UNIVERSITE MOHAMED KHIDER – BISKRA
FACULTE DES SCIENCES EXACTES, DES SCIENCES DE LA NATURE ET DE LA VIE
DEPARTEMENT D'INFORMATIQUE

Complexité et Optimisation

Master d'informatique

Réalisé par : Dr. S. SLATNIA

Version amélioée

05/01/2016

CHAPITRE 2.

CLASSES ET COMPLEXITE D'UN PROBLEME

1. Complexité d'un problème, Présentation et définitions

L'étude de la complexité des problèmes -computational complexity- s'attache à déterminer la complexité nécessaire d'un problème et à classifier les problèmes selon celle-ci, donc à répondre à des questions comme:

- Quelle est la complexité minimale d'un algorithme résolvant tel problème?
- Comment peut-on dire qu'un algorithme est optimal (en complexité)?
- Existe-t-il un algorithme polynomial pour résoudre un problème donné?
- Qu'est-ce qu'un problème "dur"?
- Comment prouver qu'un problème est au moins aussi "dur" qu'un autre?

Définition.
La complexité d'un problème est la complexité minimale dans le pire des cas d'un algorithme qui le résout. C'est souvent la complexité en temps qu'on considère mais on peut s'intéresser à d'autres mesures comme par exemple la complexité en espace-.

Calculer la complexité d'un problème est une chose extremement ardue en général. On se contente souvent d'encadrer:

- Pour trouver une *borne supérieure*, il suffit de trouver **UN** algorithme.
- Pour trouver une "***bonne***" *borne inférieure*, les choses sont souvent plus dures... : par exemple, pour montrer par exemple qu'un problème est de complexité au moins exponentielle, il faut montrer que **TOUT** algorithme le résolvant est exponentiel.

selon Sophie Tison, Cerner exactement la complexité d'un problème est souvent fort difficile : quand les deux bornes coıncident, c'est l'idéal mais c'est assez rare!

2. Trois méthodes pour trouver une borne inférieure

2.1. Les méthodes dites d'oracle ou d'adversaire

On suppose qu'il existe un algorithme utilisant moins d'un certain nombre d'opérations d'un certain type; on construit alors une donnée qui met en défaut l'algorithme.

2.2. Les arbres de décision

Ils sont utilisés pour les algorithmes de type recherche ou tri "par comparaisons": on suppose que seules des comparaisons entre les éléments sont utilisées pour obtenir de l'information sur l'ordre ou l'égalité des éléments.

Un arbre de décision "représente" toutes les comparaisons exécutées par l'algorithme sur les données d'une certaine taille:

- Un noeud correspond à une comparaison,
- Ses fils aux différentes réponses possibles de la comparaison (donc si le test est à valeur booléenne, les noeuds sont binaires);
- Une exécution possible correspond donc à une branche
- Deux données correspondant à la même branche correspondront à la même suite d'instructions.

2.3. Les Réductions

Le principe est simple. On essaie de ramener un problème à un autre (dont par exemple on connaıt la complexité).

Exemple. Si il existait un algorithme en O(nlogn) pour calculer le carré d'un entier de n chiffres, il existerait un algorithme en O(nlogn) pour calculer le produit de deux entiers de n chiffres,

3. Classes de complexité d'un problème

On s'intéresse aux ressources nécessaires pour la résolution de problèmes. Les ressources essentielles sont le temps et l'espace. L'idée est de classifier les problèmes selon l'ordre de grandeur de leur complexité.

3.1. Classes de complexité en temps

On ne considère ici que des problèmes décidables. On suppose que toutes les machines de Turing considérées s'arrêtent toujours. On définit le temps et espace de calcul relativement aux machines de Turing.

- **TIME(t(n))** = { L | L peut être décidé en temps t(n) par une machine de Turing déterministe }.
- **NTIME(t(n))** = { L | L peut être décidé en temps t(n) par une machine de Turing non déterministe }.
- Classes importantes : celles des problèmes qui peuvent être résolus en temps polynomial par une machine de Turing.

3.1.1. La classe P

- **Définition 1.** La classe des problèmes qui peuvent être résolus en temps polynomial par une machine déterministe.
  $\mathbf{P = \cup_{k\geq 0} TIME(n^k)}$

- **Définition 2. La classe P (ou PTIME)** est la classe des problèmes de décision pour lesquels il existe un algorithme de résolution polynomial en temps.

Cette définition est (quasiment-) indépendante du modèle d'algorithme choisi: un algorithme polynomial en C sera polynomial si il est traduit en termes de machine de Turing et les modèles classiques de calcul (excepté les ordinateurs quantiques) sont polynomialement équivalents.

- **Définition 3**. P est la classe des problèmes praticables

Un problème de décision est dit praticable si il est dans P, impraticable sinon;

"Existe-t-il un algorithme praticable pour ce problème?" Se ramène à "ce problème est-il dans P?

3.1.2. La classe NP

- **Définition 1**. Classe des problèmes qui peuvent être résolus en temps polynomial par une machine non déterministe.
  $\mathbf{NP = \cup_{k\geq 0} NTIME(n^k)}$

- **Définition 2**. La classe **NP** contient énormément de propriétés associées à des problèmes courants qui apparaissent dans de multiples domaines: problèmes de tournées, d'emploi du temps, placement de taches, affectation de fréquences, rotation d'équipages, découpage, ...

3.1.3. La classe EXP

- **Définition 1**. Classe des problèmes qui peuvent être résolus en temps exponentiel par une machine déterministe.
  EXP = $\cup_{k\geq 0}$ TIME(2^{nk})

- **Définition 2**. **EXPTIME,** la classe des problèmes de décision pour lesquels il existe un algorithme de résolution exponentiel en temps.

PSPACE: la classe des problèmes de décision pour lesquels il existe un algorithme de résolution polynomial en espace. Bien sur PTIME est inclus dans PSPACE: un algo polynmial en temps "consomme au plus un espace polynomial".

La classe **NP** contient **P** et est contenue dans **ExpTime** (et même dans **Pspace**). Pour beaucoup de propriétés NP, on n'a pas trouvé d'algorithme polynomial, mais pour aucune d'entre elles, on n'a prouvé qu'elle ne pouvait pas être décidée en temps polynomial.

- **P ⊆ NP ⊆ EXP**

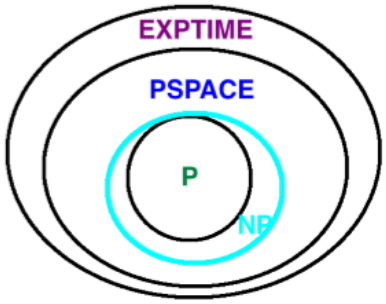

Figure 15. Classes de complexité d'un problème

3.2. Classes de complexité en espace

- **SPACE (f(n)) :** classe des problèmes de décision solvables par une TM (à K+2 bandes) en espace f(n)

- **NSPACE (f(n)) :** classe des problèmes de décision solvables par une NTM (à K+2 bandes) en espace f(n)

- **PSPACE:** classe de tous les problèmes de décision solvables par une TM utilisant un espace de travail limité par le polynôme de la taille de l'entrée.

- **NPSPACE :** classe de tous les problèmes de décision solvables par une NTM en espace polynomial de la taille de l'entrée.

4. NP-Complétude

Définition.
Un problème est simple quand le nombre d'étapes nécessaires à sa résolution peut-être borné par un polynôme ; c'est un problème **polynomial**, que l'on notera avec P.
Sinon, il est **difficile** : il s'agit d'un problème non-polynomial, noté **NP**.

De petites variations d'un problème peuvent être suffisantes pour passer d'un problème P à un problème NP :

- On veut partitionner un ensemble en deux sous-ensembles de même cardinalité, telle que la différence des sommes de ces sous-ensembles soit maximale.
- On veut partitionner un ensemble en deux sous-ensembles de même cardinalité, telle que la différence des sommes de ces sous-ensembles soit minimale.

Il est facile de couper un ensemble en deux de façon à avoir cette différence maximale.

Exemple :
On peut trier l'ensemble, et prendre comme parties les éléments de la fin et les éléments du début.
→ Obtenir une *différence minimale est un problème de combinatoire*.

Il existe de nombreux problèmes NP, souvent en combinatoire (l'algorithmique difficile) :
- Problème du voyageur de commerce (Travelling Salesman). Y a-t-il un cycle qui visite tous les noeuds d'un graphe, et dont le poids soit au plus K ? Minimiser le trajet pour voir les clients
- Factorisation. On veut décomposer un grand nombre en nombres premiers.
- Coloration de graphes. On veut attribuer à chaque sommet une couleur sans que deux sommets reliés soient de la même couleur ; le nombre minimal de couleurs requises est dit 'nombre chromatique' et noté $\gamma(G)$. Déterminer $\gamma(G)$ est NP-Complet dans le cas général.

Le problème qui se pose maintenant est de voir lorsque deux problèmes sont du même niveau de difficulté. L'outil de base pour établir des relations entre les complexités de différents problèmes est la *Réduction*.

5. Réduction d'un problème Q à un problème Π

Soit X une donnée d'entrée pour le problème Q, dont on veut une réponse 'oui' ou 'non'.
On transforme X en une donnée d'entrée Y pour le problème Π avec un algorithme A.
La réponse donnée au problème Π avec la donnée Y doit être la même que celle donnée au problème Q avec la donnée Y.

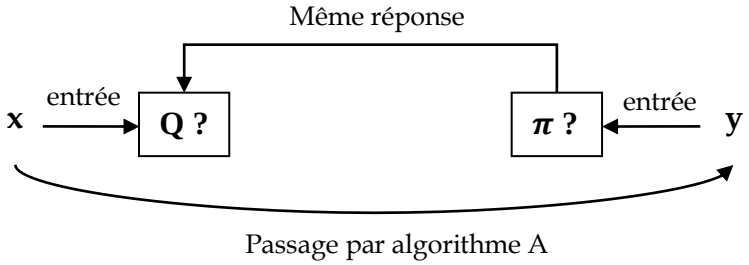

Figure 16. Réduction d'un problème Q à un problème Π

Si la donnée X peut être transformée en donnée Y par un algorithme polynomial, alors il n'est pas plus 'difficile' de résoudre Π que Q. Avec un polynôme, on passe d'un problème à l'autre.
Lorsque l'algorithme A est polynomial, on note Q → pΠ.
Si Q → pΠ et Π → pQ alors Q = pΠ ; on dit que Q et Π sont des problèmes équivalents en temps polynomial.

La plupart des problèmes d'optimisation sont NP-difficiles, en raison de la croissance de la complexité pour explorer un espace de recherche très grand, c'est-à-dire traiter des données de taille énormément pour atteindre la solution de problème.