INF4705 — Analyse et conception d'algorithmes

CHAPITRE 11 : INTRODUCTION AUX CLASSES DE COMPLEXITÉ

Nous avons déjà constaté que certains problèmes sont plus "difficiles" que d'autres. En fait, certains problèmes ne peuvent même pas être résolus. Nous verrons dans ce chapitre que certaines classes de complexité ont été définies afin de regrouper des problèmes dont le degré de difficulté est semblable.

11.1 Préliminaires

Il est préférable, pour des raisons techniques, de se restreindre aux *problèmes de décision*, c'est-à-dire ceux dont la réponse est oui ou non. Il est généralement facile de reformuler un problème d'optimisation en un problème de décision : par exemple, "Quel est le nombre minimum de couleurs nécessaires pour colorier un graphe G?" devient "Est-ce qu'on peut colorier G avec k couleurs?". Ces deux versions ne sont pas équivalentes mais il est quand même facile de répondre au premier en répétant plusieurs fois le second.

Nous prendrons également pour acquis qu'un exemplaire de problème est représenté dans une codification raisonnable : on ne peut se permettre de tricher en évitant une représentation beaucoup plus succinte (p. ex. m symboles pour représenter le nombre m plutôt que $\lg m$ symboles (en notation binaire ou décimale)).

Une *classe de complexité* est un ensemble de tous les problèmes pouvant être résolus sans consommer plus qu'une certaine quantité d'une ressource (temps, espace). Considérons d'abord la ressource temps de calcul et une quantité polynomiale (dans la taille de l'exemplaire) de celle-ci.

11.2 $\mathcal{P}$ et $\mathcal{NP}$

La classe de complexité $\mathcal{P}$ est l'ensemble des problèmes de décision qu'on peut résoudre en un temps polynomial dans la taille de l'exemplaire (autrement dit, en un temps dans $\mathcal{O}(n^k)$ pour un certain k fixé).

La classe de complexité $\mathcal{NP}$ est l'ensemble des problèmes de décision dont on peut *vérifier* une réponse *affirmative* en un temps polynomial dans la taille de l'exemplaire. Ce *n'est pas* l'ensemble des problèmes qu'on peut résoudre en temps non polynomial. C'est par contre l'ensemble des problèmes de décision qu'on peut résoudre en un temps polynomial avec un algorithme *non déterministe*.

Un algorithme non déterministe fait les bons choix pendant son exécution ou, autrement dit, explore tous les choix en même temps.

$\mathcal{P} \subseteq \mathcal{NP}$ mais $\mathcal{P} \stackrel{?}{\neq} \mathcal{NP}$: c'est *la* grande question en informatique (théorique).

La classe de complexité $\mathcal{NPC}$ est l'ensemble des problèmes $\mathcal{NP}$-complets c'est-à-dire qui appartiennent à $\mathcal{NP}$ et sont aussi difficiles que chacun des problèmes dans $\mathcal{NP}$: avec un algorithme pour résoudre un problème $\mathcal{NP}$-complet, on peut résoudre aussi facilement n'importe quel problème dans $\mathcal{NP}$.

Pour comprendre cela, nous avons besoin de la notion de *réduction de problème*. Un problème A se réduit à un autre problème B si on sait transformer efficacement chaque exemplaire e_A de A en un exemplaire e_B de B de telle façon qu'une réponse positive serait obtenue pour e_B si et seulement si une réponse positive serait obtenue pour e_A. Un algorithme résolvant B peut ainsi servir à résoudre A.

Exemple 1 *Le problème "Élever au carré" se réduit à celui de "Multiplier" : à tout exemplaire $\langle x \rangle$ pour le premier on fait correspondre $\langle x, x \rangle$ pour le second.*

Mais existe-t-il vraiment des problèmes $\mathcal{NP}$-complets? Le premier problème $\mathcal{NP}$-complet a été celui de satisfaction d'une formule Booléenne (SAT).

Soit $U = \{u_1, u_2, \ldots, u_m\}$ un ensemble de variables Booléennes. On appelle *littéral* une variable ou sa négation. Une *clause* est une disjonction de littéraux ; un ensemble C de clauses est interprété comme une conjonction de celles-ci. Une *affectation de vérité* $\mathcal{A}$ est une fonction de U vers {vrai, faux}. Une clause est

dite satisfaite (vraie) si et seulement si au moins un de ses littéraux est vrai sous $\mathcal{A}$. C est satisfait si et seulement si chacune de ses clauses est satisfaite pour une certaine affectation $\mathcal{A}$.

Exemple 2 $U = \{u_1, u_2\}$, $C = \{\{u_1, \neg u_2\}, \{\neg u_1, u_2\}\} \equiv (u_1 \vee \neg u_2) \wedge (\neg u_1 \vee u_2)$.
C est satisfait avec $\mathcal{A}(u_1) =$ vrai , $\mathcal{A}(u_2) =$ vrai .

Le problème de satisfaction SAT est donc le suivant :

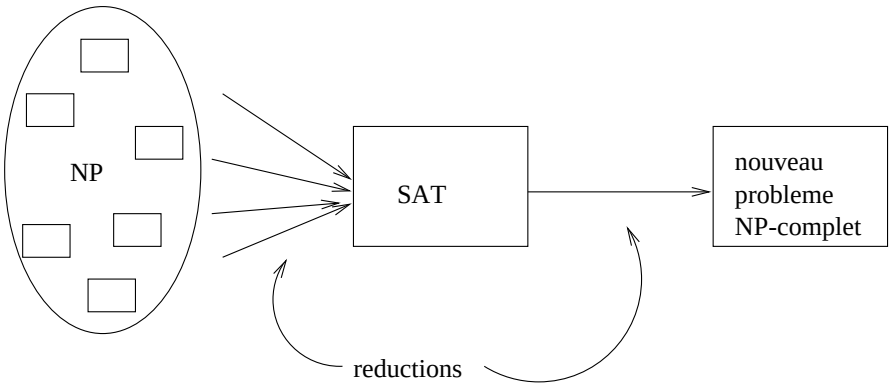
> Étant donné un ensemble de variables U et un ensemble de clauses C sur U, existe-t-il une assignation de vérité satisfaisant C ?

Théorème 1 (Cook 1971 (et Levin 1973)) $SAT \in \mathcal{NPC}$

L'idée de la démonstration est de construire une formule Booléenne qui simule un algorithme potentiel.

Comment trouver d'autres problèmes $\mathcal{NP}$-complets ? Démontrons que $\pi \in \mathcal{NPC}$ en utilisant une réduction de problème :

1. Montrer que $\pi \in \mathcal{NP}$.
2. Choisir $\pi' \in \mathcal{NPC}$.
3. Concevoir une transformation f qui fait correspondre à chaque exemplaire de π' un exemplaire de π.
4. Montrer que f se calcule en temps polynomial dans la taille d'un exemplaire de π'
5. Montrer l'équivalence : réponse positive ssi réponse positive



Un problème est dit *$\mathcal{NP}$-difficile* s'il est aussi difficile que chacun des problèmes dans $\mathcal{NP}$ (au sens de la réduction de problème) sans nécessairement être dans $\mathcal{NP}$. Cela nous permet de faire des affirmations à propos de problèmes autres que ceux de décision.

Exemple 3 *"Quel est le nombre minimum de couleurs nécessaires pour colorier un graphe G ?" est un problème $\mathcal{NP}$-difficile.*

La classe de complexité co-$\mathcal{NP}$ est l'ensemble des problèmes de décision dont on peut vérifier une réponse *négative* en un temps polynomial dans la taille de l'exemplaire.

Exemple 4 *Le problème de tautologie, "$\forall \mathcal{A}$ C est vrai ?", est dans co-$\mathcal{NP}$.*

Exemple 5 *"Est-ce qu'on* ne *peut* pas *colorier G avec k couleurs ?" est dans co-$\mathcal{NP}$.*

$\mathcal{NP} \cap$ co-$\mathcal{NP}$ n'est clairement pas vide puisqu'elle contient $\mathcal{P}$ mais est-ce que d'autres problèmes s'y trouvent ? On a su pendant longtemps que le problème de primalité, "Est-ce que n est premier ?", se trouvait dans cette intersection ; il a fallu attendre 2002 pour démontrer que ce problème était dans $\mathcal{P}$.

11.3 Complexité espace

D'autres classes sont définies non pas en fonction du temps de calcul mais plutôt de la ressource espace-mémoire. Un développement semblable à celui pour la ressource précédente fut entrepris ; nous ne donnons ici que quelques éléments.

La classe de complexité $\mathcal{PSPACE}$ est l'ensemble des problèmes de décision qu'on peut résoudre en utilisant une quantité d'espace-mémoire qui est polynomiale dans la taille de l'exemplaire.

La classe de complexité $\mathcal{LOGSPACE}$ est l'ensemble des problèmes de décision qu'on peut résoudre en utilisant une quantité d'espace-mémoire qui est logarithmique dans la taille de l'exemplaire.

Pour cette ressource, le non-déterminisme ne nous apporte rien :

$$\mathcal{PSPACE} = \mathcal{NPSPACE}.$$

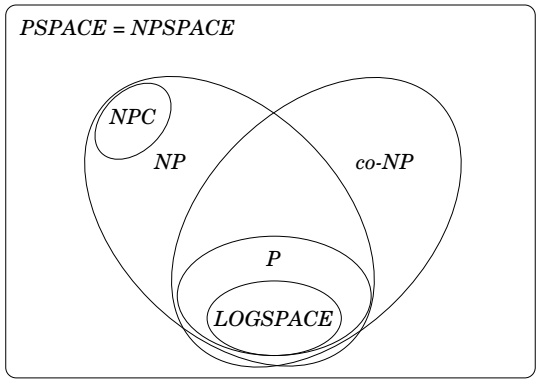
On sait que

$$\mathcal{LOGSPACE} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PSPACE}$$

et que

$$\mathcal{LOGSPACE} \subset \mathcal{PSPACE}.$$

11.4 Schéma récapitulatif



11.5 Problèmes indécidables

Les problèmes des classes précédentes semblent en fait bien faciles par rapport à d'autres qui sont carrément indécidables !

Un problème est *décidable* si on peut répondre par "oui" ou par "non" à chacun de ses exemplaires, peu importe la quantité de ressources consommées. Tous les problèmes rencontrés jusqu'à maintenant sont clairement décidables, même si répondre peut demander énormément de temps ou d'espace-mémoire. Le plus connu des problèmes qui ne soient pas décidables est le *problème d'arrêt*.

> $P_{\text{arrêt}}$: "Étant donné un programme P et une entrée e, est-ce que P terminera son exécution sur e ?"

Intuitivement, afin de connaître la réponse, il faudra simuler l'exécution de P sur e et s'il boucle, nous bouclerons aussi sans jamais pouvoir donner de réponse (on pourra répondre "oui" mais jamais "non").

Ce n'est pas le seul problème indécidable : il en existe d'autres assez naturels et on peut même en créer autant qu'on le désire, selon une certaine hiérarchie de classes d'indécidabilité (par exemple, la hiérarchie arithmétique de Kleene).

> $P_\emptyset$: "Étant donné un programme P, existe-t-il une entrée pour laquelle P répondra "oui" ?"

P_{fini} : “Étant donné un programme P, l'ensemble des entrées pour lesquelles P répondra “oui” est-il fini ?”

Problème de correspondance de Post : “Étant donné un ensemble fini de dominos, peut-on en faire une suite, en autorisant les répétitions, telle que le mot formé en haut et celui formé en bas sont les mêmes ?”