n), alors cela voudrait dire que le cycle n'est formé que d'arêtes initiales (étiquetées par 1) et donc le graphe initial G contenait un cycle hamiltonien.
La transformation f décrite ci-dessus est polynomiale puisqu'il s'agit de construire un graphe complet étiqueté à partir d'un graphe quelconque de n sommets (recopie des sommets existants O(n) et mise en place des toutes les arêtes possibles $O(n^2)$).

NP-Complétude

Un problème de décision A est dit NP-Complet, si :

1. $A \in NP$
2. $\forall X \in NP, \quad X \leq_m^p A$

La condition 1 stipule que les problèmes NP-Complets forment un sous ensemble de NP et la condition 2 indique qu'ils sont les plus difficiles de NP.

Si un jour quelqu'un trouve par exemple un algorithme polynomial pour résoudre un des problèmes NP-Complets, alors on pourrait résoudre tous les autres problèmes de NP en temps polynomial à l'aide de son algorithme et des transformations fournies. Cela voudrait donc dire que P = NP.

Concrètement, si un problème est classé NP-Complet, il y a très peu de chance que l'on puisse le résoudre un jour en temps polynomial.

Existe t-il de tels problèmes ?

Théorème de Cook : Le problème SAT est NP-Complet.

SAT est le problème de satisfiabilité de formules logiques :
« Existe t-il des valeurs de vérité (vrai/faux) que l'on attribue à des variables booléennes (xj) pour qu'une formule logique donnée soit satisfaite (vraie) ».

Exemple de formule : (x1 ou x3 ou x5) et (¬x1 ou x2 ou x4) et (x3 ou ¬x6)
Elle pourrait être satisfaite par exemple pour les valeurs suivantes :

x1=vrai, x2=vrai, x3=vrai, x4=vrai, x5=vrai, x6=vrai ou alors,
x1=vrai, x2=vrai, x3=faux, x4=vrai, x5=vrai, x6=faux ou alors,
x1=faux, x2=faux, x3=faux, x4=vrai, x5=vrai, x6=faux ou alors,
…

La démonstration du théorème de Cook est un peu complexe. Elle repose sur la description de la machine de Turing non déterministe à travers des formules logiques (de type SAT). Ainsi toute exécution sur une machine de Turing non déterministe peut être décrite par une formule à satisfaire. S'il existe un ensemble d'affectation de valeurs de vérité aux variables de la formule la rendant vraie, le mot en entrée sera accepté par la machine de Turing non déterministe, sinon le mot est rejeté.
De cette manière tout algorithme déterministe polynomial pouvant résoudre SAT, pourra aussi être utilisé pour résoudre, en temps polynomial, n'importe quel problème de décision dans NP. Donc SAT est NP-Complet.

Beaucoup d'autres problèmes intéressants ont depuis été classés aussi comme

étant NP-Complets, comme par exemple, HAMD, PVCD, Problème de la coloration d'un graphe, Serialisabilité d'historiques d'exécution de transactions concurrentes, Problème du Bin packing, Problèmes d'ordonnancement (Flow-shop, Job-shop, ...), Elaboration d'emploi du temps, Prévention d'interblocages, … etc.

Pour montrer qu'un problème A est NP-Complet, il suffit de montrer qu'il est dans NP et de trouver une réduction polynomiale (par association) liant un problème B déjà classé NP-Complet au nouveau problème A. C'est-à-dire il faut montrer que $B \leq_m^p A$ (avec B un problème de décision déjà classé NP-Complet).

De cette manière si un jour quelqu'un trouve un algorithme polynomial pour résoudre A, on pourrait alors l'utiliser pour résoudre B en temps polynomial aussi (à l'aide de la réduction utilisé pour la démonstration de la NP-Complétude de A). Ensuite on utilisera la réduction ayant montrer que B est NP-Complet pour résoudre un autre problème et ainsi de suite jusqu'à arriver à SAT. Une fois que l'on pourra résoudre SAT en temps polynomial, tout programme s'exécutant sur la machine de Turing non déterministe en temps polynomial, pourra à son tour être exécuté par un algorithme (cette fois-ci) déterministe et en temps polynomial. C'est-à-dire tous les problèmes de NP pourront alors être résolus en temps polynomial.