Calculabilité et Complexité (11X008) - Printemps 2024

10. NP-complétude

Enseignant: Arnaud Casteigts *Exercices: A. Berger, M. De Francesco, L. Heiniger*

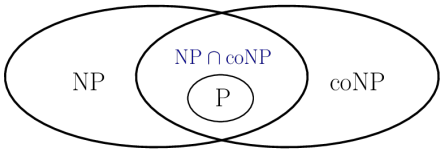
Nous connaissons maintenant les deux classes P et NP et avons vu que NP peut être définie de manière entièrement déterministe. Pour rappel :

- P : Langages qui peuvent être décidés en temps polynomial.
- NP : Langages qui admettent des preuves d'*appartenance* (certificats positifs) vérifiables en temps polynomial.

On peut définir au passage la classe co-NP :

- co-NP : Langages qui admettent des preuves de *non-appartenance* (certificats négatifs) vérifiables en temps polynomial.

Remarquez que si un langage L est dans NP, alors son complément $\overline{L}$ est dans co-NP et vice versa. Mais ces inclusions ne sont pas exclusives : un langage peut être dans les deux. D'ailleurs, on a clairement P $\subseteq$ NP et P $\subseteq$ co-NP, car un algorithme de décision peut être lui-même vu comme un vérifieur d'appartenance et de non-appartenance (sans certificat supplémentaire). La situation est illustrée ci-dessous :



Il existe aussi des problèmes dans NP qui ne sont pas (à notre connaissance) dans co-NP (et vice versa). Par exemple, pour 3-COLORING, il est facile de prouver qu'un graphe est 3-colorable (la coloration elle-même), mais beaucoup moins de prouver qu'il ne l'est pas ! Pour prouver qu'il ne l'est pas, on pourrait fournir un algorithme qui teste toutes les possibilités, mais la vérification ne serait pas polynomiale. Sauf surprise, 3-COLORING ne semble donc pas être dans co-NP. Enfin, on connaît des langages qui sont dans NP $\cap$ co-NP, mais potentiellement pas dans P (par exemple, FACTORING). Bien sûr, si P = NP, alors toutes ces classes s'écroulent et deviennent identiques à P.

Dans ce cours, nous allons nous limiter à nouveau à la classe NP seulement (ce que nous dirons peut être adapté symétriquement pour co-NP). Nous allons nous intéresser au sous-ensemble le plus "difficile" de NP, connu sous le nom de problèmes (ou langages) NP-complets. Mais avant, nous avons besoin d'un outil conceptuel pour comparer la difficulté.

10.1 Réductions polynomiales

Petit rappel de calculabilité (cours n°5) : pour montrer qu'un langage L est indécidable, il suffit de trouver un autre langage L' que l'on sait déjà indécidable, et montrer que L' peut *se réduire* à L (noté $L' \leq L$). Autrement dit, un hypothétique algorithme pour L permettrait de décider L'. Par conséquent, L est lui-aussi indécidable.

On peut adapter cette idée pour comparer des langages en termes de complexité. Pour simplifier, disons qu'un langage est "difficile" s'il n'est pas décidable en temps polynomial. Pour montrer qu'un langage L est difficile, il suffit alors de trouver un autre langage L' que l'on sait déjà difficile, et montrer que L' peut se réduire à L *en temps polynomial* (noté $L' \leq_p L$). Autrement dit, un hypothétique algorithme polynomial pour L permettrait (via cette réduction) de décider L' en temps polynomial. Par conséquent, L est lui-aussi difficile.

Voici la définition :

Un langage L' **se réduit en temps polynomial** à un langage L s'il existe une machine de Turing M (un algorithme) qui prend en entrée un mot w' (instance pour L') et produit en temps polynomial (en $|w'|$) un autre mot w (instance pour L) tel que $w' \in L' \iff w \in L$.

10.1.1 Exemple de réduction

Voici deux problèmes classiques en théorie des graphes.

Clique : Étant donné un graphe $G = (V, E)$ et un entier k, est-ce que G contient une clique de taille k ? Autrement dit, un sous-ensemble de sommets $V' \subseteq V$ tels que $|V'| = k$ et tous les sommets de V' sont voisins (reliés par une arête).

Par exemple, le graphe G (Figure 1) contient une clique de taille 4, mais aucune de taille 5.

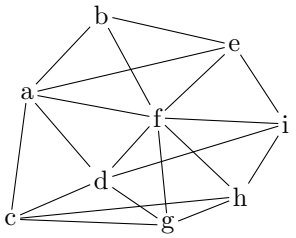


Figure 1 – Un graphe G.

Independent Set : Étant donné un graphe $G = (V, E)$ et un entier k, est-ce que G contient un ensemble indépendant de taille k ? Autrement dit, un sous-ensemble $V' \subseteq V$ de sommets tels que $|V'| = k$ et aucun des sommets de V' ne sont voisins.

Par exemple, le même graphe G de la Figure 1 contient un ensemble indépendant de taille 3, mais aucun de taille 4.

Theorème 10.1. CLIQUE $\leq_p$ INDEPENDENT SET

Démonstration. Étant donné une instance (G, k) de CLIQUE, on construit en temps polynomial un autre graphe $\overline{G}$, appelé le complément de G, tel qu'il existe une arête entre deux sommets de $\overline{G}$ si et seulement si il n'existe pas d'arête entre ces sommets dans G. Par construction, ce graphe contient un ensemble indépendant de taille k si et seulement si G contient une clique de taille k. □

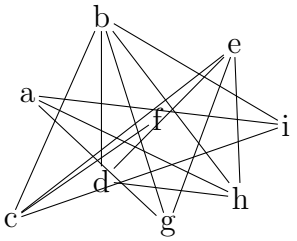


FIGURE 2 – Le graphe $\overline{G}$, complément de G.

Ainsi, nous avons montré que CLIQUE se réduit à INDEPENDENT SET, dans le sens où un algorithme polynomial pour le second implique un algorithme polynomial pour le premier. Cela établit que INDEPENDENT SET est "au moins aussi difficile" que CLIQUE.

Observez qu'une réduction similaire existe dans l'autre sens, de INDEPENDENT SET vers CLIQUE. Ces deux problèmes sont donc **polynomialement équivalents**, ce que l'on note CLIQUE $\equiv_p$ INDEPENDENT SET.

10.2 NP-complétude

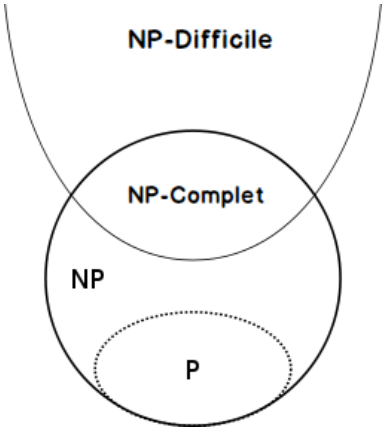
Nous y sommes presque, nous avons juste besoin d'une dernière notion. Un langage L est NP-difficile si *tous les langages* de NP peuvent s'y réduire en temps polynomial. Autrement dit, il est au moins aussi difficile que n'importe quel langage de NP. Il est logique qu'un tel langage puisse exister en dehors de NP, par exemple dans des classes de complexité qui autorisent des problèmes extrèmement difficiles (et d'ailleurs, pas forcément des problèmes de décision). Un fait surprenant de la théorie de la complexité est qu'il existe certains langages qui sont à la fois NP-difficile et dans NP, ce sont les fameux langages NP-complets.

Concrètement, un langage L est NP-complet si ces deux conditions sont réunies :

- L est dans NP,
- L est NP-difficile.

Par exemple, CLIQUE et INDEPENDENT SET sont des langages NP-complets, ainsi que SAT et 3-COLORING. En revanche, à notre connaissance, GRAPH ISOMORPHISM et FACTORING, bien que dans NP, ne semblent pas être ni NP-complets, ni dans P. On les appelle problèmes NP-intermédiaires.

Voici le diagramme d'inclusion des classes dont nous avons discutées :



Il nous reste à comprendre une chose essentielle : comment peut-on montrer qu'un langage L est NP-complet ? D'après la définition, il suffit de montrer que L est dans NP (nous savons déjà faire ça) et montrer qu'il est NP-difficile. Si l'on connaît déjà un autre langage L' qui est NP-difficile, il suffit de trouver une réduction polynomiale de L' vers L. Puisque L' est NP-difficile et qu'il se réduit à L, alors L doit être aussi NP-difficile et la boucle est bouclée.

Euh... mais alors, comment a-t-on pu montrer la première fois qu'un problème est NP-difficile, alors qu'aucun autre problème n'existait pour effectuer une réduction ? Hum...