Leçon 3.
Vous avez dit *complexe* ?

Denis Trystram

September 26, 2012

Résumé : les notions importantes à retenir :
Savoir faire : Ecrire une preuve de NP-complétude pour des problèmes simples.

1 Les ensembles Pet NP

Les quatre problèmes présentés en introduction à la section **??** sont de natures diverses. Leurs résolutions semblent plus on moins "évidentes" ou "rapides". Cette notion de difficulté de résolution a très tôt éveillé l'intérêt des chercheurs. La notion de complexité a été introduite pour qualifier la nature plus ou moins ardue des problèmes qui admettent une solution sous forme de procédure effective.

La classe de complexité qui nous intéresse *a priori*, $\mathcal{P}$, correspond aux problèmes pouvant être résolus en temps polynomial. Si tous les algorithmes polynomiaux ne sont pas efficaces (un algorithme de complexité n^{100} est inutilisable en pratique), un algorithme exponentiel, de complexité 2^n, est clairement inefficace. Malheureusement une très large proportion de problèmes "intéressants" ont peu d'espoir d'admettre un algorithme de résolution polynomiale. La plupart de ces problèmes appartiennent à une classe de complexité plus vaste, $\mathcal{NP}$. La différence entre $\mathcal{P}$ et $\mathcal{NP}$ est que pour un problème de $\mathcal{P}$ il est possible de *trouver* en temps polynomial la réponse à toute instance, tandis que pour un problème de $\mathcal{NP}$ il est possible de *vérifier* en temps polynomial qu'une réponse est correcte.

La théorie de la complexité se définit sur les problèmes de décision, et cherche à déterminer le plus petit temps d'exécution nécessaire à un algorithme pour

décider un problème. La modélisation adoptée au chapitre précédent de la notion de résolution de problème et de l'exécution d'un algorithme conduit à reformuler formellement notre objet d'étude comme la détermination du plus petit temps de calcul nécessaire à une machine de Turing pour décider un langage. Rappelons que le passage des problèmes de décision à la reconnaissance de langages se fait par le choix d'un codage naturel des instances. La complexité d'un problème ne sera ainsi définie qu'à un polynome près, dépendant du choix du codage naturel.

Les complexités en temps et en espace (mémoire) sont ainsi définies par rapport au modèle de la machine de Turing comme le nombre de transitions et le nombre de cases mémoires utilisées. Nous exprimons la complexité en fonction de la taille de l'instance à traiter, c'est à dire en fonction de la taille $|w|$ du mot en entrée. Nous dirons ainsi que machine de Turing déterministe T décide un langage L en temps $f(n)$ si pour tout mot d'entrée w sur Σ^*, la machine T accepte (ssi $w \in L$) ou rejete (ssi $w \notin L$) après au plus $f(|w|)$ transitions. Il est important de remarquer que la complexité d'un algorithme, et donc d'un problème, est définie comme une complexité dans le pire des cas, c'est à dire par rapport à l'entrée la plus défavorable pour l'algorithme. On peut alors définir la classe $TIME(f(n))$ comme l'ensemble des langages pouvant être décidés en temps $\mathcal{O}(f(n))$ par une machine de Turing déterministe.

Définition 1 *La classe $\mathcal{P}$ est l'ensemble des langages L pouvant être décidés par une machine de Turing déterministe polynomiale.*

$$\mathcal{P} = \bigcup_k TIME(n^k)$$

On dira par extension qu'un problème de décision Π appartient à $\mathcal{P}$ si pour une fonction d'encodage naturelle, le langage $L_\Pi \in \mathcal{P}$. Par exemple, le problème Reachability décidant de l'existence d'un chemin entre 2 sommets d'un graphe est polynomiale (en fait linéaire).

La classe $\mathcal{NP}$ est définie par rapport aux machines de Turing non déterministes (NTM). Rappelons qu'une NTM accepte un mot simplement si il existe une exécution acceptant ce mot. Ainsi une NTM T *reconnait* le langage L en temps $f(n)$ si :

- Pour tout mot $w \in L$, il existe une exécution de T acceptant w en au plus $f(|w|)$ transitions.

- Pour tout mot $w \notin L$, aucune exécution de T (quelle que soit sa longueur) ne conduit à un état accepteur.

Reconnaître un langage en temps polynomial n'implique pas à première vue qu'on puisse le décider en temps polynomial. $\mathcal{NP}$ est ainsi l'alter ego de $\mathcal{P}$, marquant la différence *reconnaître* versus *décider*. Nous définissons la classe $NTIME(f(n))$ comme l'ensemble des langages pouvant être *reconnus* en temps $\mathcal{O}(f(n))$ par une machine de Turing non déterministe.

Définition 2 *La classe $\mathcal{NP}$ est l'ensemble des langages L pouvant être reconnus par une machine de Turing non déterministe polynomiale.*

$$\mathcal{NP} = \bigcup_k NTIME(n^k)$$

Il est clair que nous avons l'inclusion $\mathcal{P} \subseteq \mathcal{NP}$. En effet, la machine de Turing déterministe est un cas particulier de machine de Turing non déterministe... Le problème ouvert le plus célèbre de la complexité est de savoir si $\mathcal{P} = \mathcal{NP}$ou si cette inclusion est stricte.

Attention, soulignons que la complexité d'un problème est défini par rapport à un codage naturel de ses instances. Considérons par exemple le problème du test de primalité d'un nombre :
Prime un entier N N est-il premier?

Il existe un algorithme connu pour répondre à cette question en temps $O(\sqrt{N})$. Mais un codage naturel de PRIME consiste à représenter N en binaire (ou décimal). Le mot d'entrée étant ainsi codé sur $\log N$ bits, la complexité de l'algorithme est en fait exponentielle. Ce problème est un exemple de problème dont il n'est pas facile simplement de prouver qu'il est dans $\mathcal{NP}$.

Une autre caractérisation possible $\mathcal{NP}$ consiste, plutôt qu'à exhiber un algorithme polynomiale non déterministe reconnaissant un langage, à exhiber un certificat de positivité *concis* (polynomiale). C'est à dire que pour tout mot du langage, il existe un preuve d'appartenance vérifiable en temps polynomial.

Définition 3 *$\mathcal{NP}$ est l'ensemble des langages L tels que pour tout mot $w \in L$, il existe une preuve π_w d'appartenance de w à L vérifiable en temps (déterministe) polynomiale en $|w|$.*

Ainsi pour le problème HamiltonianCircuit (HC) de décider l'existence d'un cycle Hamiltonien dans un graphe :
HC Un graphe $G = (V, E)$ Existe-t-il un cycle Hamiltonien, c'est à dire un cycle passant une fois et une seule par chaque sommet.

Le problème HC appartient à $\mathcal{NP}$. Considérons comme codage naturel la matrice d'adjacence du graphe. Un certificat de positivité consiste à donner une permutation des sommets. Ce certificat est de taille $\mathcal{O}(n \log n)$. On peut vérifier en temps $\mathcal{O}(n^2)$ que la permutation correspond à un cycle du graphe. Ce certificat est bien polynomial en la taille de l'instance $\mathcal{O}(n^2)$.

2 Approfondissements : les classes de complexité

Ladner
Hiérarchie polynomiale.