Calculabilité et Complexité (11X008) - Printemps 2025

9. Réductions polynomiales

Enseignant: Arnaud Casteigts

Assistants: A. Berger, M. De Francesco
Moniteurs: E. Bussod, N. Beghdadi

Dans ce cours, nous continuons à étudier les problèmes/langages qui sont dans NP mais que l'on ne sait pas résoudre/décider en temps polynomial (ces problèmes sont donc potentiellement dans $\mathsf{NP} \setminus \mathsf{P}$). Nous allons donner plusieurs exemples de **réductions polynomiales** entre ces problèmes, qui montrent que bien qu'on ne sache pas les résoudre rapidement, il suffit de savoir en résoudre rapidement certains pour en résoudre rapidement d'autres.

9.1 Réductions en temps polynomial

Dans cette section, nous utilisons l'adjectif "rapide" comme synonyme de "en temps polynomial". C'est plus léger.

Bien que l'on ne sache pas résoudre rapidement certains problèmes de NP, on peut montrer que l'existence d'un algorithme rapide pour l'un impliquerait l'existence d'un algorithme rapide pour l'autre. L'idée est la même qu'en calculabilité : on suppose qu'on dispose d'un tel algorithme et on l'utilise pour résoudre un autre problème. La différence est que la réduction elle-même doit être rapide (sinon c'est triché).

Reprenons l'exemple de Clique et de Independent Set. Nous allons montrer que si un algorithme rapide existe pour Independent Set, alors cela nous permet de résoudre Clique rapidement. On dit aussi que Clique **se réduit en temps polynomial** à Independent Set, noté Clique $\leq_p$ Independent Set.

Theorème 9.1. Clique $\leq_p$ Independent Set

Démonstration. Supposons que l'on dispose d'un algorithme rapide pour Independent Set. Étant donné une instance (G, k) de Clique, on peut construire en temps polynomial un autre graphe G', appelé le complément de G, tel qu'il existe une arête entre deux sommets de G' si et seulement si il n'existe pas d'arête entre ces sommets dans G. Par construction, G contient une clique de taille k si et seulement si G' contient un ensemble indépendant de taille k (voir l'exemple de la Figure 1). On peut donc utiliser l'algorithme pour Independent Set afin de répondre à la question. □

Voici la même réduction donnée sous forme de pseudo code :

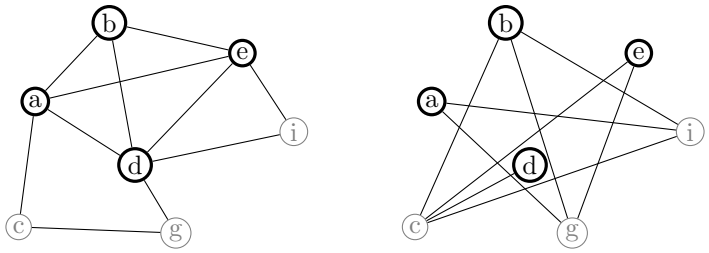


FIGURE 1 – Un graphe G et son complément G'. Les sommets $\{a, b, d, e\}$ forment une clique dans G et un ensemble indépendant dans G'.

```
def has_clique(G, k):
   G' = complement_of(G)  // rapide à calculer
   return has_independent_set(G', k)
```

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.

9.2 Exemples de réductions

Nous avons déjà donné un exemple ci-dessus. Trouver une réduction est parfois simple, et parfois plus compliqué. Tout dépend à quel point les problèmes sont proches l'un de l'autre conceptuellement. Nous allons donner quelques exemples, dont certains sont simples et d'autres moins.

9.2.1 CLIQUE $\leq_p$ SUBGRAPH ISOMORPHISM

Définition des problèmes :

- CLIQUE : Étant donné un graphe G et un entier k, G contient-il une clique de taille k ?

- SUBGRAPH ISOMORPHISM : Étant donnés deux graphes G_1 et G_2, est-ce que G_1 contient G_2 comme sous-graphe ? *(pas tout à fait le même problème que* GRAPH ISOMORPHISM*)*

Réduction : Supposons qu'il existe un algorithme rapide pour SUBGRAPH ISOMORPHISM. Étant donné une entrée (G, k) pour le problème CLIQUE, on construit un graphe G' ayant k sommets et toutes les arêtes possibles entre eux (autrement dit, un *graphe complet*). Puis on demande à l'algorithme pour SUBGRAPH ISOMORPHISM si G contient G' comme sous-graphe. Clairement, G' est un sous-graphe de G si et seulement si G contient une clique de taille k, on peut donc résoudre CLIQUE rapidement en utilisant cet algorithme. □

Pseudo-code :

```
def has_clique(G, k):
   G' = graphe_complet(k)  // rapide à calculer
   return has_subgraph(G, G')
```

9.2.2 Hamiltonian Cycle $\leq_p$ Subgraph Isomorphism

Hamiltonian Cycle : Étant donné un graphe G, existe-t-il un chemin qui visite les n sommets du graphe exactement une fois, puis termine au point de départ ?

Réduction : Même réduction que pour Clique $\leq_p$ Subgraph Isomorphism, sauf que le graphe G' construit est un *cycle* de n sommets au lieu d'un graphe complet. □

9.2.3 3-Coloring $\leq_p$ 4-Coloring

Définition des problèmes :

- 3-Coloring : Étant donné un graphe G, peut-on colorier ses sommets en utilisant 3 couleurs sans donner la même couleur à des sommets voisins ?
- 4-Coloring : Idem, avec 4 couleurs.

Réduction : Supposons qu'il existe un algorithme rapide pour 4-Coloring. Étant donné un graphe G dont on veut savoir s'il est 3-colorable, on construit un graphe G' comme une copie de G, à laquelle on ajoute un nouveau sommet que l'on relie à tous les autres (sommet universel). Ce sommet étant relié à tous les autres, il doit avoir une couleur unique. Du coup, G' est 4-colorable si et seulement si G est 3-colorable. Il suffit donc de poser cette question pour G' à l'algorithme pour 4-Coloring afin de savoir si G est 3-colorable. □

Pseudo-code :

```
def is_3_colorable(G):
   G' = copie de G + un sommet universel       // rapide à calculer
   return is_4_colorable(G')
```

9.2.4 SAT $\leq_p$ Clique

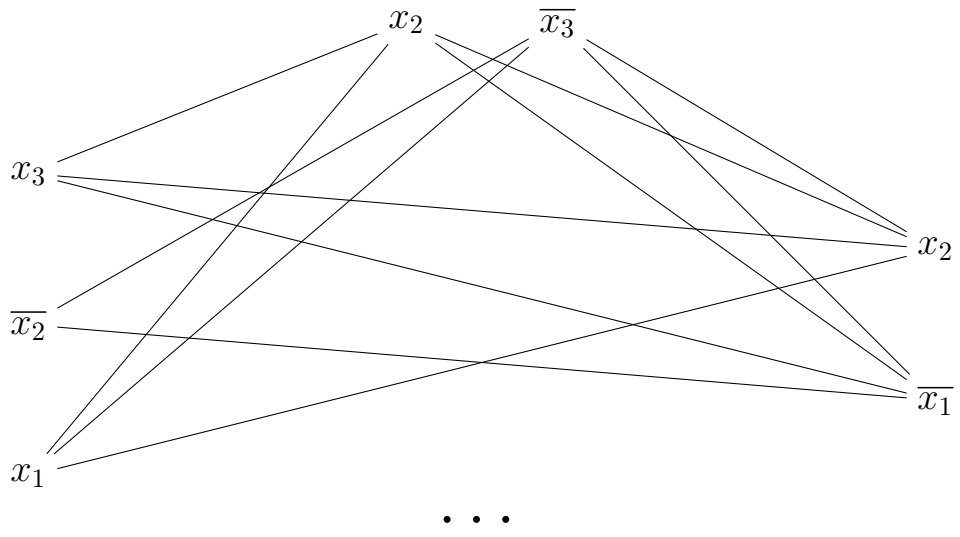
SAT : Étant donné une formule booléenne ϕ sous forme normale conjonctive, c.à.d. une conjonction de disjonction de *littéraux* (variables ou leur négation), par exemple :

$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_3})$ ($\overline{x_i}$ est la négation de x_i)

Existe-t-il des valeurs `vrai`/`faux` pour chaque variable x_i telles que la formule est satisfaite ? Chaque disjonction de la formule, par exemple $(x_1 \vee \overline{x_2} \vee x_4)$, est appelée une *clause*. L'exemple de formule ci-dessus a donc 2 clauses, 3 variables, et 5 litéraux. En l'occurrence, elle peut être satisfaite en mettant (p.ex.) toutes les variables à `vrai`. Un exemple de formule non-satisfaisable est $\phi_2 = (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2})$.

Nous allons montrer que s'il existe un algorithme rapide pour CLIQUE, alors on peut résoudre SAT rapidement. Il s'agit de notre première réduction non-triviale.

Réduction : Supposons que l'on dispose d'un algorithme rapide pour CLIQUE. Étant donné une formule, par exemple $\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2) \wedge \cdots$ comportant k clauses, on construit un graphe G comme suit : d'abord, on crée un sommet pour chaque litéral de chaque clause. Puis on relie chaque litéral avec les litéraux des *autres* clauses qui sont *compatibles* (par exemple, on ne relie pas x_1 et $\overline{x_1}$, mais on relie x_1 et x_2, ainsi que x_2 et x_2). Cela donne le graphe suivant :



Théorème : G admet une clique de taille k si et seulement si ϕ est satisfaisable.

Preuve : Pour démontrer ce "si et seulement si", il faut montrer les deux sens : (1) une clique de taille k implique l'existence d'une solution pour ϕ, et (2) une solution pour ϕ implique l'existence d'une clique de taille k.

(1) : Chaque litéral n'est relié qu'à des litéraux d'autres clauses, donc une clique de taille k doit faire intervenir un litéral dans chaque clause. Par ailleurs, ces litéraux sont tous compatibles entre eux (car ils partagent une arête deux à deux). Une clique de taille k donne donc bien une solution pour ϕ. Par exemple, la clique x_3 (gauche), x_2 (haut) et x_2 (droite) indique que les trois clauses peuvent être satisfaites en mettant x_3 et x_2 à `vrai` (x_1 étant quelconque). De même, la clique x_3 (gauche), x_2 (haut), $\overline{x_1}$ (droite) indique que les trois clauses peuvent être satisfaites en mettant x_3 et x_2 à `vrai` et x_1 à `faux`.

(2) : Si une solution pour ϕ existe, alors il existe une affectation pour les variables qui satisfait chacune des k clauses de ϕ. Chacune de ces clauses est satisfaite via un ou plusieurs litéraux. On peut choisir arbitrairement un litéral satisfait pour chaque clause. Cet ensemble de litéraux est nécessairement compatible, donc les sommets correspondants forment une clique dans G. □

9.2.5 3-SAT $\leq_p$ 3-Coloring

Une réduction assez astucieuse (pour ne pas dire géniale), que vous verrez en exercices.

9.2.6 Résumé des réductions

- Clique $\leq_p$ Independent Set
- Independent Set $\leq_p$ Clique
- Clique $\leq_p$ Subgraph Isomorphism
- Hamiltonian Cycle $\leq_p$ Subgraph Isomorphism
- 3-Coloring $\leq_p$ 4-Coloring
- SAT $\leq_p$ Clique
- SAT $\leq_p$ 3-SAT (prochain cours)
- 3-SAT $\leq_p$ 3-Coloring (en séance d'exercices)

9.2.7 Définition formelle d'une réduction polynomiale (pour info)

Un langage L **se réduit en temps polynomial** à un langage L' s'il existe une machine de Turing M qui prend en entrée un mot w dont on veut décider l'appartenance à L, et produit en temps polynomial (en $|w|$) un autre mot w' tel que $w' \in L' \iff w \in L$.