Équivalence entre problèmes de graphes
Fiche scientifique

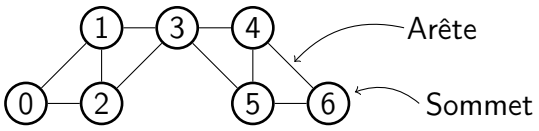
L'objectif de ce document est de présenter quelques problèmes de graphes, afin d'illustrer le concept d'équivalence et de réductions entre problèmes algorithmiques.

1 Introduction

1.1 Les graphes : histoire et modèle

Un graphe est, sous sa forme la plus simple, un objet mathématique permettant de modéliser des entités et les relations entre celles-ci. À ce titre, il est difficile d'en retracer un historique complet. On remonte souvent l'étude des graphes à Leonhardt Euler et à son fameux problème dit des ponts de Königsberg, publié en 1741 [Eul41]. Le terme de "graphe" ("graph", en anglais) fut utilisé pour la première fois par James Sylvester en 1878 [Syl78]. Le premier livre traitant de théorie des graphes fut publié en 1936 par le mathématicien hongrois Dénes Kőnig [K̋36]. En France, Claude Berge, grand nom de la théorie des graphes, a publié un livre majeur sur le sujet en 1958 [Ber58]. Un livre du mathématicien américain Frank Harary de 1969 a aussi eu une influence importante dans le domaine [Har69]. Le fameux mathématicien hongrois Paul Erdős a notamment travaillé sur la théorie des graphes. L'essor de la théorie des graphes va de pair avec l'essor de l'informatique à partir des années 50, et c'est devenu une thématique classique autant en informatique qu'en mathématiques.
On définit cette notion de la façon suivante :

Un **graphe** G est donné par un ensemble de **sommets** V et un ensemble d'**arêtes** $E \subseteq V^2$.



Le graphe ci-dessus a pour sommets $V = \{0, 1, 2, 3, 4, 5, 6\}$ et pour arêtes
$E = \big\{\{0,1\}, \{0,2\}, \{1,2\}, \{1,3\}, \{2,3\}, \{3,4\}, \{3,5\}, \{4,5\}, \{4,6\}, \{5,6\}\big\}$.

À noter qu'on considère ici les graphes *simples*, *non-orientés* et *non-pondérés*.Sommets et arêtes peuvent représenter des objets concrets ou des notions abstraites. Par exemple, dans une ville avec de nombreux canaux et rivières, on pourra représenter les îles par des sommets, et mettre des arêtes entre deux sommets si les îles qu'ils représentent sont reliées par un pont. Deuxième exemple : dans un réseau social, on représente les utilisateurs par des sommets, et on met une arête entre deux sommets si les utilisateurs sont amis. Troisième exemple : dans un réseau informatique, les objets (ordinateurs, routeurs, etc) sont représentés par des sommets, et on met une arête entre deux sommets si les deux objets sont directement connectés (par exemple via un câble réseau).
Quelques définitions importantes :

- Deux sommets u et v sont dits **adjacents** ou **voisins** s'il existe une arête $\{u, v\}$. Dans le cas contraire, on dit qu'ils sont **non-adjacents**.
- Pour une arête $\{u, v\}$, les sommets u et v sont appelés ses **extrémités**.

1.2 Des problèmes classiques en théorie des graphes

Les graphes étant un modèle très générique pour des entités en relation, on peut les utiliser pour résoudre des problèmes concrets sur ce modèle abstrait, de façon à ne pas inventer plusieurs fois le même algorithme pour résoudre des problèmes qui sont superficiellement différents mais fondamentalement les mêmes. Reprenons des exemples précédents : que l'on cherche un groupe d'amis mutuels dans un réseau social (donc des gens tous amis les uns avec les autres) ou un cluster d'ordinateurs dans un réseau informatique (donc des ordinateurs tous connectés entre eux), passer au modèle graphe permet de remarquer que l'on cherche dans les deux cas un ensemble de sommets qui sont tous adjacents les uns aux autres.
Ainsi, à travers l'histoire de la théorie des graphes, de nombreux problèmes ont été définis, lesquels permettent d'abstraire des situations concrètes à l'aide du modèle, et de les résoudre. Cette activité propose de découvrir trois problèmes très connus et les liens existant entre eux : l'ENSEMBLE INDÉPENDANT (Section 2.1), la CLIQUE (Section 2.2) et la COUVERTURE PAR SOMMETS (Section 2.3). D'autres problèmes classiques sont la COLORATION, la DOMINATION, le PLUS COURT CHEMIN ou le CHEMIN HAMILTONIEN.

1.3 Complexité algorithmique et réductions

Quand on veut résoudre un problème concret à l'aide des graphes, on commence donc par abstraire notre situation avec le modèle graphe (les entités sont les sommets, les relations sont les arêtes) et on trouve le problème classique qui nous permettra de répondre à notre question initiale. Cependant, il s'agit aussi de pouvoir résoudre ledit problème ! C'est là qu'entrent en jeu les notions d'*algorithme* et de *complexité algorithmique*. Le but de cette activité n'est pas de donner des définitions exhaustives de ces concepts, mais d'en avoir une compréhension suffisante.

Un **problème algorithmique** définit formellement une tâche qui doit être réalisée (généralement par un ordinateur). Cette définition décrit la forme des **entrées** attendues du problème (c'est-à-dire, les données sur lesquelles porte le problème), et la tâche à effectuer, sous la forme d'une **sortie** qui représente le résultat attendu.
Un **algorithme** est une méthode abstraite et générique de résolution d'un problème algorithmique. Il a une entrée, qui correspond à l'entrée du problème que résout l'algorithme, et effectue une série d'opérations utilisant cette entrée, pour produire la sortie. On parle de méthode "abstraite" car un algorithme est indépendant de son implémentation concrète et de méthode "générique" car un algorithme doit fonctionner (c'est-à-dire donner une réponse correcte) sur toutes les entrées qui vérifient ses conditions de fonctionnement.

La complexité algorithmique recouvre deux notions différentes, selon que l'on parle d'un algorithme ou d'un problème.

- La **complexité d'un algorithme** est le nombre d'*instructions élémentaires* que celui-ci effectue, et s'exprime en fonction de la taille de l'entrée. On considère généralement qu'une instruction élémentaire est une opération très simplement réalisable par un processeur électronique (l'élément de base permettant de réaliser des calculs dans un ordinateur, un smartphone, etc) : une opération arithmétique simple (addition par exemple), une comparaison, une affectation (donner une valeur à une variable)... Cette complexité dépend donc généralement de la taille de l'entrée : si l'on souhaite trier un ensemble de nombres, il est normal d'effectuer plus de comparaisons lorsque la taille de l'ensemble augmente ! Par ailleurs, on considère généralement la complexité **dans le pire cas**, c'est-à-dire que, pour une taille d'entrées donnée, la complexité de l'algorithme pour cette taille est le plus grand nombre d'instructions élémentaires effectuées par l'algorithme, parmi toutes les entrées possibles de cette taille.
- La **complexité d'un problème algorithmique** est, informellement, vue comme la meilleure (c'est-à-dire la plus basse) complexité possible parmi tous les algorithmes résolvant le problème.
- En général, on tend à ne garder d'une complexité que son facteur principal, c'est-à-dire la valeur qui sera la plus importante pour exprimer la croissance du nombre d'instructions en fonction de la taille de l'entrée. En effet, c'est ce facteur principal qui permettra d'évaluer la vitesse d'exécution d'un algorithme.
  On utilise pour cela des notations asymptotiques formalisées par Donald Knuth en 1976 [Knu76]. La plus célèbre décrit la **domination**, via la notation $\mathcal{O}$. De façon formelle, pour deux fonctions f et g, on a :
  $$f(x) = \mathcal{O}(g(x)) \;\Leftrightarrow\; \exists C, N \in \mathbb{R} :\; \forall x > N, |f(x)| \leq C|g(x)|.$$
  En d'autres termes : à partir d'un certain seuil, la fonction f est bornée par une constante multipliée par la fonction g, donc la fonction g domine la fonction f. Cela permet notamment d'ignorer les constantes multiplicatives et additives.
  Premier exemple : une complexité de $8n^3 + 49n^2 + 28n + 256$ sera écrite $\mathcal{O}(n^3)$, car n^3 est le facteur principal lorsque n devient suffisamment grand. Deuxième exemple : une complexité de $1.1^n + 4587n^{27} + 12n^6$ sera écrite $\mathcal{O}(1.1^n)$, le facteur principal étant 1.1^n lorsque n devient suffisamment grand.
- Quand on veut être plus précis et ne pas simplement savoir ce qui domine la complexité de l'algorithme, mais de la comprendre aussi finement que possible, on utilise la notion d'**équivalence**, exprimée via la notation Θ. Informellement, $f(x) = \Theta(g(x))$ signifie que $f(x) = \mathcal{O}(g(x))$ **et** que $g(x) = \mathcal{O}(f(x))$. De façon formelle, pour deux fonctions f et g :
  $$f(x) = \Theta(g(x)) \;\Leftrightarrow\; \exists C_1, C_2, N \in \mathbb{R} :\; \forall x > N, C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|.$$
  Par exemple, une complexité de $0.2n^8 + 16n^5 + 2n^2 + 41$ sera écrite $\Theta(n^8)$. La différence avec la notation précédente est que cette complexité est $\mathcal{O}(n^8)$ mais aussi $\mathcal{O}(n^9)$ ou $\mathcal{O}(n^{150})$, car la notation $\mathcal{O}$ est une borne supérieure alors que la notation Θ est une borne tant supérieure qu'inférieure.
- On regroupe généralement différentes complexités algorithmiques différentes mais équivalentes sous le même terme pour exprimer leur qualité globale. On parle d'**algorithme polynomial** lorsque la complexité de l'algorithme est de $\Theta(n^C)$ pour un C constant. On parle

d'**algorithme exponentiel** lorsque la complexité de l'algorithme est de $\Theta(C^n)$ pour un C constant. Attention, ces C doivent bien être des constantes absolues, et ne pas dépendre de la taille n de l'entrée.

On va généralement considérer qu'un algorithme polynomial constitue un algorithme efficace en pratique, alors qu'un algorithme exponentiel est trop coûteux (cf. le tableau ci-dessous pour une progression selon la valeur de n). Ceci a été postulé par Alan Cobham et Jack Edmonds dans les années 1960 [Cob65, Edm65]. Bien sûr, dans la réalité, selon la taille des entrées, le temps de calcul peut être moins grand dans une complexité théoriquement plus élevée selon les valeurs de constantes (par exemple, dans le tableau ci-dessous, on voit que 2^n est plus efficace que $100n$ pour de petites valeurs de n). Mais, en algorithmique, on considère que les entrées peuvent être de tailles arbitrairement grandes, et donc un algorithme polynomial restera raisonnablement exécutable tandis que le temps de calcul d'un algorithme exponentiel va exploser beaucoup plus vite, et l'algorithme ne pourra pas être exécuté sur une entrée de grande taille.

n	$\log(n)$	$100n$	n^2	n^3	2^n
2	1	200	4	8	4
10	≈ 3.32	1000	100	1000	1024
100	≈ 6.64	10.000	10.000	1.000.000	30 chiffres
1.000	≈ 9.97	100.000	1.000.000	10 chiffres	302 chiffres

Prenons un exemple simple : on cherche à trouver le plus petit élément d'un ensemble S de n nombres. Un algorithme pour résoudre le problème consiste à initialiser une variable x avec le premier élément de S, puis à comparer x avec chaque élément y de S ; si $y < x$, on affecte à x la valeur de y. Lorsque tous les éléments de S ont été considérés, on répond alors x. Cet algorithme effectue $n-1$ comparaisons et au plus n affectations, et a donc une complexité d'au plus $2n-1$, donc $\mathcal{O}(n)$, soit une complexité *linéaire* (en d'autres termes, quand la taille de l'entrée augmente, la complexité de l'algorithme augmente de façon proportionnelle en parallèle). Comme il est nécessaire de s'assurer qu'un élément donné en réponse est bien le plus petit de S, un algorithme répondant au problème du plus petit élément fera au moins $n-1$ comparaisons, et aura donc une complexité au moins linéaire. Cela signifie que le problème de trouver le plus petit élément d'un ensemble de nombres a une complexité linéaire : il est impossible de faire mieux, et on sait le faire en temps linéaire.

Il arrive que des problèmes soient fortement liés, c'est-à-dire qu'on puisse transformer l'entrée d'un problème pour obtenir une entrée d'un autre problème, et que la résolution de ce dernier nous permette de résoudre le problème de départ. Cela sera défini de manière plus concrète dans la Section 3.1. On appelle une telle transformation une **réduction** entre deux problèmes algorithmiques, et on s'en sert souvent pour montrer l'**équivalence** entre deux problèmes (c'est-à-dire qu'ils ont la même complexité). Cela permet de résoudre un problème en s'aidant d'un algorithme résolvant un autre problème.

2 Les problèmes de l'activité

Avant toute chose, il convient de définir formellement le type de problèmes algorithmiques que nous allons considérer :

> Un **problème d'optimisation** est un problème dans lequel on essaye de minimiser ou maximiser une certaine fonction objectif, c'est-à-dire que la réponse doit être un objet ou un ensemble d'objets dont on va essayer de minimiser ou maximiser la valeur ou la taille.

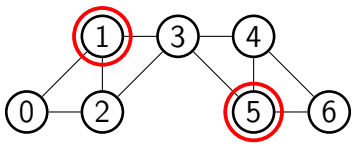
Par exemple :

- Étant donné deux nombres x et y, donner le nombre $z \in [x, y]$ tel que $f(z)$ (pour une fonction f donnée) soit le plus grand possible.
- Étant donnés un ensemble S de nombres et x un nombre, donner le plus possible de nombres de S plus petits que x.
- Étant donné un graphe G, donner le plus grand ensemble possible de sommets sans voisin.

2.1 Ensemble indépendant

> Dans un graphe, un **ensemble indépendant** (parfois aussi appelé *stable*) est un ensemble de sommets qui sont deux à deux non-adjacents.

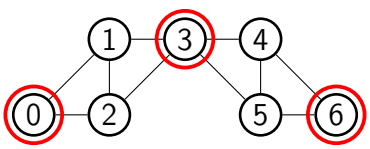
Par exemple, dans le graphe présenté en Section 1.1, nous pouvons aisément identifier un ensemble indépendant de taille 2 (ici l'ensemble $\{1, 5\}$) :



La notion d'ensemble indépendant permet de modéliser des problèmes où l'on cherche des entités n'ayant pas de relations entre elles. Par exemple, si les sommets représentent des individus et les arêtes des conflits, et qu'on veut organiser un événement sans conflit, on cherchera un ensemble indépendant. Le problème d'optimisation correspondant est le suivant :

> Le problème de l'ENSEMBLE INDÉPENDANT consiste à trouver un ensemble indépendant de taille la plus grande possible dans le graphe donné en entrée.

Par exemple, toujours dans le même graphe, l'ensemble indépendant le plus grand possible est de taille 3 (l'ensemble $\{0, 3, 6\}$) :

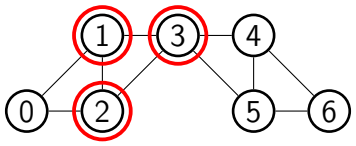


Dans ce graphe, il n'y a qu'un ensemble indépendant de taille 3, mais il peut bien sûr exister dans certains graphes plusieurs ensembles indépendants de taille maximum.
Notons que, s'il est facile de vérifier qu'un ensemble de sommets donné est bien un ensemble indépendant (il suffit de vérifier qu'aucun sommet proposé n'est adjacent à un autre), il est beaucoup plus difficile de trouver le plus grand possible ! Dans cette activité, nous donnons les solutions et il faudra admettre leur exactitude (ou passer du temps à vérifier tous les ensembles de taille supérieure à la main : si aucun n'est un ensemble indépendant, alors, c'est que nous avons bien le plus grand possible).

2.2 Clique

Dans un graphe, une **clique** est un ensemble de sommets qui sont tous adjacents deux à deux.

Notons que chaque arête d'un graphe est une clique (de taille 2) ! Dans le graphe présenté au début, il est aisé d'identifier des cliques de taille 3 (ici l'ensemble $\{1, 2, 3\}$) :



La notion de clique permet de modéliser des problèmes où l'on cherche des entités étant toutes en relation. Pour reprendre des exemples décrits plus haut, chercher un groupe d'amis mutuels sur un réseau social ou un cluster d'ordinateurs dans un réseau local revient à chercher une clique. Le problème d'optimisation est le suivant :

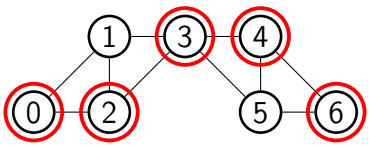
Le problème de la CLIQUE consiste à trouver une clique de taille la plus grande possible dans le graphe donné en entrée.

Dans le graphe ci-dessus, la clique de taille 3 est la plus grande que l'on puisse trouver. Notons qu'il y en a plusieurs : les solutions ne sont pas forcément uniques ! Comme pour l'ensemble indépendant, il est facile de vérifier qu'un ensemble de sommets donné est bien une clique, mais bien plus difficile de trouver la plus grande possible.

2.3 Couverture par sommets

Dans un graphe, une **couverture par sommets** est un ensemble de sommets tel que, pour toute arête du graphe, au moins une de ses extrémités appartient à l'ensemble.

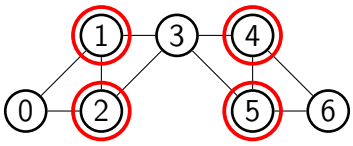
On dit qu'un sommet *couvre* les arêtes dont il est l'extrémité, et donc cette notion peut être reformulée : une couverture par sommets est un ensemble de sommets qui, pris tous ensemble, couvrent toutes les arêtes du graphe.
Par exemple, dans le graphe présenté au début, nous pouvons trouver une couverture par sommets de taille 5 (ici l'ensemble $\{0, 2, 3, 4, 6\}$) :



La notion de couverture par sommets permet de modéliser des problèmes où l'on cherche des entités couvrant toutes les relations existantes. Par exemple, si les sommets représentent des villes et les arêtes des routes, et qu'on veut placer des véhicules de secours dans des villes pouvant intervenir sur n'importe quelle route, on cherchera une couverture par sommets. Le problème d'optimisation associé est le suivant :

> Le problème de la COUVERTURE PAR SOMMETS consiste à trouver une couverture par sommets de taille la plus petite possible dans le graphe donné en entrée.

Par exemple, dans le graphe présenté au début, la couverture par sommets la plus petite possible est de taille 4 (l'ensemble $\{1, 2, 4, 5\}$) :



Là encore, il est facile de vérifier qu'un ensemble de sommets donné est bien une couverture par sommets, mais bien plus difficile de trouver la plus petite possible.

3 Réductions

3.1 Principe des réductions algorithmiques

Les trois problèmes définis ci-dessus partagent des caractéristiques communes : il est facile de vérifier si une solution donnée est correcte (dans le sens où elle vérifie la définition), mais difficile de maximiser ou minimiser sa taille. De plus, il semble y avoir des liens fondamentaux entre les problèmes. Cela paraît évident pour les problèmes de la clique et de l'ensemble indépendant : l'un cherche à trouver de nombreux sommets adjacents deux à deux et l'autre à trouver de nombreux sommets non-adjacents deux à deux. Notre objectif ici est d'expliciter le lien entre différents problèmes.
On appelle **réduction algorithmique** une façon de transformer un problème en un autre. De façon formelle :

> Une **réduction** du problème P au problème Q est un premier algorithme (appelé **algorithme aller**) transformant toute entrée e_P de P en une entrée e_Q de Q, et un second algorithme (appelé **algorithme retour**) pour transformer une solution de Q sur e_Q en une solution de P sur e_P.
> On note alors $P \leq Q$.

Ainsi, une réduction de P à Q est un outil algorithmique qui permet de résoudre le problème P, en supposant qu'on sache déjà résoudre le problème Q. En effet, imaginons que l'on sache résoudre Q sur n'importe quelle entrée, que $P \leq Q$ et qu'on sache exécuter les algorithmes aller et retour de la réduction de façon efficace. Maintenant, considérons le problème P : sur une entrée donnée e_P, on applique l'algorithme aller pour obtenir e_Q, on utilise notre méthode de résolution de Q sur e_Q, et on transfère la solution à P sur e_P en utilisant l'algorithme retour. Cela signifie que *savoir résoudre* Q *permet de résoudre* P, donc P n'est pas plus complexe que Q. Ainsi, la réduction est un processus permettant de comparer des problèmes en terme de complexité. En ce sens, la notation $P \leq Q$ est bien une relation d'ordre : le problème Q est au moins aussi complexe que le problème P.
Il est essentiel de noter que, pour les problèmes d'optimisation, une réduction de P à Q implique donc l'existence de deux algorithmes : un pour transformer les *entrées* (de P vers Q) et un pour transformer les *résultats* (de Q vers P).
Prenons un exemple concret : la multiplication et l'élévation au carré. La multiplication a deux entrées x et y et renvoie $x \times y$. L'élévation au carré a une entrée x et renvoie x^2. Soit une entrée x du problème de l'élévation au carré ; on forme l'entrée (x, x) du problème de la multiplication. Le résultat est conservé. Ceci est une réduction de l'élévation au carré à la multiplication, donc la multiplication est au moins aussi complexe que l'élévation au carré.
Une autre utilisation de la réduction est de montrer que plusieurs problèmes ont en fait la même difficulté à facteur polynomial près. Pour cela, on utilise la **réduction polynomiale** :

Une **réduction polynomiale** du problème P au problème Q est une réduction de P à Q telle que les algorithmes aller et retour s'effectuent en temps polynomial en la taille de e_P (cela implique en particulier que la taille de e_Q est polynomiale en la taille de e_P).
On note alors $P \leq_p Q$.

Si $P \leq_p Q$ et que Q se résout en temps polynomial, alors, P se résout aussi en temps polynomial. En effet, si l'on sait résoudre Q en temps polynomial (disons, $\mathcal{O}(n^a)$ pour un certain a fixé) et que la réduction de P à Q s'effectue en temps $\mathcal{O}(n^b)$ (pour les deux algorithmes, aller et retour), on peut alors résoudre P en temps polynomial $\mathcal{O}(n^{ab})$.
De plus, si $P \leq_p Q$ *et* $Q \leq_p P$, alors P et Q sont *polynomialement équivalents en terme de complexité* : si l'on sait résoudre l'un en temps polynomial, alors on sait également résoudre l'autre en temps polynomial.
Ce principe de réduction polynomiale est à la base de la notion de classe de complexité.

3.2 Réduction entre Clique et Ensemble indépendant

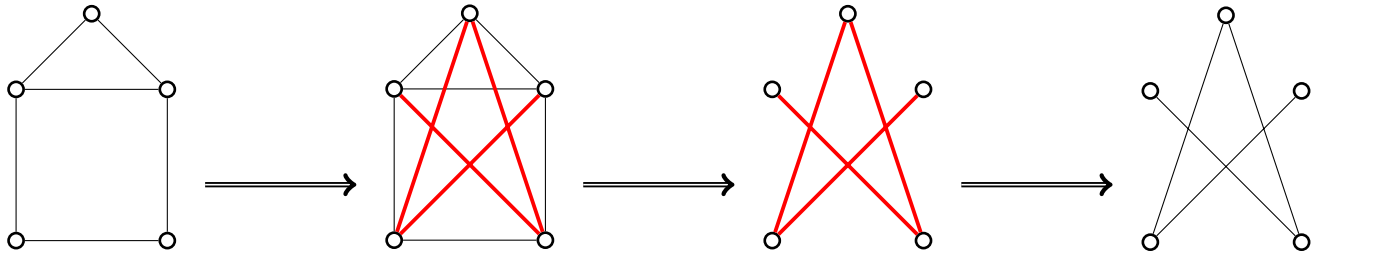
On perçoit intuitivement que les notions de clique et d'ensemble indépendant sont intimement liées. Nous allons montrer que CLIQUE $\leq_p$ ENSEMBLE INDÉPENDANT. Pour cela, nous avons besoin de la notion suivante :

Pour tout graphe G avec un ensemble de sommets V et un ensemble d'arêtes E, on définit le **graphe complémentaire** $\overline{G}$ de la façon suivante :

- $\overline{G}$ a comme ensemble de sommets $\overline{V} := V$;

- $\overline{G}$ a comme ensemble d'arêtes $\overline{E} := \{\{u, v\} \ : \ u, v \in V \text{ et } \{u, v\} \notin E\}$.

En d'autres termes, le graphe complémentaire consiste à garder les mêmes sommets mais à inverser la relation d'arête : là où il y avait une arête, il n'y en a plus ; et là où il n'y en avait pas, il y en a une. Voici ci-dessous un exemple de construction du complémentaire :



Ce graphe s'appelle la maison.

On ajoute les arêtes là où il n'y en avait pas (en rouge).

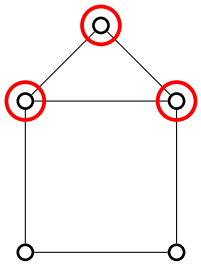
On enlève les arêtes qui existaient.

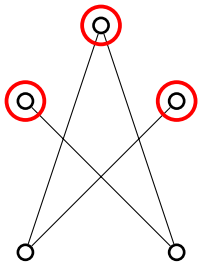
On obtient le complémentaire de la maison.

Nous allons à présent voir comment obtenir une réduction polynomiale de CLIQUE à ENSEMBLE INDÉPENDANT, en utilisant le complémentaire d'un graphe.
Soit G une entrée du problème CLIQUE. On transforme G en une entrée H du problème ENSEMBLE INDÉPENDANT. Pour cela, on pose $H := \overline{G}$, le complémentaire de G. Construire H se fait bien en temps polynomial par rapport à la taille de G : il suffit de parcourir toutes les paires de sommets et vérifier s'ils sont reliés par une arête ou pas ; si ce n'est pas le cas, on ajoute l'arête dans H. L'entrée G contient n sommets et donc au plus $\frac{n(n-1)}{2}$ arêtes, donc est de taille $\mathcal{O}(n^2)$, et vérifier chaque paire de sommets et ajouter l'arête si la condition est remplie prend un temps au plus $2 \times \frac{n(n-1)}{2} = \Theta(n^2)$, on a donc bien un algorithme polynomial.
Montrons maintenant que toute clique de G correspond à un ensemble indépendant de $H = \overline{G}$. Soit S une clique de G. Par définition, tous les sommets de S sont reliés deux à deux par une arête dans G. Donc, dans $\overline{G}$, pour toute paire de sommets de S, ceux-ci ne sont pas reliés par une arête. Donc S est un ensemble indépendant de $\overline{G}$. (Le même raisonnement fonctionne si S est un ensemble indépendant de G : S est aussi une clique de $\overline{G}$.)
Ceci nous donne donc une réduction polynomiale de CLIQUE à ENSEMBLE INDÉPENDANT : en effet, trouver un ensemble de taille maximum dans $H = \overline{G}$ nous donne directement une clique de taille maximum de G. Donc, si on sait résoudre ENSEMBLE INDÉPENDANT, on sait aussi résoudre CLIQUE, et CLIQUE $\leq_p$ ENSEMBLE INDÉPENDANT.

On peut montrer de la même façon que ENSEMBLE INDÉPENDANT $\leq_p$ CLIQUE. Les deux problèmes sont donc polynomialement équivalents.

On peut constater la simplicité de la réduction sur la figure ci-dessous : à gauche, une clique de taille 3, et à droite les mêmes sommets forment un ensemble indépendant de taille 3 dans le complémentaire.



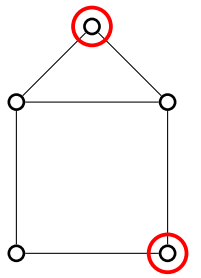


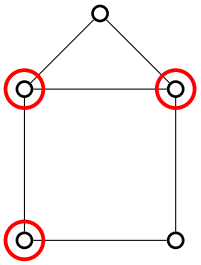
3.3 Réduction entre Ensemble indépendant et Couverture par sommets

La réduction entre ENSEMBLE INDÉPENDANT et COUVERTURE PAR SOMMETS est à la fois plus simple (car elle ne nécessite pas de transformer le graphe) et plus originale (car elle fait intervenir une propriété de graphes intéressante). Soit G un graphe, une entrée pour le problème ENSEMBLE INDÉPENDANT. On construit l'entrée H du problème COUVERTURE PAR SOMMETS de la façon suivante : $H := G$. Il s'agit trivialement d'une réduction polynomiale.
Nous pouvons montrer que l'on peut obtenir une couverture par sommets de H à partir d'un ensemble indépendant de G et inversement. Notons par V l'ensemble des sommets de G. Soit S un ensemble indépendant de G. Prenons une arête de G. Cette arête ne peut pas avoir ses deux extrémités dans S. Elle a donc au moins une de ses deux extrémités dans $V \setminus S$. Donc $V \setminus S$ est une couverture par sommets. Réciproquement, soit S une couverture par sommets de G. Prenons deux sommets quelconques v et w de $V \setminus S$. Si vw est une arête de G, alors l'un des deux est forcé d'être dans S par définition de la couverture par sommets, c'est donc impossible. Donc, $V \setminus S$ est un ensemble indépendant de G. Notons par ailleurs que, grâce à cette équivalence, si S est un ensemble indépendant maximum, $V \setminus S$ sera une couverture par sommets minimum.
Ceci nous donne donc une réduction polynomiale de ENSEMBLE INDÉPENDANT à COUVERTURE PAR SOMMETS : en effet, trouver un ensemble de taille minimum dans $H = G$ nous donne un ensemble indépendant maximum dans G. Donc, si on sait résoudre COUVERTURE PAR SOMMETS, on sait aussi résoudre ENSEMBLE INDÉPENDANT, et ENSEMBLE INDÉPENDANT $\leq_p$ COUVERTURE PAR SOMMETS.

On peut montrer de la même façon que COUVERTURE PAR SOMMETS $\leq_p$ ENSEMBLE INDÉPENDANT. Les deux problèmes sont donc polynomialement équivalents.

On peut constater la simplicité de la réduction sur la figure ci-dessous : à gauche, un ensemble indépendant, et à droite les autres sommets couvrent bien toutes les arêtes du graphe.





4 Conclusion

Nous avons vu que les graphes sont un modèle mathématique très générique permettant de modéliser une multitude de problèmes de la vie réelle. Ils sont utilisés naturellement dans la plupart des domaines scientifiques dès que l'on cherche à modéliser des relations entre des éléments. C'est particulièrement le cas en informatique, puisqu'ils peuvent modéliser par exemple des réseaux de communication, des bases de données, etc.
De plus, certains problèmes (de graphes, mais pas seulement) sont algorithmiquement équivalents et ainsi, si on sait en résoudre un de façon efficace, on sait aussi résoudre l'autre de façon efficace. Ce principe dit de réduction entre problèmes est une technique fondamentale en informatique, dans le domaine de la complexité algorithmique. Il s'agit d'ailleurs d'un domaine de recherche très actif.

Références

[Ber58] C. Berge. *Théorie des graphes et ses applications*. Collection universitaire de mathématiques. Dunod, 1958.

[Cob65] Alan Cobham. The intrinsic computational difficulty of functions. In Yehoshua Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science, Proceedings of the 1964 International Congress*, Studies in Logic and the Foundations of Mathematics, pages 24–30, Amsterdam, 1965. North-Holland Publishing Company. Presented at the 1964 International Congress.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17 :449–467, 1965.

[Eul41] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii Academiae Scientiarum Petropolitanae*, 8 :128–140, 1741. Presented to the St. Petersburg Academy in 1735.

[Har69] Frank Harary. *Graph Theory*. Addison-Wesley, Reading, MA, 1969.

[K$\H{o}$36] Dénes Kőnig. *Theorie der endlichen und unendlichen Graphen*. Akademische Verlagsgesellschaft, Leipzig, 1936.

[Knu76] Donald E. Knuth. Big omicron and big omega and big theta. *ACM SIGACT News*, 8(2) :18–24, 1976.

[Syl78] James Joseph Sylvester. On an application of the new atomic theory to the graphical representation of the invariants and covariants of binary quantics, with three appendices. *American Journal of Mathematics*, 1(1) :64–104, 1878.

Conception et rédaction par Antoine Dailly et Florent Foucaud, avec l'aide et la relecture du groupe ISO.