La complexité

Maria Malek

4 décembre 2010

Notions générales

La complexité en temps des algorithmes : le temps nécessaire à leurs exécutions dépend de :

- La machine utilisée (systèmes d'exploitation, langage, compilateur).
- Les données auxquelles l'algorithme est appliqué

Le temps de calcul est influencé par la taille des données traitées.
Exemple : Un algorithme de tri par sélection a une fonction de complexité de la forme cn^2, où n est le nombre d'entiers à trier.

Algorithmes efficaces

- Algorithme efficace
  - A partir de quelle complexité peut-on considérer qu'un algorithme est efficace ? $O(n), O(n^2), O(n^3)$,
  - Un algorithme polynomial de complexité $O(n^{100})$ n'est pas efficace.
  - Le plus souvent les algorithmes polynomiaux connus correspondent à un degré inférieur à 5.
- Définition I : Soit M une machine de Turing déterministe qui s'arrête toujours. La complexité en temps de est la fonction :

$$T_M(n) = max\{m | \exists x \in \Sigma^*, |x| = n \text{ et l'exécution de M sur x comporte m étapes}\}$$

- Cette fonction donne le nombre d'étapes maximum pour décider un mot de longueur n.

La classe P

- Définition II : Une machine de Turing est polynomiale (en temps) s'il existe un polynôme p(n) telque la fonction de complexité satisfait :

$$T_M(n) <= p(n)$$

pour tout $n >= 0$

- Définition III : La classe P est la classe des langages décidés par une machine de Turing polynomiale.
- Classe des algorithmes efficace.
- Définition IV : Une fonction est calculable en temps polynomial s'il existe une machine de Turing polynomiale qui la calcule.

Les transformations polynomiales - 1

- Définition V : Soit un langage $L_1 \subseteq \Sigma_1^*$ et un langage $L_2 \subseteq \Sigma_2^*$. Une transformation polynomiale de L_1 vers L_2 (on note $L_1 \propto L_2$) est une fonction $f : \Sigma^* \longrightarrow \Sigma^*$ qui satisfait :
  1. elle est calculable en temps polynomial.
  2. $f(x) \in L_2$ ssi $x \in L_1$.
- Soient les deux exemples :
  1. Problème de voyageur de commerce (VC) : dans un graphe G=(V,E), V est un ensemble de n villes et $d(v_i, v_j)$ est la distance pour aller de v_i à v_j, déterminer s'il existe un parcours fermé inférieur à b : il s'agit de trouver une permutation $v_{p_1}, v_{p_2}, .., v_{p_n}$ telle que $\sum_{1<=i<n} d(v_{p_i}, v_{p_i}) + d(v_{p_n}, v_{p_1}) <= b$
  2. Problème de circuit hamiltonien (CH) : dans un graphe G=(X,E), déterminer s'il existe un parcours fermé du graphe contenant chaque sommet une et une seule fois. On cherche une permutation $x_{p_1}, x_{p_2}, .., x_{p_n}$ telle que :$(x_{p_i}, x_{p_{i+1}}) \in E$

Les transformations polynomiales - 2

- Il existe une transformation polynomiale du problème du (CH) vers le problème de (VC).
  1. S=V
  2. $$d(v_i, v_j = \begin{cases} 1 & \text{si} \quad (v_i, v_j) \in E \\ 2 & \text{si} \quad (v_i, v_j) \notin E \end{cases}$$
  3. La constante b est égale au nombre de villes, c'est à dire $b = |V|$
- Il faut aussi démontrer que cette transformation préserve le caractère positif des instances.

Propriétés des transformations polynomiales

- Lemme 1 : Si $L_1 \propto L_2$ alors :
  - Si $L_1 \in P$ alors $L_2 \in P$.
  - Si $L_1 \notin P$ alors $L_2 \notin P$.
- Lemme 2 : Les transformations polynomiales sont transitives : Si $L_1 \propto L_2$ et $L_2 \propto L_3$ alors $L_1 \propto L_3$.
- Définition VI : Deux langages L_1 et L_2 sont polynomialement équivalents (on note $L_1 \equiv L_2$) ssi $L_1 \propto L_2$ et $L_2 \propto L_1$.

La classe NP - exemple ! !

- Le problème de circuit hamiltonien (CH) est un problème NP :
  - Enumérer toutes le permutations possibles (coûte cher) :*énumérer toutes les instances d'un problème*.
  - Vérifier pour une permutation donnée si elle correspond à un circuit hamiltonien (ne côte pas cher) : vérifier si une instance est positive ou non.
- Pour vérifier qu'une permutation donnée est un circuit hamiltonien ou non :
  - On peut construire une machine de Turing it non déterministe qui accepte une instance de CH ssi il existe une permutation définissant un circuit hamiltonien pour l'instance en question.

La classe NP - Théorie ! !

- Définition VII : Le temps de calcul d'une machine de Turing non déterministe sur un mot w est donné par :
  - la longueur de la plus courte exécution si le mot est accepté.
  - la valeur 1 (par convention) si le mot n'est pas accepté.
- Définition VIII : Soit M une machine de Turing non déterministe qui s'arrête toujours. La complexité en temps de est la fonction :

$$T_M(n) = max\{m | \exists x \in \Sigma^*, |x| = n \text{ et l'exécution de M sur x comporte m étapes}\}$$

- Définition IX : La classe NP (non déterministe polynomial) est la classe des langages acceptés par une machine de Turing non déterministe polynomiale.

La classe NP

- Le problème CH appartient à la classe NP.
- L'algorithme non déterministe résolvant ce problème est le suivant :
  1. L'algorithme génère d'un façon non déterministe une permutation ;
  2. L'algorithme vérifie que la permutation générée correspond à un circuit hamiltonien.
- C'est un algorithme non déterministe qui ne teste pas d'une façon exhaustive toute les permutations. Il a donc une complexité polynomiale.
- Théorème I : Soit L un langage NP. Il existe une machine de Turing M *déterministe* et un polynôme p(n) tel que M décide L et est de complexité bornée par $2^{p(n)}$.
- Tout langage appartenant à NP peut être décidé en temps exponentiel.

La structure deNP

- Définition X : Une classe d'équivalence polynomiale C_1 est inférieure à une classe d'équivalence C_2 ($C_1 <= C_2$) s'il existe une transformation polynomiale de tout langage C_1 vers un langage C_2.
- Lemme 3 : La classe NP contient la classe P ($P \subseteq NP$).
- Lemme 4 : La classe P est une classe d'équivalence polynomiale.
- Lemme 5 : Pour tout $L_1 \in P$ et pour tout $L_2 \in NP$ on $L_1 \propto L_2$.
- Par conséquence : Pour toute classe d'équivalence $C \subseteq NP$, nous avons $P <= C$.

Les problèmes NP-complets

- Définition XI : un langage L est NP-complet si
  1. $L \in NP$.
  2. Pour tout langage $L' \in NP$, $L' \propto L$.
- Théorème II : S'il existe un langage NP-complet L décidé par un algorithme polynomial alors tous les langages de NP sont décidables en temps polynomial, c'est à dire P=NP.
- Premier langage NP-complet est celui de la satisfiabilité d'une forme normale conjonctive (SAT) : Existe-t-il et une interprétation qui la rend vraie ?
- Théorème de Cook : Le problème SAT est NP-complet.