

9. Réductions polynomiales

Enseignant: Arnaud Casteigts

Assistants: M. De Francesco, M. Marsello

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 $\text{NP} \setminus \text{P}$). 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 Rappel de quelques problèmes

- **3-COLORING** : Étant donné un graphe G , les sommets de G peuvent-ils être coloriés avec 3 couleurs de sorte que tous les sommets voisins ont des couleurs différentes.
- **4-COLORING** : Même chose avec 4 couleurs.
- **CLIQUE** : Étant donné un graphe G et un entier k , est-ce que G contient k sommets qui sont tous voisins ?
- **INDEPENDENT SET** : Étant donné un graphe G et un entier k , est-ce que G contient k sommets qui ne sont pas voisins ?
- **HAMILTONIAN CYCLE** : Étant donné un graphe G , existe-t-il un chemin qui visite tous les sommets exactement une fois, puis termine au point de départ ?
- **SUBGRAPH ISOMORPHISM** : Étant donnés deux graphes G_1 et G_2 , est-ce que G_1 contient un graphe identique à G_2 comme sous-graphe ?
- **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 :

$$\phi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee \bar{x}_3) \quad (\bar{x}_i \text{ est la négation de } x_i)$$

Existe-t-il des valeurs **vrai/faux** pour chaque variable x_i telles que la formule est satisfaite? Chaque disjonction, par exemple $(x_1 \vee \bar{x}_2 \vee x_4)$, est appelée une *clause*. L'exemple de formule ci-dessus a donc 2 clauses, 3 variables, et 5 littéraux. En l'occurrence, elle peut être satisfaite en mettant (p.ex.) toutes les variables à **vrai**. Un exemple de formule non-satisfaisable est $\phi_2 = (x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$.

- **3-SAT** : Même chose en supposant que chaque clause a exactement trois littéraux.

9.2 Réductions polynomiales

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 $\text{CLIQUE} \leq_p \text{INDEPENDENT SET}$.

Theorème 9.1. $\text{CLIQUE} \leq_p \text{INDEPENDENT SET}$

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. \square

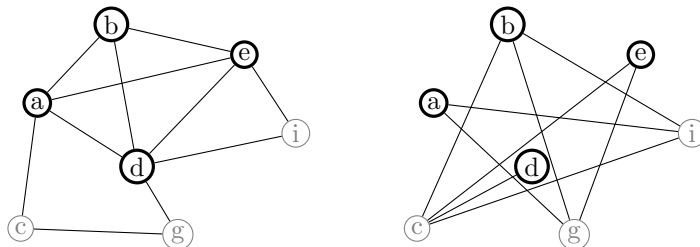


FIGURE 1 – Un graphe G et son complément G' . Les sommets $\{a, b, d, e\}$ forment une clique dans G et un ensemble indépendant dans G' .

Voici la même réduction donnée sous forme de pseudo code :

```
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 $\text{CLIQUE} \equiv_p \text{INDEPENDENT SET}$.

9.3 Autres exemples

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.

9.3.1 CLIQUE \leq_p SUBGRAPH ISOMORPHISM

Réduction : Soit A un algorithme pour SUBGRAPH ISOMORPHISM. Si l'on a accès à A , on peut concevoir un algorithme B pour CLIQUE comme suit. L'entrée de B est (G, k) (un graphe G et un entier k), notre algorithme B consiste à construire un graphe G' ayant k sommets et toutes les arêtes possibles entre eux (un "graphe complet" à k sommets). Puis on demande à l'algorithme A si G contient G' comme sous-graphe. Clairement, G contient une clique de taille k si et seulement si G' est un sous-graphe de G . La réponse de A nous donne donc bien la réponse pour B . \square

Pseudo-code :

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

9.3.2 HAMILTONIAN CYCLE \leq_p SUBGRAPH ISOMORPHISM

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. \square

9.3.3 3-COLORING \leq_p 4-COLORING

Réduction : Supposons qu'il existe un algorithme 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. \square

Pseudo-code :

```

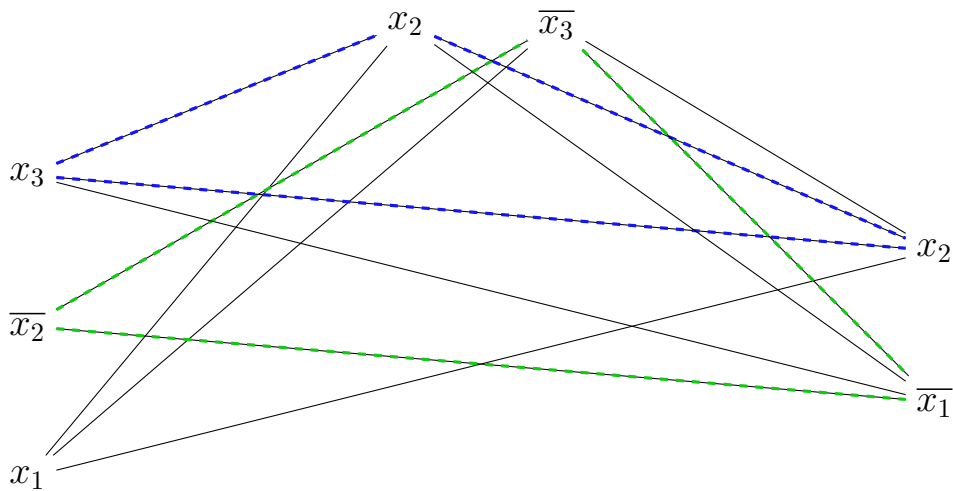
def is_3_colorable(G):
    G' = copie de G + un sommet universel           // rapide à construire
    return is_4_colorable(G')

```

9.3.4 SAT \leq_p CLIQUE

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 SAT, par exemple $\phi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2)$ comportant k clauses (ici $k = 3$), on construit un graphe G comme suit : d'abord, on crée un sommet pour chaque littéral de chaque clause. Puis on relie chaque littéral avec les littéraux des *autres* clauses qui sont *compatibles* (c.à.d. qui ne se contredisent pas, par exemple, x_1 et \bar{x}_1 se contredisent). 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 littéral n'est relié qu'à des littéraux d'*autres* clauses, donc une clique de taille k doit faire intervenir un littéral dans chaque clause. Ces littéraux étant compatibles, ils nous indiquent quelles valeurs affecter aux variables correspondantes pour satisfaire chaque clause. 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 \bar{x}_2 (gauche), \bar{x}_3 (haut) et \bar{x}_1 (droite) indique que les trois clauses peuvent être satisfaites en mettant toutes les variables à **faux**. Il y a plusieurs solutions.

(2) : Si une solution pour ϕ existe, par définition chacune des k clauses a au moins un littéral satisfait par cette solution et ces littéraux sont compatibles. Ils correspondent donc à une clique de taille k dans G . \square

9.3.5 SAT \leq_p 3-SAT

Une réduction que nous verrons dans un prochain cours.

9.3.6 3-SAT \leq_p 3-COLORING

Une réduction très astucieuse que vous verrez en exercices.

9.3.7 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)

9.3.8 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 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 $w' \in L' \iff w \in L$.

Subtilité (hors programme) : il existe en fait plusieurs notions de réductions polynomiales, plus ou moins strictes. La plus stricte s'appelle *Réduction de Karp* et correspond à la définition ci-dessus. Dans cette réduction, on ne peut utiliser une machine pour L' qu'une seule fois à la fin de la réduction. Une notion moins forte est celle de *Réduction de Turing*, qui permet d'utiliser une potentielle machine pour L' plusieurs fois dans une machine qui déciderait L . Nous ne savons pas si les deux notions sont équivalentes, c'est une question ouverte. Nous utiliserons l'une ou l'autre de manière interchangeable.