MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ MOULOUD MAMMERI, TIZI-OUZOU

FACULTÉ DES SCIENCES

DÉPARTEMENT DE MATHÉMATIQUES

MÉMOIRE DE FIN D'ÉTUDE

Filière : Mathématiques Appliquées

Spécialité : Recherche Opérationnelle

Présentée par :

LOUNES Sid Ali

Sujet :

Calcul d'invariants dans les graphes distances héréditaires

Devant le jury d'examen composé de :

Nom	Grade	Université	qualité
Dj. TALEM	MCB	UMMTO	Encadreur
L. OUBAKOUK	MAA	UMMTO	Examinatrice
F.AKLOUCHE	MAA	UMMTO	Présidente

Année universitaire 2023-2024

Remerciements

Tout d'abord, je remercie **Dieu** d'avoir fait cette recherche, car tout vient de **Allah**.

Je remercie sincèrement ma famille : ma mère, mon père, mes frères, mes sœurs, ainsi que tous mes amis et proches, pour leur soutien indéfectible tout au long de ce cheminement.

Je tiens à exprimer ma profonde gratitude à mon encadrant, le **Dr TALEM Djamel**, pour son soutien continu, sa présence et ses précieux conseils tout au long de ce mémoire.

Je remercie également les membres de jury pour avoir accepté de prendre part au jury de soutenance.

TABLE DES FIGURES

TABLE DES MATIÈRES

INTRODUCTION

La recherche opérationnelle (RO) est une discipline clé qui vise à optimiser la prise de décision dans des systèmes complexes. Ce mémoire se concentre sur l'étude des graphes, un outil fondamental en RO, qui permet de modéliser divers problèmes pratiques, tels que la logistique et la gestion des réseaux.

Nous commencerons par une introduction aux notions fondamentales sur les graphes. Ce premier chapitre établira les bases nécessaires pour comprendre les parcours dans les graphes, tels que les algorithmes BFS (parcours en largeur) et DFS (parcours en profondeur), ainsi que d'autres méthodes avancées comme LexBFS. Et à la fin nous avons basé sur la classe des graphes distance héréditaire (GDH).

Le deuxième chapitre portera sur la théorie de la complexité, un aspect crucial pour évaluer l'efficacité des algorithmes. Nous examinerons les différences entre les problèmes de décision et d'optimisation, ainsi que les classes de complexité, notamment P et NP.

Le troisième chapitre se concentrera sur les problèmes de stabilité, de coloration, de dominance et de transversal. Ces problèmes sont non seulement théoriques mais ont également des applications pratiques dans divers domaines, tels que la planification et la gestion des ressources.

Enfin, nous aborderons les graphes bipartis de distance héréditaire (GBDH) dans le quatrième chapitre. Nous y discuterons de la reconnaissance de ces graphes, de la génération de bicliques maximales et de l'optimisation des ensembles dominants, tout en mettant en lumière les défis algorithmiques associés.

Ce mémoire vise à fournir une compréhension approfondie des graphes, et plus particulièrement (GBDH), en tant qu'outil de recherche opérationnelle, en soulignant leur pertinence dans la résolution de problèmes complexes et en proposant des solutions algorithmiques optimisées.

CHAPITRE 1

PRÉLIMINAIRES

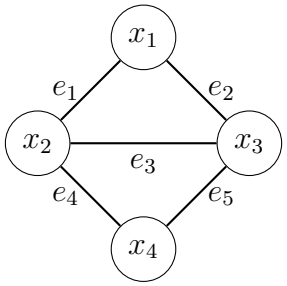
Au cours de ce premier chapitre, nous revisiterons toutes les définitions et notions fondamentales liées à la théorie des graphes, y compris les parcours dans les graphes, qui seront utiles pour les chapitres suivants. En outre, nous rappellerons les définitions de quelques classes de graphes, avec chaque terme nouvellement introduit accompagné de son équivalent en anglais, mis en évidence entre parenthèses. Pour plus de détails sur le chapitre, voir [19, 18].

1.1 Notions fondamentales sur les graphes

1.1.1 Définitions et notations

Un **graphe non orienté** G (graph) est défini par la paire (X, E), où X est un ensemble non vide d'éléments appelés **sommets** (vertices), et E est un ensemble (qui peut être vide) d'éléments appelés **arêtes** (edges). Chaque arête e est associée à deux sommets distincts ou non x et y appelés les extrémités de e. On peut également noter les ensembles X et E comme $X(G)$ et $E(G)$. Le nombre de sommets du graphe noté n ou $|X|$, est le **cardinal** (order) de X, tandis que le nombre d'arêtes du graphe noté m ou $|E|$ est le cardinal de E. Les extrémités x et y d'une arête e sont simplement notées xy ou yx, plutôt que la notation habituelle (x, y) lorsque $x \neq y$.

Lorsque x et y sont **voisins**(neighbors), on dit que l'arête e relie les sommets x et y, et que e est **incidente** (incident) à x et y. Si $x = y$, e est une

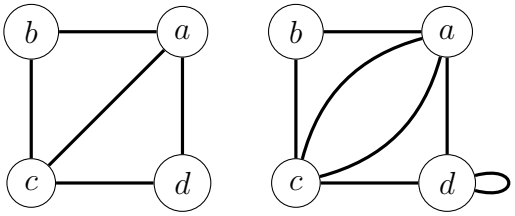


G

FIGURE 1.1 – Graphe non orienté

boucle(loop). Plusieurs arêtes peuvent avoir les mêmes extrémités. dans ce cas, elles sont parallèles ou constituent des **arêtes multiples** (multiple edges) indiquent que G est **un multigraphe** (multigraph).

Un graphe est dit **simple**(simple graph) s'il ne comporte ni boucles ni arêtes multiples. Pour un graphe non simple G, on peut associer le **graphe simple sous-jacent** (underlying simple graph) qui partage le même ensemble de sommets que G. Deux sommets sont reliés dans ce graphe si et seulement s'ils sont distincts et reliés par au moins une arête dans G.



A B

FIGURE 1.2 – Graphe simple A et graphe non simple B

Pour un sommet u et un ensemble non vide W, nous écrivons $u \sim_G W$ pour indiquer que le sommet u est adjacent à tous les sommets de W. L'ensemble des sommets adjacents à v, à l'exception de lui-même, est appelé **voisinage ouvert** (open neighborhood) de v, il est noté $N(v) = \{u \in V \mid vu \in E\}$. Le **voisinage fermé** (closed neighborhood) d'un sommet v, noté $N[v]$, est l'ensemble $N(v) \cup \{v\}$.

Le **degré** (degree) d d'un sommet x dans un graphe G est le nombre d'arêtes de G qui sont incidentes à x. Chaque boucle est comptée deux fois dans le calcul du degré. On note $d(x)$ cet entier.

Un sommet de degré nul est dit **isolé** (isolated vertex), et un sommet de degré un est un **sommet pendant** (pendant vertex). Le plus petit degré d'un graphe G est $\delta(G) = \min\{d(v), v \in V\}$, et son plus grand degré est $\Delta(G) = \max\{d(v), v \in V\}$.

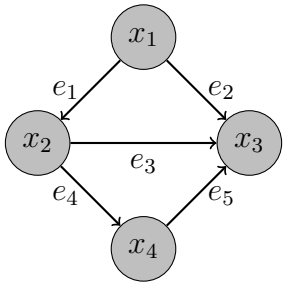
Deux sommets u et v sont des **vrais jumeaux** (true twins) s'ils ont le même voisinage fermé $N[u] = N[v]$. En revanche, ils sont des **faux jumeaux** (false twins) s'ils partagent le même voisinage mais ne sont pas reliés par une arête, soit $N(u) = N(v)$.

Les **graphes complets** (complete graphs) sont les graphes simples tels que deux sommets distincts quelconques sont reliés par une arête.
Un graphe complet est caractérisé de manière unique par son nombre n de sommets, et il est généralement noté K_n. Notons que le nombre d'arêtes m de K_n est égal à $\frac{n(n-1)}{2}$, car dans un graphe complet, chaque paire de sommets est reliée par une arête, à l'exception des boucles. Ainsi, il y a $\frac{n(n-1)}{2}$ arêtes distinctes dans un graphe complet avec n sommets.

Une **clique** (clique) est un sous-ensemble de sommets dans un graphe où chaque paire de sommets est reliée par une arête.

Un sommet x est dit **simplicial** (simplicial vertex) si son voisinage $N(x)$ forme une clique. Cependant, un sommet est simplicial s'il appartient à une unique clique maximale. Un **stable** (stable set ou independent set) dans G est un ensemble de sommets deux à deux non adjacents.

Dans le contexte de la théorie des graphes, un **graphe orienté ou digraphe** G (directed graph or digraph) est défini par un ensemble non vide X de sommets, et un ensemble A d'**arcs**(arcs), ou arêtes orientées.



G

FIGURE 1.3 – Graphes orientés

La **distance** (distance) entre deux sommets u et v dans un graphe est notée $d(u, v)$. Elle représente le nombre minimum d'arêtes qu'il faut parcourir pour aller de u à v. Cette distance est aussi appelée la longueur du plus court

chemin entre ces deux sommets. Si aucun **chemin** (path) n'existe entre u et v, alors $d(u, v) = \infty$.

Une **chaîne**(chain) est une séquence de sommets et d'arêtes $(x_0, e_1, x_1, \ldots, e_k, x_k)$ où chaque arête relie les sommets adjacents dans la séquence. Une chaîne est **simple** (simple) si elle n'utilise pas deux fois la même arête et **élémentaire** (elementary) si elle n'utilise pas deux fois le même sommet. Une chaîne **fermée** (closed) a ses extrémités identiques.

Exemple 1.1.1. *Soit G(a,e_1,b,e_2,c,e_3,d,e_4,e) et H(a,e_1,b,e_2,c,e_2,b,e_3,d,e_4,e)*

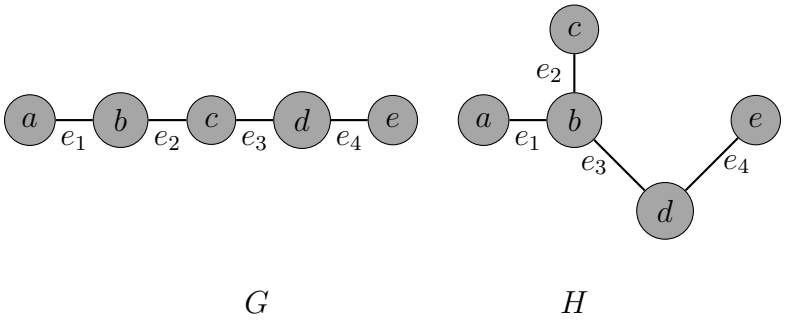


FIGURE 1.4 – (G) Chaîne simple et elémentaire (H) Chaîne non simple et non elémentaire

Un **cycle simple** (simple cycle) est une chaîne simple et fermée. C'est donc une chaîne de la forme : $(x_0, e_1, x_1, ..., e_k, x_0)$.

Un **cycle élémentaire** (elementary cycle) est un cycle dans lequel tous les sommets sont distincts, sauf le premier et le dernier, qui sont identiques. Un cycle est dit **pair ou impair** (even or odd) selon que sa longueur est paire ou impaire. Une **corde** (chord) est une arête qui relie deux sommets d'un cycle sans faire partie de ce cycle. Un **trou** (hole) est un cycle d'au moins quatre sommets et sans corde, noté C_n avec $n > 3$. Un **antitrou** (antihole) est le complémentaire d'un trou.

Un **sous-graphe**(subgraph) d'un graphe $G = (X, E)$ est une structure $H = (Y, F)$ où $Y \subseteq X$ et $F \subseteq E$, où chaque arête de F relie des sommets de Y. Un sous-graphe peut être **partiel** (partial graph), en conservant les mêmes sommets mais en retirant certaines arêtes, ou **induit** (induced graph) par un ensemble de sommets $A \subset X$, en incluant toutes les arêtes reliant des sommets de A dans G.

Un graphe est **connexe** (connected graph) s'il existe un chemin reliant chaque paire de sommets. Les **composantes connexes** (connected component) sont les **sous-graphes connexes** (connected subgraphs) les plus

grands, et un graphe est connexe s'il possède une seule composante connexe. Les composantes connexes sont mutuellement disjointes et définissent la décomposition en composantes connexes unique du graphe.

Un sommet (resp. un ensemble) **d'articulation** (cutvertex (resp. cutset)) est un sommet (resp. ensemble de sommets) tel que sa suppression augmente le nombre de composantes connexes. Parfois, on utilise également le terme **séparateur** (separator) pour désigner un ensemble d'articulation.

1.1.2 Quelque type de graphe

Deux graphes $G = (V_G, E_G)$ et $H = (V_H, E_H)$ sont dits **isomorphes**(isomorphic graph) s'il existe une bijection $f : V_G \to V_H$ telle que pour tout couple de sommets $u, v \in V_G$, u et v sont adjacents dans G si et seulement si $f(u)$ et $f(v)$ sont adjacents dans H. On note alors $G \cong H$.

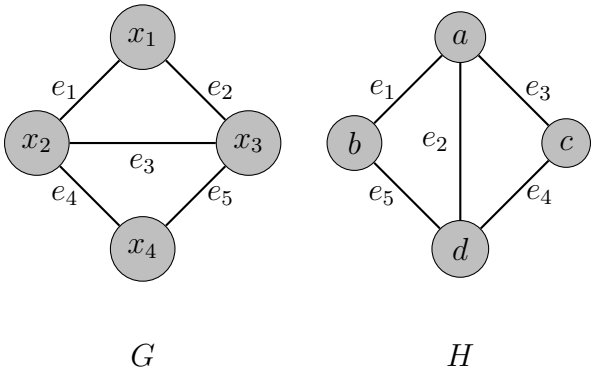


FIGURE 1.5 – G et H sont deux graphe isomorphes

Pour un graphe $G = (V, E)$, le **graphe complémentaire** (complementary graph) $G^c = (V, E^c)$ a le même ensemble de sommets V, mais E^c est l'ensemble des arêtes qui manquent à G pour qu'il soit complet.

graphes valués ou graphe pondéré (weighted graph), est un type de graphe dans lequel des valeurs numériques (appelées poids ou valeurs) sont assignées aux arêtes. Ces valeurs peuvent représenter différentes choses selon le contexte, comme des distances, des coûts, des capacités, des durées, etc.

Un graphe G est dit **régulier** (regular graph) lorsque les degrés de ses sommets sont tous égaux. On peut préciser le degré commun k des sommets en disant k-régulier.

Un **arbre** (tree) est un graphe connexe et **acyclique** (acyclic), ce qui signifie qu'il ne contient aucun cycle. Il s'agit d'un graphe simple, car la présence d'une boucle ou d'une arête double créerait un cycle. Une chaîne élémentaire est un exemple d'arbre.

Un graphe $G = (V, E)$ est **biparti** (bipartite graph) s'il existe une partition de l'ensemble des sommets V en deux sous-ensembles disjoints X et Y, tels que $V = X \cup Y$ et pour chaque arête $(u, v) \in E$, $u \in X$ et $v \in Y$ ou inversement. Une **biclique** (biclique) dans un graphe biparti $G = (X \cup Y, E)$ est **un sous-graphe biparti complet** (a complete bipartite subgraph) où tous les sommets de X sont connectés à tous les sommets de Y. Une biclique est souvent notée $K_{m,n}$, où m est le nombre de sommets dans X et n est le nombre de sommets dans Y.

Les **graphes planaires** (planar graphs) sont un type particulier de graphes où une représentation peut être réalisée sans que les arêtes ne se croisent, sauf à leurs extrémités communes. Ils sont importants en théorie des graphes, notamment grâce au théorème des quatre couleurs, affirmant que tout graphe planaire peut être colorié avec quatre couleurs sans que deux sommets reliés par une arête ne soient de la même couleur [1].

Exemple 1.1.2. *Exemple graphe planaire avec trois face F1.F2.F3*

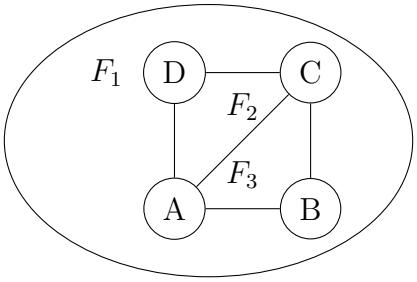


FIGURE 1.6 – Graphe planaire

1.1.3 Représentation

Représentation par matrice d'adjacences (adjacency matrix)

La matrice d'adjacence d'un graphe simple $G = (V, E)$ d'ordre n est une matrice M de dimensions $n \times n$, où n est le nombre de sommets dans V, et E est l'ensemble des arêtes du graphe.

Pour chaque paire de sommets i et j, la case $m_{i,j}$ de la matrice d'adjacence est définie comme suit :

$$m_{i,j} = \begin{cases} 1 & si \quad ij \in E \\ 0 & sinon \end{cases} \tag{1.1}$$

Exemple 1.1.3. *La matrice d'adjacence pour la Figure 1.1 est la suivante :*

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

Représentation par la matrice d'incidence (incidence matrix)

1. Matrice d'incidence pour un graphe orienté :

- Si $G = (V, E)$ est un graphe orienté avec n sommets et m arêtes, alors la matrice d'incidence B est une matrice $n \times m$.
- Pour un graphe orienté, b_{ij} est défini comme suit :

$$b_{ij} = \begin{cases} 1 \quad si \quad x_i \text{ est l'extrémité initiale de } \mathrm{e}_j; \\ -1 \quad si \quad x_i \text{ est l'extrémité terminale de } \mathrm{e}_j; \\ 0 \quad si \quad x_i \text{ n'est pas une extrémité de } \mathrm{e}_j. \end{cases} \tag{1.2}$$

Exemple 1.1.4. *La matrice d'incidence pour la Figure 1.3 est la suivante :*

$$\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 \\ 0 & -1 & -1 & 0 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \end{array}$$

2. Matrice d'incidence pour un graphe non orienté :

- Si $G = (V, E)$ est un graphe non orienté avec n sommets et m arêtes, alors la matrice d'incidence B est une matrice $n \times m$.

- Pour un graphe non orienté, b_{ij} est défini comme suit :

$$b_{ij} = \begin{cases} 1 \quad si \quad x_i \text{ est une extrémité de } \mathrm{e}_j; \\ 0 \quad sinon. \end{cases} \tag{1.3}$$

Exemple 1.1.5. *La matrice d'incidence associée au graphe non orienté G de la figure 1.1 est la suivante :*

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Dans les deux cas, les colonnes de la matrice représentent les arêtes du graphe et les lignes représentent les sommets. Cette représentation matricielle est utile pour analyser les relations entre les sommets et les arêtes d'un graphe donné.

1.2 Prcours dans les graphe

Pour identifier les sommets accessibles dans un graphe, nous employons des méthodes de parcours systématique. Principalement, deux types de parcours sont utilisés : le parcours en largeur (BFS) et le parcours en profondeur (DFS). Soit $G = (X, E)$ un graphe non orienté parceque permet de se concentrer sur la recherche de connexité et l'accessibilité mutuelle des sommets sans se soucier des directions des arêtes, ce qui est souvent l'objectif principal lors de l'exploration de graphes pour identifier les sommets accessibles. L'objectif de ces deux parcours est d'explorer le graphe G à partir d'un sommet de départ s, souvent appelé racine ou source. En parcourant le graphe à partir de s, nous visitons un par un tous les sommets accessibles via des chemins.

1.2.1 Parcours en largeur BFS

Le parcours en largeur d'un graphe ne nécessite pas nécessairement de suivre la stratégie "premier visité, premier exploré"(FIFO) au niveau des sommets individuels. Au lieu de cela, on peut appliquer cette stratégie au niveau des **"vagues"** (wave) de sommets visités, où une vague est l'ensemble des successeurs d'un sommet exploré. En gérant l'ordre de reprise des successeurs d'un sommet donné de cette manière, on ne modifie pas la stratégie de parcours en largeur d'abord en elle-même. Par exemple, l'application de cette méthode au calcul des distances dans un graphe reste fonctionnelle.

Le parcours lexicographique exploite cette liberté en reprenant les sommets de chaque vague dans un ordre choisi, afin d'obtenir certaines propriétés sur la suite des sommets explorés.

1.2.2 Parcours en profondeur DFS

L'algorithme de parcours en profondeur explore un graphe à partir d'un sommet source s. Il descend plus profondément dans le graphe chaque fois que cela est possible. Initialement, le sommet de départ s est marqué comme exploré, puis un voisin quelconque est choisi. Si ce voisin n'a pas déjà été visité, il est marqué comme exploré. Si ce voisin n'est pas fermé, c'est-à-dire s'il a au moins un voisin non encore visité, on explore ce voisin et on répète le processus jusqu'à ce que tous les sommets soient explorés.

Cet algorithme utilise une **pile** (stack) comme structure de données, suivant le principe "dernier marqué, premier sorti" (LIFO). Les étapes du parcours en profondeur peuvent être résumées comme suit :
1. Marquer le sommet de départ s comme exploré et choisir un voisin quelconque.
2. Si le voisin choisi n'a pas encore été visité, le marquer comme exploré.
3. Si ce voisin n'est pas fermé (c'est-à-dire s'il a au moins un voisin non encore visité), l'explorer et répéter le processus pour chaque voisin non encore visité jusqu'à ce que tous les sommets soient explorés.

Pour plus de d´etails sur ces deux parcours de graphes, nous proposons de consulter la référence [9]

1.2.3 LexBFS

LexBFS a été introduit en 1976 par ROSE et al [32] pour la reconnaissance d'une classe de graphes particulière, les graphes triangulés. Le parcours en largeur lexicographique, également connu sous le nom de LexBFS ou LBFS, est un algorithme utilisé pour générer un ordre total des sommets d'un graphe non orienté, à partir d'un sommet origine donné. Contrairement au parcours en largeur classique (BFS), l'ordre résultant peut être utilisé pour le parcours BFS, mais pas nécessairement dans l'autre sens.

Dans le parcours LexBFS, on énumère les sommets de G de n à 1 à partir d'un sommet d'origine s de la manière suivante : on associe à chaque sommet v une étiquette $L(v)$ qui est un vecteur. Au début, $L(s) = n$, et pour $v \neq s$, $L(v) = \emptyset$. Pour i allant de n à 1, on attribue l'entier i au sommet v ayant l'étiquette la plus grande lexicographiquement, et pour tout sommet

w, voisin de v, qui n'est pas encore numéroté, on concatène l'entier i à la fin de l'étiquette $L(w)$.

Définition 1.2.1. *Le niveau de distance N_i d'un graphe G par rapport à un sommet a est l'ensemble des sommets x de G tels que la distance $d(x, a) = i$. Ces niveaux de distances sont calculés par un parcours en largeur à partir de a. Une fois les niveaux de distances calculés, on note p le plus petit entier tel que $N_{p+1} = \emptyset$, ce qui signifie que la composante connexe de G contenant a est partitionnée en $[a, N_1, N_2, \ldots, N_p]$. Pour chaque sommet $x \in N_i$, on note $N^-(x)$ (respectivement $N^=$ et N^+) le voisinage de x dans N_{i-1} (respectivement N_i et N_{i+1}), et $d^-(x)$ (respectivement $d^=$ et d^+) le cardinal de $N^-(x)$ (respectivement $N^=$ et N^+).*

Remarque 1.2.1. *Puisque l'ordre σ est un ordre total sur les sommets de G, donc, si v_j est un voisin de v_i, alors $v_j \in N^+(v_i)$ ou $v_j \in N^-(v_i)$. Ainsi, pour tout $i < n$, $N(v_i) = N^+(v_i) \cup N^-(v_i)$*

Algorithm 1 [4] Lex-BFS

Entrée : Un graphe $G = (V, E)$ et un sommet de source s.
Sortie : Un ordre σ de V.
pour chaque sommet $x \in V$ **faire**
    $L(x) \leftarrow \emptyset$
fin pour
$L(s) \leftarrow \{n\}$
pour $i \leftarrow n$ à 1 **faire**
    choisir un sommet non numéroté $v \in V$ d'étiquette maximum
    $\sigma(i) \leftarrow v$
    pour chaque voisin non numéroté w de v **faire**
        $L(w) \leftarrow L(\mathrm{w}) \cup \{i\}$ ▷ Concaténation de i à la fin de l'étiquette de w.
    fin pour
fin pour

Dans la figure 1.7, à gauche, nous avons un graphe biparti $G = (X, Y, E)$; à droite, nous avons le même graphe biparti G ordonné par LexBFS σ qui est défini comme suit : $\sigma(1) = e = v_1, \sigma(2) = f = v_2, \sigma(3) = t = v_3, \sigma(4) = u = v_4, \sigma(5) = v = v_5, \sigma(6) = a = v_6, \sigma(7) = b = v_7, \sigma(8) = c = v_8, \sigma(9) = d = v_9, \sigma(10) = s = v_{10}$

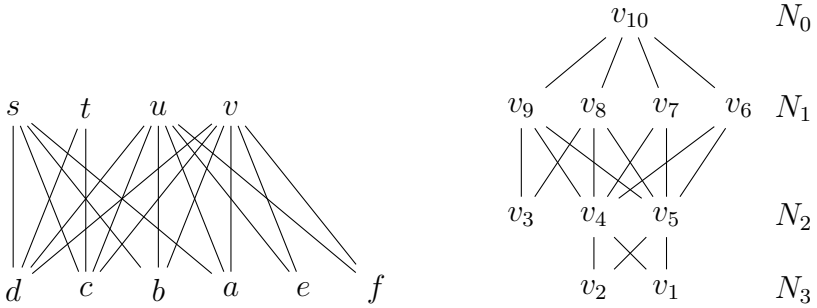


FIGURE 1.7 – Graphe biparti ordonné par l'ordre LexBFS

En tenant compte de la définition et du contenu de la référence [32], nous déduisons les observations suivantes :

Observation 1. *Si $\sigma = (v_1, v_2, \ldots, v_n)$ est un LexBFS, alors pour tout $i = 1, \ldots, n-1$,*

$$j \in L(v_i) \Leftrightarrow v_j \in N^+(v_i)$$

Observation 2. *L'ordre σ est un LexBFS si et seulement si pour toute paire de sommets v_i, v_j avec $v_i <_\sigma v_j$ l'une des conditions suivantes est satisfaite*

- $N^+(v_i) = N^+(v_j)$;
- $N^+(v_i) \subset N^+(v_j)$;
- $max\Big(N^+(v_i) - N^+(v_j)\Big) <_\sigma max\Big(N^+(v_j) - N^+(v_i)\Big)$

Dans la figure 1.7, nous avons :
$v_1 <_\sigma v_2$ car $N^+(v_1) = \{v_4, v_5\} = N^+(v_2)) = \{v_4, v_5\}$;
$v_3 <_\sigma v_4$ car $N^+(v_3) = \{v_8, v_9\} \subset N^+(v_4)) = \{v_6, v_7, v_8, v_9\}$;
$v_2 <_\sigma v_3$ car $\max\left(N^+(v_2) - N^+(v_3)\right) = \max\{v_4, v_5\} = v_5 <_\sigma v_9 = \max\left(N^+(v_3) - N^+(v_2)\right)$.

Observation 3. *Dans un graphe ordonné par lexBFS :*

1. *L'ensemble des sommets est partitionné en niveaux $N_0, N_1, \ldots, N_k$ tels que $N_0 = \{v_n\}$, et*

$$\forall i = 1, \ldots, k, N_i = \{v, d(v, v_n) = i\}$$

2. *Une arête relie deux sommets qui sont, soit dans un même niveau soit dans deux niveaux consécutifs ;*
3. *Si N_i, N_{i+1} sont deux niveaux consécutifs, les sommets de N_i sont énumérés avant l'énumération des sommets de N_{i+1}.*

Les sommets du graphe de la figure 1.7 est partitionné en 4 niveau $N_0, N_1, N_2, N3$. Remarquons que les sommets d'un même niveau ne sont pas adjacents, car G est biparti et donc il est sans cycle impair.

Observation 4. *[33] Soient u et v deux sommets dans un graphe G ordonné par LexBFS σ. Alors*

$$u <_\sigma v \Rightarrow \forall u' \in N^+(u), \exists v' \in N^+(v) : u' \leq_\sigma v'$$

Exemple 1.2.1. *On montre comment dérouler le parcours lexicographique LexBFS sur un graphe.*

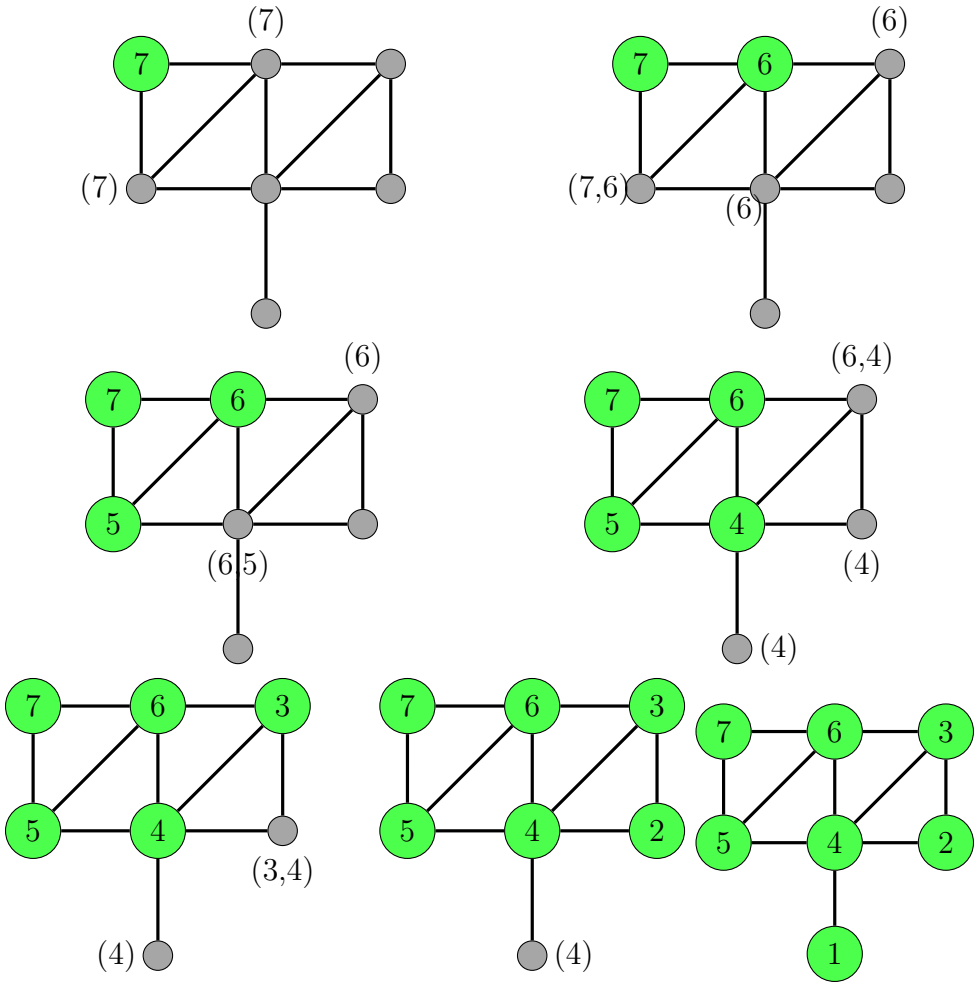


FIGURE 1.8 – Illustration d'un parcours LexBFS

1.3 Classes de graphes

Les **classes de graphes** (graph classes) sont des catégories de graphes ayant des propriétés spécifiques qui simplifient l'analyse et la résolution de problèmes. En étudiant ces classes, on peut développer des algorithmes plus efficaces pour des tâches variées, telles que l'optimisation, la recherche de chemins, et la modélisation de réseaux. Leur utilité réside dans la capacité à exploiter leurs caractéristiques structurelles pour améliorer la performance des solutions dans des applications pratiques et théoriques.

1.3.1 Graphes Triangulé

Un **graphe triangulé** (triangulated graph) ou cordal est un graphe qui ne contient pas de cycle de longueur supérieure à trois sans corde. En d'autres termes, dans un graphe cordal, chaque cycle de longueur quatre ou plus il contient une corde reliant deux sommets non consécutifs du cycle, formant ainsi des triangles.

Un **ordre d'élimination simpliciale** (elimination ordering) est une séquence de sommets d'un graphe telle que, lorsqu'on élimine les sommets dans cet ordre, chaque sommet éliminé est un sommet simplicial dans le sous-graphe restant. Cela signifie que, pour chaque sommet éliminé, ses voisins dans le sous-graphe induit par ses voisins forment une clique, c'est-à-dire qu'ils sont tous reliés entre eux.

Theorem 1.3.1. *(Fulkerson et Gross (1965)) Un graphe est cordal si et seulement s'il admet un ordre d'élimination simplicial.*

Rose, Tarjan et Lueker ont introduit dans un algorithme appelé parcours en largeur lexicographique (ou Lex-BFS pour Lexicographic Breadth-First Search) qui permet de trouver efficacement une ordre d'élimination simplicial d'un graphe cordal, et qui permet par conséquent de déterminer rapidement si un graphe est cordal ou non [30].

Exemple 1.3.2. *Soit dans la figure 1.9 l'ordre des sommets obtenu par le parcours LexBFS. Cet ordre est un ordre d'élimination simpliciale, donc le graphe est triangulé.*

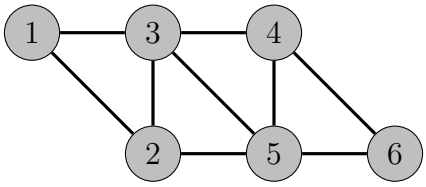


FIGURE 1.9 – Graphe Triangulé

1.3.2 Cographe

Les **cographes** (cograph) forment une sous-classe des graphes à distance héréditaire avec des propriétés et des contraintes spécifiques supplémentaires. Elle a été introduite par plusieurs auteurs indépendamment dans les années 1970. Un cograph est un graphe simple qui peut être construit de manière récursive selon les règles suivantes :

1. Un graphe à un sommet est un cograph.
2. Le complémentaire d'un cograph est un cograph.
3. L'union de deux cographes est un cograph.

Il existe de nombreuses caractérisations des cographes, notamment :

- Les cographes sont les graphes P_4-free, c'est-à-dire ceux qui n'ont pas le chemin sur quatre sommets comme sous-graphe induit.

La plupart des problèmes algorithmiques peuvent être résolus sur cette classe en temps polynomial, et même linaire, du fait de ses propriétés structurelles.

Un coarbre (cotree) est une représentation unique et compacte d'un cographe, où :

1. **Les feuilles** (leaves) du coarbre représentent les sommets du cographe.
2. **Les nœuds** (nodes) internes représentent des opérations :
   - Un nœud étiqueté par 0 signifie l'union des sous-arbres.
   - Un nœud étiqueté par 1 signifie la jointure des sous-arbres.
3. Il y a une arête entre deux sommets du cographe si leur **plus petit ancêtre commun** (lowest common ancestor (LCA)), c'est-à-dire "le nœud le plus profond qui est un ancêtre de ces deux feuilles" dans le coarbre est étiqueté par 1.
4. En échangeant les 1 et les 0, on obtient le coarbre du graphe complémentaire.

Exemple 1.3.3. *Dans la figure 1.10, les sommets a et b ne sont pas connectés car leur plus petit ancêtre commun est étiqueté par 0. De même, les sommets d et e ne sont pas connectés pour la même raison. Le sommet c est connecté à tous les sommets car leur plus petit ancêtre commun est étiqueté par 1.*

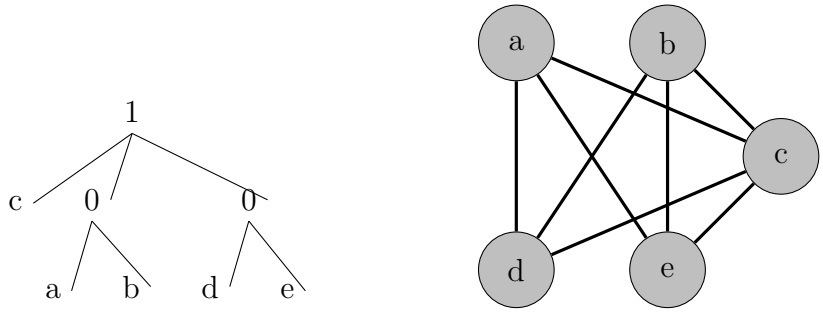


FIGURE 1.10 – Cographe avec le coarbre associé

Conclusion du chapitre 1

Ce chapitre a couvert les notions fondamentales sur les graphes, incluant les définitions, types, et représentations. Nous avons également exploré les méthodes de parcours BFS, DFS, et en particulier LexBFS, ainsi que les classes de graphes telles que les graphes triangulés et les cographes. Ces concepts et techniques posent les bases pour des études plus avancées en théorie des graphes.

CHAPITRE 2

THÉORIE DE LA COMPLEXITÉ

La théorie de la complexité est une branche des mathématiques et de l'informatique, qui s'intéresse à l'étude de la difficulté intrinsèque des problèmes algorithmiques. Elle se concentre sur la quantité de ressources nécessaires pour résoudre un problème, telles que le temps d'exécution (complexité temporelle) et l'espace mémoire (complexité spatiale). Certains problèmes sont considérés comme plus difficiles car ils nécessitent des algorithmes avec un temps de calcul bien supérieur à d'autres. Afin d'approfondir le concept du chapitre, voir [2].

2.1 Problème de décision et problème d'optimisation

Un problème algorithmique est une question générique qui s'applique à un ensemble d'éléments en entrée appelés **ses instances** (problem instances) ou **ses données** (problem data).

2.1.1 Problèmes de décision

Un **problème de décision** (decision problem) consiste à répondre par "oui" ou "non" à une question spécifique. Par exemple :

- **Existence d'un chemin :** Déterminer s'il existe un chemin entre deux sommets donnés dans un graphe.
- **Cycle eulérien :** Vérifier si un graphe possède un cycle eulérien, un cycle qui passe par chaque arête exactement une fois.

— **Coloration de graphe :** Déterminer si un graphe peut être coloré avec un nombre donné de couleurs sans que deux sommets adjacents aient la même couleur.

La version décision du problème de la clique est :

Instance : Un graphe G et un entier positif k.
Question : Existe-il dans G une clique avec au moins k sommets ?

Ces problèmes sont souvent utilisés pour classifier des graphes et évaluer leurs propriétés structurelles.

2.1.2 Problèmes d'optimisation

Les problèmes d'optimisation cherchent à trouver la meilleure solution selon un critère donné. Voici quelques exemples :

— **Problème du voyageur de commerce (TSP) :**Trouver le chemin le plus court qui permet de visiter un ensemble de villes et de revenir à la ville de départ.

— **Maximisation de la durée de vie dans les réseaux de capteurs :** Ce problème vise à optimiser la durée de vie d'un réseau de capteurs en planifiant la communication entre les nœuds pour minimiser la consommation d'énergie.

— **Recherche de sous-graphes connexes équilibrés :** Trouver le plus grand sous-graphe connexe contenant un nombre égal de sommets de différentes couleurs.

La version d'Optimisation du problème de la clique est :

Instance : un graphe G.
Objectif : Déterminer une clique maximum de G.

Les problèmes de décision et d'optimisation sont essentiels pour comprendre la structure et le comportement des graphes. Ils sont utilisés dans divers domaines, allant des réseaux de communication à la biologie, et leur étude continue d'évoluer avec les avancées en algorithmique et en théorie de la complexité.

2.2 Complexité d'un algorithme et complexité d'un problème

La complexité d'un algorithme et la complexité d'un problème sont deux concepts fondamentaux en informatique théorique, chacun ayant ses propres implications et méthodes d'évaluation.

2.2.1 Complexité d'un algorithme

Un **algorithme** (algorithm) est une suite finie et ordonnée d'instructions ou d'étapes définies de manière claire et précise, conçue pour résoudre un problème particulier ou accomplir une tâche spécifique. La donnée est ce que l'algorithme traite, tandis que la taille de l'algorithme fait généralement référence à la taille de l'entrée, qui est un facteur déterminant dans l'analyse de l'efficacité de l'algorithme. La complexité d'un algorithme fait référence à la quantité de ressources nécessaires pour exécuter cet algorithme, généralement mesurée en termes de **complexité temporelle** (temporal complexity) et de **complexité spatiale** (spacial complexity).

Complexité temporelle

Elle mesure le temps d'exécution d'un algorithme en fonction de la taille des données d'entrée. On utilise souvent la notation grand $\mathcal{O}$ pour exprimer cette complexité, qui décrit le comportement asymptotique de l'algorithme dans le pire des cas. Par exemple, un algorithme de tri peut avoir une complexité de $\mathcal{O}(n^2)$, dans le cas du tri par insertion. Tandis qu'un tri rapide peut avoir une complexité moyenne de $\mathcal{O}(n \log n)$.

L'idée principale du tri par insertion est de construire le tableau trié un élément à la fois, en déplaçant progressivement chaque nouvel élément vers sa position correcte dans la partie triée du tableau.

Étapes du Tri par Insertion :

- On considère que le premier élément du tableau est déjà trié.
- Pour chaque élément suivant du tableau (à partir du deuxième) :
  - On compare cet élément avec les éléments de la partie triée du tableau.
  - On déplace tous les éléments plus grands que cet élément vers la droite.
  - On insère l'élément à sa position correcte dans la partie triée.
- Répéter jusqu'à ce que tous les éléments soient triés.

Exemple 2.2.1. *Prenons un tableau d'exemple :* $[5, 2, 9, 1, 5, 6]$.

- *Considérer que le premier élément* 5 *est déjà trié* $[5, 2, 9, 1, 5, 6]$.
- *Examiner le deuxième élément* 2. 2 *est plus petit que* 5*, donc on déplace* 5 *vers la droite et insère* 2 *à la première position* $[2, 5, 9, 1, 5, 6]$.
- *Examiner le troisième élément* 9. 9 *est plus grand que* 5*, donc on le laisse à sa position* $[2, 5, 9, 1, 5, 6]$.
- *Examiner le quatrième élément* 1. 1 *est plus petit que* $9, 5$ *et* 2*, donc on les déplace tous vers la droite et insère* 1 *en première position* $[1, 2, 5, 9, 5, 6]$.
- *Examiner le cinquième élément* 5. 5 *est plus petit que* 9 *mais plus grand que* 2*, on déplace* 9 *à droite et insère* 5 *avant* 9*, le tableau :* $[1, 2, 5, 5, 9, 6]$.
- *Examiner le sixième élément* 6. 6 *est plus petit que* 9 *mais plus grand que* 5*, on déplace* 9 *à droite et insère* 6 *avant* 9*. Le tableau final trié :* $[1, 2, 5, 5, 6, 9]$.

En raison de sa simplicité et de son efficacité sur de petites entrées ou des entrées partiellement triées, il est parfois utilisé dans des algorithmes de tri plus complexes comme le tri rapide (quicksort) pour trier de petites sous-parties du tableau.

Étapes du Tri Rapide :

1. Choisir un Pivot : Sélectionner un élément du tableau comme pivot. Ce choix peut être aléatoire, le premier élément, le dernier, ou tout autre élément du tableau.

2. Partitionner le Tableau : Réorganiser le tableau de telle sorte que tous les éléments inférieurs au pivot se trouvent à gauche du pivot et tous les éléments supérieurs au pivot se trouvent à droite.

3. Appliquer le Tri de Manière Récursive : Appliquer récursivement les étapes 1 et 2 aux sous-tableaux à gauche et à droite du pivot.

4. Fusionner les Résultats : Les sous-tableaux étant triés, le tableau global est également trié.

Exemple 2.2.2. *Prenons un tableau d'exemple :* $[3, 6, 8, 10, 1, 2, 1]$.

- *Choisissons* 6 *comme pivot.*
- *Partitionner le Tableau*
  - *Tous les éléments inférieurs à* 6 *vont à gauche :* $[3, 1, 2, 1]$.
  - *Le pivot* 6 *reste au milieu.*
  - *Tous les éléments supérieurs à* 6 *vont à droite :* $[8, 10]$
  - *Tableau après partitionnement :* $[3, 1, 2, 1, 6, 8, 10]$.
- *Appliquer le tri rapide aux sous-tableaux* $[3, 1, 2, 1]$ *et* $[8, 10]$.
- *Sous-tableau Gauche* $[3, 1, 2, 1]$ *:*
  - *Choisir un pivot, disons* 3.
  - *Partitionner :* $[1, 2, 1, 3]$

  - *Appliquer récursivement à* $[1, 2, 1]$ *(choisir pivot* 1*, partitionner en* $[1, 1, 2]$*).*
  - *Résultat trié du sous-tableau gauche :* $[1, 1, 2, 3]$.
- *Sous-tableau Droit* $[8, 10]$ *: Déjà trié car seulement deux éléments et* $8 < 10$.
- *Fusionner les Résultats :* $[1, 1, 2, 3, 6, 8, 10]$.

Lorsque nous analysons la complexité d'un algorithme, nous cherchons généralement à déterminer son ordre de grandeur plutôt que sa complexité exacte. Pour cela, nous utilisons des notations asymptotiques qui décrivent le comportement des fonctions lorsque la taille de l'entrée devient grande.
Les notations asymptotiques sont les suivantes :

La notation $\Theta(g(n))$ représente la borne asymptotique exacte. Si $f(n) = \Theta(g(n))$, cela signifie que $f(n)$ est asymptotiquement équivalent à $g(n)$ à la fois en tant que majorant et minorant. En d'autres termes, $f(n)$ croît au même rythme que $g(n)$ à mesure que n tend vers l'infini.
- Θ : $f = \Theta(g)$ si et seulement si $f = \mathcal{O}(g)$ et $g = \mathcal{O}(f)$

La notation $\mathcal{O}(g(n))$ représente une borne asymptotique supérieure. Si $f(n) = \mathcal{O}(g(n))$, $f(n)$ ne croît pas plus vite que $g(n)$ asymptotiquement.
- $\mathcal{O}$: $f = \mathcal{O}(g)$ s'il existe n_0 et $c \geq 0$ tels que pour tout $n \geq n_0$, $f(n) \leq c \times g(n)$

La notation $\Omega(g(n))$ représente une borne asymptotique inférieure. Si $f(n) = \Omega(g(n))$, cela signifie qu'il existe des constantes $c > 0$ et $n_0 \geq 0$ telles que pour tout $n \geq n_0$, $f(n) \geq c \cdot g(n)$. Autrement dit, $f(n)$ croît au moins aussi vite que $g(n)$ asymptotiquement.
- Ω : $f = \Omega(g)$ si $g = \mathcal{O}(f)$

Exemple 2.2.3. — ***Notation*** Θ ***:***

- *Soit* $f(n) = 3n^2 + 2n + 1$.
- *Soit* $g(n) = n^2$.

Pour montrer que $f(n) = \Theta(g(n))$*, il faut montrer qu'il existe des constantes positives* c_1, c_2 *et* n_0 *telles que :*

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \quad pour \quad tout \quad n \geq n_0.$$

Pour n *suffisamment grand, les termes de moindre ordre* $(2n + 1)$ *deviennent négligeables par rapport à* n^2*. Ainsi, il existe des constantes* $c_1 = 1$*,* $c_2 = 4$*, et* $n_0 = 3$ *telles que :*

$$n^2 \leq 3n^2 + 2n + 1 \leq 4n^2.$$

Donc, $f(n) = 3n^2 + 2n + 1 = \Theta(n^2)$.

— ***Notation $\mathcal{O}$:***

- *Soit* $f(n) = 3n^2 + 2n + 1$.
- *Soit* $g(n) = n^2$.

Pour montrer que $f(n) = \mathcal{O}(g(n))$, *il faut trouver une constante* c *telle que :*

$$f(n) \leq c \cdot g(n) \quad \forall \quad n \geq n_0.$$

On peut choisir $c = 5$ *et* $n_0 = 3$, *et on a :*

$$3n^2 + 2n + 1 \leq 5 \cdot n^2 \quad \forall \quad n \geq 3.$$

Ainsi, $f(n) = 3n^2 + 2n + 1 = \mathcal{O}(n^2)$.

— ***Notation Ω :***

- *Soit* $f(n) = 3n^2 + 2n + 1$.
- *Soit* $g(n) = n^2$.

Pour montrer que $f(n) = \Omega(g(n))$, *il faut trouver une constante* c *telle que :*

$$f(n) \geq c \cdot g(n) \quad \forall \quad n \geq n_0.$$

Pour n *suffisamment grand, les termes* $2n + 1$ *deviennent négligeables par rapport à* $3n^2$. *En prenant* $c = 1$ *et* $n_0 = 1$, *on a :*

$$3n^2 + 2n + 1 \geq 1 \cdot n^2 \quad \forall \quad n \geq 1.$$

Ainsi, $f(n) = 3n^2 + 2n + 1 = \Omega(n^2)$.

Ces notations nous permettent d'évaluer la croissance relative des fonctions lorsque la taille de l'entrée augmente, ce qui nous donne une indication de leur comportement asymptotique.

Complexité spatiale

Elle mesure la quantité de mémoire supplémentaire requise par l'algorithme en plus des données d'entrée. Par exemple, un algorithme qui nécessite de stocker des données temporaires dans des structures de données supplémentaires aura une complexité spatiale supérieure à celle d'un algorithme qui ne nécessite pas de telles structures.

Exemple 2.2.4. *Par exemple, l'entier* $x = 23$ *est une instance du problème de primalité de taille 5. En effet, le codage en binaire de* 23 *est* 10111, *donc il faut 5 cases mémoire pour la représentation de* 23 *par un ordinateur. Quand une instance est un graphe, sa taille est une fonction des paramètres* n, m *qui sont respectivement le nombre de sommets et d'arêtes de ce dernier.*

Ces deux aspects de la complexité sont souvent indépendants. Un algorithme peut avoir une bonne complexité temporelle mais une mauvaise complexité spatiale, ou vice versa.

2.2.2 Complexité d'un problème

Pour un problème donné, il peut exister plusieurs algorithmes différents chacun avec une complexité différente. La complexité d'un algorithme donné ne change pas la complexité intrinsèque du problème, mais elle détermine l'efficacité avec laquelle ce problème peut être résolu.

Trouver un algorithme optimal pour un problème signifie trouver un algorithme dont la complexité est la plus basse possible. La complexité d'un problème peut être évaluée par ses bornes inférieures, c'est-à-dire la complexité minimale possible que tout algorithme résolvant ce problème doit avoir. Si un algorithme atteint cette borne inférieure, il est considéré comme optimal.

La complexité d'un problème représente une mesure abstraite de sa difficulté, tandis que la complexité d'un algorithme est une mesure concrète de l'efficacité d'une méthode spécifique pour résoudre ce problème. Les deux concepts sont étroitement liés car la compréhension de la complexité d'un problème guide souvent la conception et le choix des algorithmes pour le résoudre.

2.3 Classes des problèmes

Nous abordons maintenant la complexité des problèmes, cherchant à les classer selon la performance des algorithmes qui les résolvent. Si un problème peut être résolu par un algorithme dont la complexité est bornée par un polynôme en fonction de la taille des données, il est qualifié de polynomial sinon il est dit exponentiellement difficile. Cette distinction est essentielle pour comprendre les limites et les défis rencontrés dans le domaine de l'algorithme et de la résolution de problèmes .

2.3.1 La classe P

La classe de complexité P regroupe les problèmes de décision qui peuvent être résolus de manière efficace par un algorithme déterministe en temps polynomial dans le pire des cas. Typiquement, ces algorithmes ont une complexité temporelle de la forme $O(n^k)$, où n est la taille de l'entrée et k est une

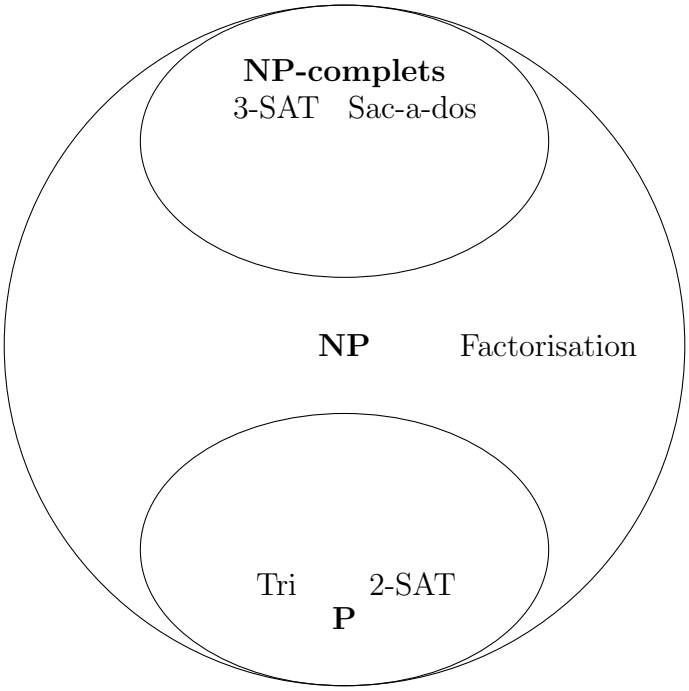


FIGURE 2.1 – Classes de Problèmes

constante dépendant de l'algorithme. Trouver un algorithme avec la plus petite constante k pour un problème donné est souvent un défi. Les problèmes de la classe P sont considérés comme faciles car ils peuvent être résolus efficacement. Il est également noté que les versions de décision et d'optimisation d'un même problème ont la même complexité temporelle.

2.3.2 La classe NP

La classe de complexité NP (Non-Deterministic Polynomial) regroupe les problèmes de décision pour lesquels il existe un algorithme non déterministe permettant de les résoudre en temps polynomial. En d'autres termes, il existe un algorithme polynomial capable de vérifier une solution (certificat) au problème en un temps polynomial par rapport à la taille des données. Les problèmes de la classe NP sont ceux pouvant être résolus en énumérant toutes les solutions possibles et en les testant à l'aide d'un algorithme polynomial.

Il est évident que $P \subseteq NP$, car si une solution peut être trouvée en temps polynomial, elle peut forcément être vérifiée en temps polynomial. La grande question en informatique reste de savoir si $P = NP$ ou si $P \neq NP$. De nombreuses personnes pensent que l'inclusion est stricte, mais cela reste à prouver. Cette intuition repose sur le fait qu'il est généralement plus facile de

vérifier si un certificat est une solution que de trouver la solution elle-même.

2.3.3 Problèmes NP-complets

La principale raison sous-tendant la conjecture $P \neq NP$ réside dans l'existence de la classe des problèmes NP-complets. Ces problèmes représentent le summum de la difficulté au sein de la classe NP. Ils possèdent une propriété remarquable : s'il était possible de résoudre l'un de ces problèmes en temps polynomial, alors tous les problèmes NP pourraient être résolus en temps polynomial, ce qui conduirait à l'égalité P = NP. Jusqu'à présent, de nombreux problèmes ont été identifiés comme NP-complets, mais aucun algorithme polynomial n'a été découvert pour les résoudre.

Un problème de décision est NP-complet s'il satisfait deux conditions principales :
1. Il appartient à la classe NP.
2. Tous les problèmes de la classe NP peuvent être réduits à celui-ci via une réduction polynomiale [30].

Conclusion du chapitre 2

Ce chapitre a exploré la théorie de la complexité, mettant en lumière les concepts clés de complexité temporelle et spatiale, ainsi que les distinctions entre problèmes de décision et d'optimisation. Il a également souligné l'importance de classer les problèmes en différentes catégories, telles que P et NP, pour mieux comprendre leurs difficultés respectives et les ressources nécessaires pour les résoudre. En résumé, la théorie de la complexité fournit un cadre essentiel pour évaluer l'efficacité des algorithmes et la difficulté des problèmes, guidant ainsi les avancées en informatique théorique et appliquée.

CHAPITRE 3

QUELQUES PROBLÈMES NP-COMPLET

Ce chapitre présente un état de l'art sur quelques problèmes classiques de l'optimisation combinatoire : **le problème de stable**, **le problème de coloration**, **le problème de dominant**, et **le problème de transversal minimum**. Ces problèmes sont fondamentaux en théorie des graphes et trouvent de nombreuses applications pratiques et théoriques.

3.1 Problème de stable

Un stable (ou ensemble indépendant)dans un graphe est un ensemble de sommets deux à deux non adjacents. Un stable est qualifié de maximum si aucun autre stable du graphe n'a plus de sommets. Un stable est maximal s'il ne peut être agrandi par l'ajout d'un autre sommet du graphe. Il est important de noter qu'un stable maximum est toujours maximal, mais l'inverse n'est pas forcément vrai. La taille d'un stable maximum dans un graphe G est appelée nombre de stabilité de G et est notée $\alpha(G)$. Le calcul du paramètre $\alpha(G)$ est un problème NP-complet [27]. Il reste NP-complet dans la classes des graphes K_3-free (ou graphe sans triangle) [31] et aussi dans la classes de graphes planaires [21].

Des exemples de stables maximaux et maximum dans le graphe de Petersen sont illustrés dans la figure 3.1.

Le problème de l'ensemble stable maximum peut être appliqué dans les domaines suivants

1. **Réseaux de communication :** Application : Dans les réseaux sans fil, un ensemble stable maximum peut être utilisé pour modéliser des

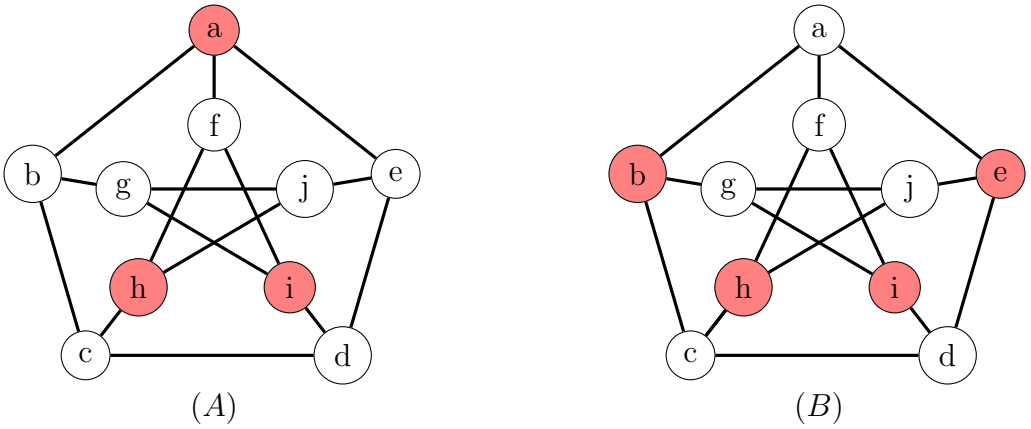


FIGURE 3.1 – $\{a, h, i\}$ Un stable maximal, et $\{b, e, h, i\}$ un stable maximum

problèmes de planification de fréquences ou de couverture de réseau, où l'objectif est de minimiser les interférences entre stations **??**.

2. **La Chimie moléculaire :** en chimie théorique, l'ensemble stable maximum est utilisé pour modéliser des molécules en termes de structures stables ou d'interactions entre atomes. Les graphes moléculaires sont étudiés pour identifier des ensembles d'atomes indépendants ou non interagissants [25].
3. **Planification et d'ordonnancement :** le problème de l'ensemble stable maximum est lié à l'ordonnancement des tâches et à la gestion des ressources. Dans ces contextes, il est utilisé pour organiser des tâches de manière à minimiser les conflits ou les chevauchements, comme dans la planification de cours universitaires ou d'examens [24].

Les stables et les cliques sont également liés de manière simple. Une clique dans un graphe est un ensemble de sommets deux à deux adjacents. La taille maximum d'une clique dans un graphe G est notée $\omega(G)$. Notons qu'un ensemble de sommets C forme une clique dans un graphe G si et seulement si ce même ensemble C est un stable dans le graphe complémentaire $\overline{G}$. Plus présisement,

$$\omega(G) = \alpha(\overline{G})$$

Ainsi, toute affirmation sur les stables peut être reformulée en termes de cliques. Par conséquent, de trouver une clique maximum dans un graphe est aussi NP-complet. Toutefois, dans le cas des graphes bipartis, un stable maximum peut être trouvé en temps polynomial à l'aide de techniques de programmation linéaire en nombres entiers (PLNE). On peut cependant caracteriser certaines classes de graphes dans lesquelles le Problème est poly-

nomial, par exemple les graphes triangules [5], les graphes serie-parallele [34] etc. . .

Ci-dessous, nous présentons un problème concret, proposé par Shanon en 1956, dont la solution est donnée par la taille d'un stable maximum dans un graphe.

Exemple 3.1.1. *[6]Transmission de messages par un canal bruité*

Un transmetteur sur un canal de communication est capable d'envoyer des signaux appartenant à un certain ensemble fini (ou alphabet) A. Certaines paires de ces signaux sont si semblables l'un à l'autre qu'ils pourraient être confondus par le récepteur à cause des distorsions possibles au cours de la transmission. Étant donné un entier strictement positif k, quel est le plus grand nombre de suites de signaux (ou mots) de longueur k qui peuvent être transmises sans confusion possible par le récepteur ?

Pour traduire ce problème en théorie des graphes, nous avons besoin du concept de produit fort de deux graphes G et H. Celui-ci est le graphe $G \boxtimes H$ dont l'ensemble de sommets est $V(G) \times V(H)$, deux sommets (u, x) et (v, y) étant adjacents si et seulement si $uv \in E(G)$ et $x = y$, ou $u = v$ et $xy \in E(H)$, ou $uv \in E(G)$ et $xy \in E(H)$.

Notons G le graphe dont l'ensemble de sommets est A, dans lequel deux sommets u et v sont adjacents s'ils représentent des signaux qui peuvent être confondus l'un avec l'autre. Soit G^k le produit fort de k copies de G. Ainsi, G^k est le graphe dont les sommets sont les mots de longueur k sur A, dans lequel deux mots distincts $(u_1, u_2, \ldots, u_k)$ et $(v_1, v_2, \ldots, v_k)$ sont reliés par une arête si $u_i = v_i$ ou $u_i v_i \in E(G)$, pour tout $1 \leq i \leq k$. Autrement dit, deux mots distincts sont adjacents dans G^k s'il y a possibilité que l'un d'eux puisse être pris pour l'autre par le récepteur. Il s'ensuit que le nombre maximum de mots de longueur k ayant la propriété souhaitée est tout simplement la stabilité de G^k.

Par exemple, si $A = \{0, 1, 2, 3, 4\}$ et que tout signal i peut être confondu avec $i-1$ ou $i+1$ (mod 5), alors $G = C_5$. Un dessin de $G^2 = C_5^2$ sur le tore, ainsi qu'un stable d'ordre 5 indiqué par les points noirs, est donné Figure 3.2. Notons que, comme le graphe est dessiné sur le tore, les quatre sommets des coins représentent un seul et même sommet, (0, 0). Cela montre que $\alpha(G^2) \geq 5$. On peut en fait vérifier que $\alpha(G^2) = 5$. Ainsi, dans ce cas, un maximum de cinq mots de longueur 2, par exemple 00, 12, 24, 31, 43, peut être transmis sans risque de confusion par le récepteur.

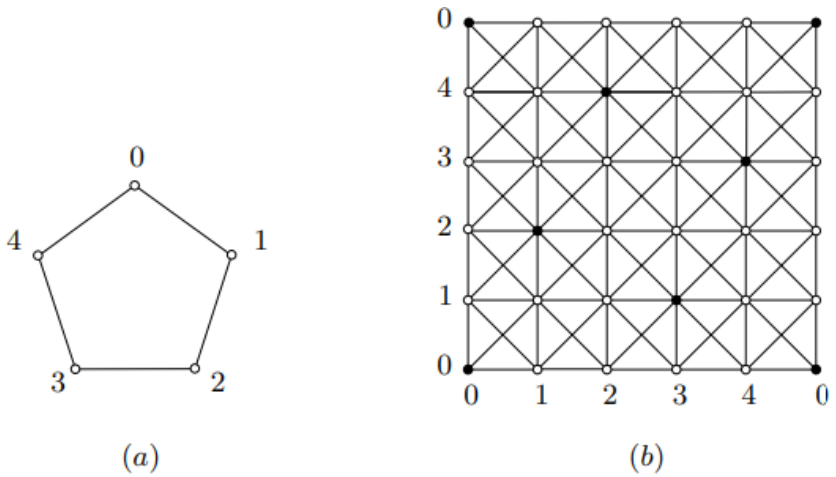


FIGURE 3.2 – (a) $C5$, (b) un stable à cinq sommets dans C_5^2

3.2 Problème de coloration

La coloration dans les graphes simples sans boucles ni arêtes multiples consiste à partitionner les sommets en ensembles indépendants, appelés stables, et à leur attribuer des couleurs de manière à ce que deux sommets adjacents aient des couleurs différentes. La coloration minimum est la coloration valide utilisant le moins de couleurs possible (nombre chromatique). Le nombre chromatique, noté $\chi(G)$, représente le nombre minimum de couleurs nécessaires pour cette coloration. Les premiers résultats de coloration de graphe se sont concentrés sur les graphes planaires, notamment lorsqu'il s'agissait de colorier des cartes géographiques[16, 29]. Dans le domaine de la théorie des graphes, le problème de la coloration est un sujet classique.

1. Problème de décision de la coloration :

Entrée : Un graphe G et un entier k.
Sortie : Oui si G peut être coloré avec k couleurs de manière à ce que deux sommets adjacents aient des couleurs différentes sinon, Non.

---

2. Problème d'optimisation de la coloration :

Entrée : Un graphe G.
Sortie : Le nombre minimum de couleurs ($\chi(G)$) nécessaires pour colorer G de manière valide, c'est-à-dire sans que deux sommets adjacents aient la même couleur.

Des exemples de coloration et coloration minimum dans le graphe de Petersen sont illustrés dans la figure 3.3.

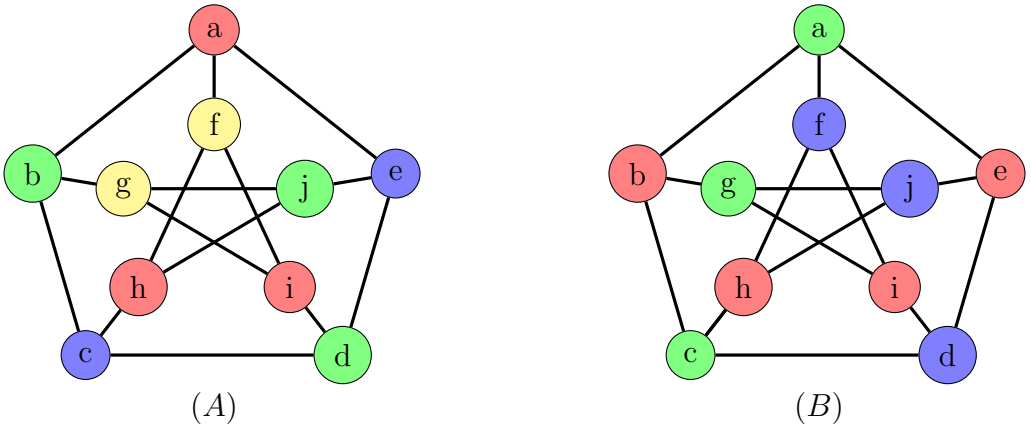


FIGURE 3.3 – Graphe (A) coloré, et graphe (B) avec coloration minimum

Le problème de coloration avec k couleurs devient NP-complet dès que k dépasse 2 [20]. Concernant le nombre chromatique d'un graphe, déterminer cette valeur est NP-difficile dans le cas général. En d'autres termes, sauf si P=NP. Il n'existe pas d'algorithme polynomial capable de résoudre ce problème pour n'importe quel graphe donné. Cette difficulté algorithmique pose une question d'optimisation :

pour un graphe donné G, quel est le nombre minimum de couleurs requis pour une coloration valide de ce graphe ?

Le problème de coloration de graphe a de nombreuses applications dans divers domaines de la science, de l'ingénierie, de la gestion, et même des domaines plus créatifs. Voici quelques-uns de ces domaines :

1. **Planification et ordonnancement** : Le problème de la coloration de graphe est couramment utilisé pour modéliser des problèmes d'ordonnancement. Par exemple, dans la planification d'examens ou la gestion de ressources, il est utilisé pour éviter des conflits (comme deux examens pour des étudiants communs à la même heure)[28].
2. **Affectation de fréquences dans les réseaux de communication sans fil** : Dans les réseaux sans fil, la coloration de graphes peut être utilisée pour attribuer des fréquences à différentes stations de manière à éviter des interférences entre stations adjacentes. Chaque fréquence est représentée par une couleur, et les stations adjacentes (dans le graphe de couverture) doivent recevoir des couleurs différentes [23].
3. **Optimisation de la mémoire dans les compilateurs (coloration des registres)** : La coloration de graphe est utilisée dans la phase de

génération de code des compilateurs, particulièrement pour l'allocation de registres. Le problème consiste à attribuer un registre physique (ou mémoire) à chaque variable de manière à éviter les conflits entre les variables qui sont actives en même temps [7].

4. **Coloration des cartes (coloration géographique)** : Le problème classique des quatre couleurs en géographie est un exemple célèbre de coloration de graphe. Ce problème consiste à colorer les régions d'une carte de manière à ce que deux régions adjacentes ne partagent pas la même couleur. Ce problème est modélisé par un graphe planaire [1].
5. **Réseaux sociaux** : La coloration de graphe peut être utilisée pour analyser des communautés dans les réseaux sociaux, où l'objectif est de partitionner un réseau de relations en groupes distincts (couleurs) de manière à minimiser les conflits ou maximiser l'indépendance des groupes [14].

Exemple 3.2.1. *Soit un groupe de cinq étudiants admis aux examens de rattrapage dans les modules suivants : anglais (A), file d'attente (F), théorie des jeux (G), graphe et ordre (O), contrôle optimal (C), informatique (I), TIC (T).*

Chaque étudiant doit passer un seul examen par jour.

Problème : *Trouver le minimum de jours pour planifier ces examens..*

étudiant	modules
E 1	A,T,I
E 2	T,I,C
E 3	C,F,O
E 4	O,F,C,G
E5	O,T,C

TABLE 3.1 – Tableau associe à chaque étudiant les modul qui doit passé.

soit $G = (V, E)$ le graphe défini par : $V = \{A, T, C, I, O, F, G\}$

Deux modules sont adjacents si et seulement si on ne peut pas les programmer le même jour (c'est-à-dire s'il y a au moins un étudiant concerné par les deux modules). Ainsi, les modules programmés le même jour ne doivent pas être adjacents deux à deux dans G, et donc forment un stable.

Par conséquent, pour chaque jour correspond un stable dans G, et donc le nombre minimum de jours est donné par le nombre minimum de stables dans G, ce qui est exactement égal au nombre minimum de couleurs.

On applique l'algorithme Lex-BFS à G pour vérifier s'il admet un ordre d'élimination simplicial. Donc, on trouve que le graphe est triangulé.

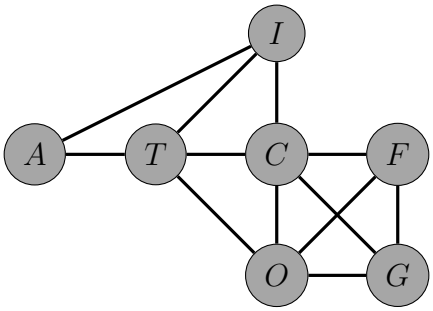


FIGURE 3.4 – Graphe G

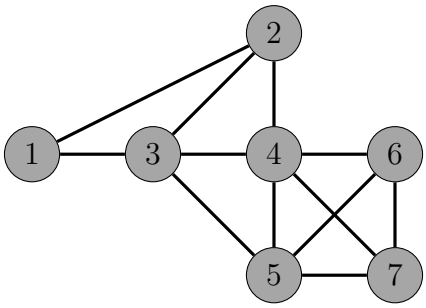


FIGURE 3.5 – Graphe (G) ordonner par lex-BFS

Pour colorier un graphe triangulé, on applique l'algorithme suivant :

Algorithm 2 Colorier un graphe triangulé

Données Le graphe G peut être coloré avec k couleurs .

Résultat Coloration du graphe G.

1: Calculer l'ordre d'élimination simplicial (O.E.S).
2: Considérer l'ordre inverse de l'O.E.S.
3: Donner la plus petite couleur non utilisée au plus grand sommet non encore coloré et en parcourant l'ordre décroissant des sommets, attribuer cette couleur dès que cela est possible.
4: Répéter l'étape 3 jusqu'à la fin.

Puisque le nombre de couleurs $\gamma = 4$, la planification des examens nécessite quatre jours.

3.3 Problème de transversal minimum

Le problème du **transversal** minimum, appelé aussi problème de couverture par des sommets (vertex cover problem) consiste à trouver le plus petit

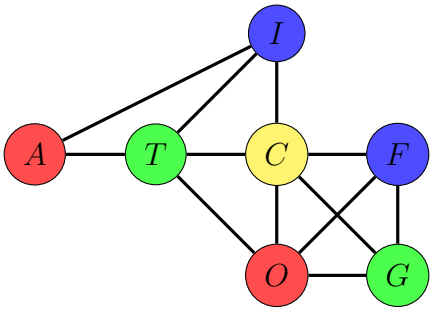


FIGURE 3.6 – coloration de Graphe G

ensemble de sommets d'un graphe qui couvre au moins une extrémité de chaque arête. C'est un problème d'optimisation bien connu, répertorié parmi les 21 problèmes de Karp [27] classés NP-complet. Ce même problème reste aussi NP-complet sur la classe des graphes planaires. Transversal minimal est un ensemble de sommets minimal qui couvre toutes les arêtes. Transversal minimum est un ensemble de sommets de taille minimale possible. Dans ce problème, l'objectif est de trouver un transversal de taille minimale possible, notée $\tau(G)$. En termes simples, $T \subseteq V$ est un transversal si et seulement si le sous-graphe induit par $V \setminus T$ est un stable.

Des exemples de transversal minimal et minimum dans le graphe de Petersen sont illustrés dans la figure 3.7.

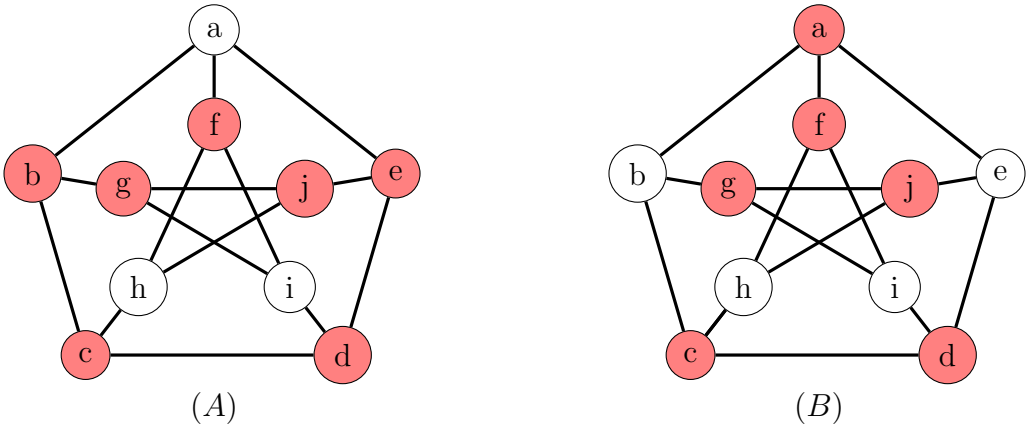


FIGURE 3.7 – $\{b, c, d, e, f, j, g\}$ Un transversal minimal, et $\{a, c, d, f, i, g\}$ un transversal minimum

Theorem 3.3.1. *[30] Pour tout graphe biparti, la cardinalité d'un couplage maximum est égale à la cardinalité d'un transversal minimum. Autrement dit*

$$\nu(G) = \tau(G)$$

Le théorème ci-dessus montre que les problèmes de calcul des paramètres $\nu(G)$ et $\tau(G)$ sont équivalents en terme de difficulté quand le graphe G est biparti. Par ailleurs, Edmonds, présente un algorithme polynomial pour le problème du couplage maximum dans un graphe quelconque [15], et donc le problème est plus simple sur les graphes bipartis. Par conséquent, le problème de transversal est aussi polynomial sur les graphes bipartis. Le problème d'optimisation associé au problème de couverture minimum par sommets, est le suivant :

Entrée : un graphe G
Question : quel est le plus petit entier k, tel qu'il existe une couverture par sommets de G de taille k ?

Exemple 3.3.2. *Soit un entrepôt divisé en 7 zones : $Z_1, Z_2, Z_3, Z_4, Z_5, Z_6, Z_7$. qui représentent Les arêtes, et les capteurs sont les intersections de ces arêtes qui représentent les sommets.*

Nous avons les capteurs suivants, chacun couvrant certaines zones :
- Capteur A : $\{Z_1, Z_2, Z_3\}$ - Capteur B : $\{Z_4, Z_5\}$ - Capteur C : $\{Z_6, Z_7\}$
- Capteur D : $\{Z_1, Z_6\}$ - Capteur E : $\{Z_2, Z_4\}$ - Capteur F : $\{Z_3, Z_5, Z_7\}$
Pour que chaque zone soit couverte, nous devons sélectionner un ensemble de capteurs tel que chaque zone Z_i soit surveillée par au moins un capteur.

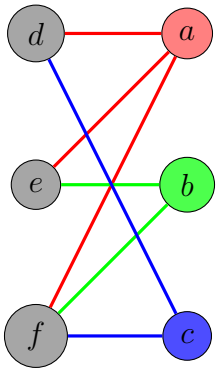
Pour trouver le transversal minimum, nous cherchons le plus petit ensemble de capteurs qui couvre toutes les zones. Nous pouvons reformuler cela en un problème de couverture d'ensembles où Nous cherchons le sous-ensemble minimal de capteurs dont les zones surveillées forment une couverture de toutes les zones $\{Z_1, Z_2, Z_3, Z_4, Z_5, Z_6, Z_7\}$.

Étapes de résolution :
1. Listez toutes les couvertures possibles.
2. Trouvez celle avec le moins de capteurs.

Une solution possible est de sélectionner les capteurs A, C et E :
- Capteur A couvre Z_1, Z_2, Z_3
- Capteur B couvre Z_4, Z_5
- Capteur C couvre Z_6, Z_7

Ainsi, toutes les zones $\{Z_1, Z_2, Z_3, Z_4, Z_5, Z_6, Z_7\}$ sont couvertes, et on utilise seulement trois capteurs. Après vérification, il s'avère que trois capteurs sont effectivement le nombre minimum nécessaire pour cette couverture, car toute autre combinaison de deux capteurs échoue à couvrir toutes les zones.

Le concept de transversal minimum est utilisé ici pour optimiser le placement de capteurs en minimisant leur nombre tout en assurant une couverture



G

FIGURE 3.8 – (G) Graphe pour optimiser le placement de capteurs

complète des zones. Ce type de problème a des applications dans divers domaines tels que la surveillance, la maintenance de réseaux, et même dans la planification de tests ou de sondages.

3.4 Problème de l'ensemble dominant

Un ensemble dominant dans un graphe $G = (V, E)$ est un sous-ensemble de sommet D tel que tout sommet du graphe soit, il est dans D ou il a un voisin dans D. Autrement dit : $D \subset V$ est un ensemble dominant si

$$\forall v \in V, v \in D \vee D \cap N(v) \neq \emptyset$$

Le dominant minimal est un ensemble dominant où tous les sommets sont nécessaires pour garder la propriété de domination. Le dominant minimum est l'ensemble dominant ayant la plus petite taille. Des exemples de dominant minimum et minimal dans le graphe de Petersen sont illustrés dans la figure 3.9.

Le problème de l'ensemble dominant (dominating set) consiste à déterminer dans un graphe donné un ensemble dominant ayant une taille $|D|$ la plus petite possible. La version décision de ce problème est :

Entrée : Un graphe $G = (V, E$ et un entier positif k.
Sortie : Existe-il dans G un ensemble dominant verifiant $|D| \leq k$?

Le problème de l'ensemble dominant est un problème NP-complet [27]. il est parmi les problèmes les plus difficiles. En effet, il est NP-complet sur de

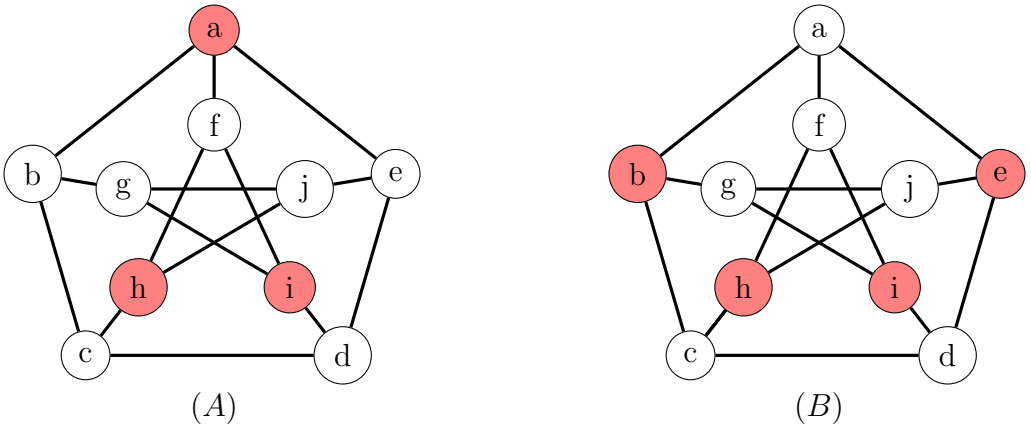


FIGURE 3.9 – $\{a, h, i\}$ Un dominant minimum, et $\{b, e, h, i\}$ un dominant minimal

nombreuses classes de graphes particulières y compris les graphes biparti [12] et les graphes triangulés [21], malgré le fait que d'autres problèmes comme le stable, la coloration de graphes soient résolus en temps polynomial dans ces classes. Quant aux domaines d'application de ceux problème, il sont très variés. On le rencontre dans :

1. **Réseaux de capteurs sans fil (Wireless Sensor Networks)** [22] : Dans les réseaux de capteurs, un ensemble dominant peut être utilisé pour définir un sous-ensemble de capteurs qui couvrent efficacement toute la zone d'un réseau, réduisant ainsi le nombre de capteurs actifs tout en garantissant la surveillance de l'ensemble du réseau.
2. **Planification et optimisation des réseaux de communication** [36] : L'ensemble dominant est utilisé pour minimiser le nombre de relais dans les réseaux de télécommunications tout en assurant que chaque noeud est connecté à au moins un relais, optimisant ainsi la communication dans les réseaux.
3. **Biologie computationnelle et analyse des réseaux biologiques** [17] : Dans l'analyse des réseaux biologiques, les ensembles dominants sont utilisés pour identifier des ensembles de protéines clés ou d'organismes qui influencent ou interagissent avec un grand nombre d'autres entités dans le réseau.
4. **Couverture dans les réseaux sociaux** [35] : Dans les réseaux sociaux, l'ensemble dominant peut être utilisé pour identifier un petit sous-ensemble de personnes influentes (leaders d'opinion), qui peuvent atteindre efficacement le reste du réseau par des connexions directes.

5. **Optimisation de la distribution des ressources** [3] : L'ensemble dominant est appliqué dans la gestion des ressources pour réduire le nombre de points de distribution nécessaires tout en assurant que chaque emplacement de demande est couvert.

Exemple 3.4.1. *Nous voulons construire des écoles, pour que chaque école puisse accueillir les habitants de la région et des zones voisines. L'objectif est de placer le minimum d'écoles. Il peut donc être converti en Problème de dominant. Soit le graphe de la figure 3.10, où les sommet représente les villes et deux ville sont voisin si il y a une arrêt entre ces sommets.*

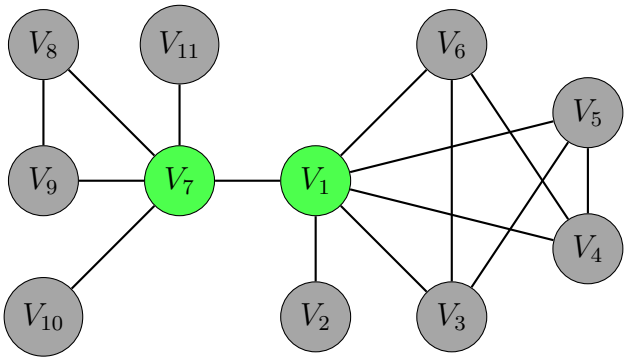


FIGURE 3.10 – Graphe représente un ensemble dominant minimum

L'ensemle dominant minimum dans le graphe est $D = \{V_1, V_7\}$. *Donc le minimum des ecoles que en peut placer est deux*

Conclusion du Chapitre 3

En résumé, ce chapitre a exploré quatre problèmes fondamentaux en théorie des graphes : le problème du stable, de la coloration, de la dominance et du transversal. Ces concepts sont essentiels pour comprendre les structures et les propriétés des graphes, avec des applications variées en optimisation, en informatique théorique, et dans des domaines pratiques comme la planification, la communication, et la gestion des réseaux. Les défis algorithmiques que posent ces problèmes, notamment leur complexité NP-difficile, soulignent l'importance de développer des méthodes efficaces pour les résoudre dans des cas spécifiques.

CHAPITRE 4

GRAPHE DISTANCE HÉRÉDITAIRE

4.1 Graphe distance héréditaire

Un graphe distance héréditaire (distance-hereditary graph) est un type particulier de graphe dans lequel pour tout sous-graphe induit et connexe H de G, on a :

$$\forall u, v \in V(H) \mid d_H(u, v) = d_G(u, v)$$

Chaque sous-graphe induit hérite les mêmes distances que le graphe original. Les graphes distance héréditaire sont des **graphes parfaits** (perfect graphs) [13] "Un graphe est parfait si et seulement si ni lui ni son complémentaire ne contiennent de cycle impair induit de longueur au moins cinq", comme démontré par Olaru et Sachs en 1970 . Cette classe de graphes a été étudiés pour la première fois par Howorka en 1977[26]. Elle est parmi les classes de graphes qui ont été intensivement étudiés d'un point de vue algorithmique [8]. Une grande variété d'outils ont été utilisés, comme les ordonnancements Lex-BFS ou le calcul d'une séquence d'élagage caractéristique introduit par Bandelt et Mulder , mais il existe actuellement deux schémas généraux de résolution de problèmes qui s'appliquent aux graphes de distance héréditaire, c'est-à-dire des algorithmes basés sur largeur de clique limitée et programmation dynamique avec un arbre de décomposition spécial. Bien que ces deux axes d'investigation indépendants utilisent en réalité le même arbre étiqueté, dérivé d'une séquence d'élagage du graphe d'entrée, ils génèrent des résultats différents par la suite.

Theorem 4.1.1. *[10] Les conditions suivantes sont équivalentes :*
1. G est distance héréditaire.

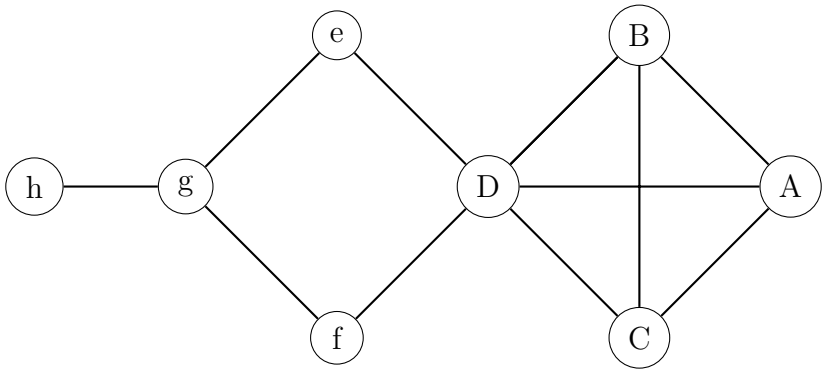


FIGURE 4.1 – Graphe distance héréditaire

2. G n'a pas de sous-graphe isomorphe à un graphe de la figure 4.2 ou à un C_k $(k \geq 5)$.
3. tous les C_k $(k \geq 5)$ *de G ont deux cordes croisées.*
4. Tous les sous-graphes induits de G ont un sommet pendant ou une paire de sommets jumeaux.

La propriété 4 permet de détruire un graphe distance héréditaire en appliquant récursivement des opérations de contraction de sommets jumeaux et de suppression de sommets pendants, processus appelé élagage.

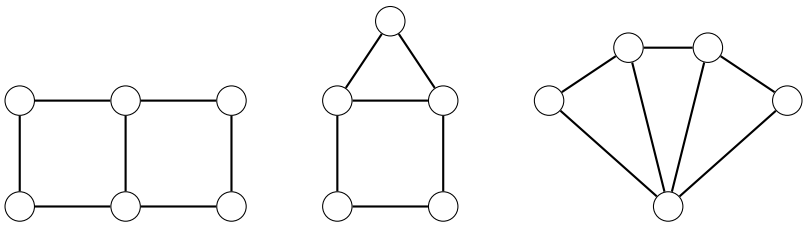


FIGURE 4.2 – Les graphes domino, maison et diamant respectivement

Définition 4.1.1. *[10] Une séquence d'élagage d'un graphe G est une séquence de mots* $(s_2, s_3, \ldots, s_n)$ *telle qu'il existe une numérotation des sommets de G de* $1, \ldots, n$ *vérifiant, pour tout* $i \in \{2, \ldots, n\}$ *:*

1. Si s_i *est un mot de la forme* P_j, F_j *ou* T_j *avec* $j < i$.

2. $G(\{1, \ldots, i\})$ *s'obtient à partir de* $G(\{1, \ldots, i-1\})$ *en insérant le sommet numéro i de manière à ce qu'il soit respectivement un sommet pendant, un faux ou un vrai jumeau du sommet numéro j.*

La lettre P, F ou T est appelée le type du sommet numéro i, et j est appelé le sommet relatif de i. Signifie respectivement du sommet pendant, faux jumeau et Vrai jumeau.

Corollaire 4.1.2. *[10] Un graphe G a une séquence d'élagage ssi G est distance héréditaire.*

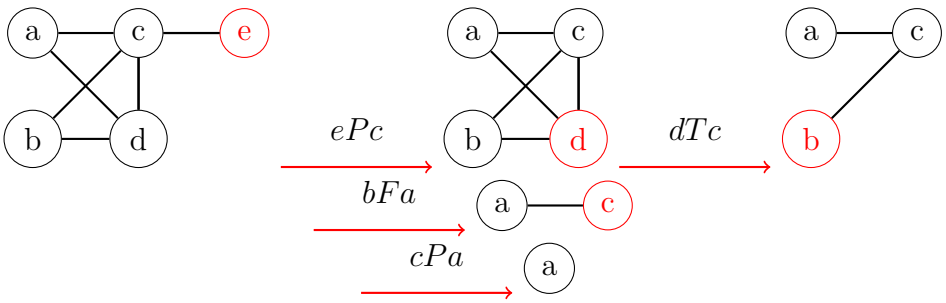


FIGURE 4.3 – Exemple séquence d'élagage

[8] Propose un nouvel algorithme simple pour calculer des ensembles stables maximaux pour les graphiques distance héréditaires. Il corrige l'algorithme présenté par Hammer et Maffray [Discrete Appl. Mathématiques. 27 (1990) 85–99], mais reste dans le cadre général de schéma proposé par Hammer et Maffray (1990).

4.1.1 Algorithme de reconnaissance

Cet algorithme se divise en deux phases. La première phase consiste à calculer une séquence d'élagage candidate (S.E.C). La deuxième phase de l'algorithme vérifie la validité de cette séquence d'élagage candidate, voir [11] pour plus de détails.

Theorem 4.1.3. *[10] L'algorithme Calcule S.E.C (G) calcule une séquence d'élagage de G si et seulement si G est distance héréditaire.*

- Dans les lignes 9, 13 et 17, x est un sommet pendant et on peut donc le détruire en insérant en début de la S.E.C le mot xPy avec y le sommet relatif, c'est-à-dire le seul sommet adjacent à x.
- Dans la ligne 10 quand on vérifie si $G(L_{j-1})$ est un cographe, si c'est vrai alors T reçoit le coarbre, sinon on retourne faux, G n'est pas distance héréditaire.
- Les opérations dans les lignes 7,12 et 16 . Contracter les composantes connexes de $G(L_j)$ et contracter chaque $N^-(x)$ en un seul sommet va se faire à l'aide du coarbre T préalablement calculé (effectuées en appelant l'algorithme Elague Cographe). Il faut avoir un accès direct sur les feuilles associées aux sommets.

L'algorithme **Vérifie Séquence Elagage** retourne Vrai ssi la séquence d'élagage candidate est une séquence d'élagage de G.

Algorithm 3 [10] Elagage Cographe

Données $G = (V, E)$ un cographe
Résultat Construit la séquence d'élagage de G.

$T \leftarrow$ le coarbre de G
$D \leftarrow$ la liste des noeuds de T n'ayant que des feuilles pour fils
tant que $D \neq \emptyset$ **faire**
    $n \leftarrow$ un noeud de D
    $x \leftarrow$ un fils de n
    pour Tout les autres fils y de n **faire**
        si le type de n est Série **alors**
            Ajouter en début de la séquence d'élagage yTx
        sinon
            Ajouter en début de la séquence d'élagage yFx
        fin si
    fin pour
    Remplacer le noeud n par x
    si le père de n n'a plus que des feuilles pour fils **alors**
        Ajouter le père de n dans D
    fin si
fin tant que

Algorithm 4 [10] Calcule S.E.C

Données $G = (V, E)$ un graphe non orienté
Résultat Une séquence d'élagage candidate de G

$W \leftarrow V$
tant que $W \neq \emptyset$ **faire**
    $a \leftarrow$ un sommet de W
    Calculer les niveaux de distances $L1, L2, \ldots, Lp$ à partir de a
    Vérifier que $G(Lp)$ est un cographe
    pour $j \leftarrow p, p-1, \ldots, 2$ **faire**
        Contracter chaque composante connexe de $G(Lj)$
        Trier les sommets x de Lj par ordre croissant sur $d^-(x)$
        Détruire les sommets x de Lj tels que $d^-(x) = 1$
        Vérifier que $G(Lj-1)$ est un cographe
        pour chaque $x \in Lj$ pris dans l'ordre croissant sur $d^-(x)$ **faire**
            Contracter $N^-(x)$ en un seul sommet
            Détruire x
        fin pour
    fin pour
    Contracter chaque composante connexe de $G(L1)$
    Détruire les sommets restants x de $L1$
    $W \leftarrow W \setminus \{a\}$
fin tant que

Algorithm 5 [10] Vérifie Séquence Élagage

Données $(s_2, s_3, \ldots, s_n)$ une séquence d'élagage candidate et G le graphe
Résultat Vrai si $(s_2, s_3, \ldots, s_n)$ est une séquence d'élagage de G, Faux sinon.

pour $j \leftarrow n, n-1, \ldots, 2$ **faire**
    $xSy \leftarrow s_j$
    si $S = P$ **alors**
        si $d(x) \neq 1$ ou $xy \notin E$ **alors**
            Retourner Faux
        sinon
            si $N(x) \cap \{x_1, x_2, \ldots, x_{j-1}\} \neq N(y) \cap \{x_1, x_2, \ldots, x_{j-1}\}$ **alors**
                Retourner Faux
            fin si
        fin si
    fin si
fin pour
Retourner Vrai

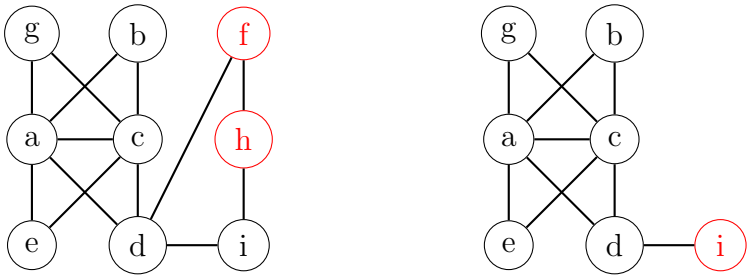
Exemple 4.1.4. *Nous suivrons les étapes de l'algorithme 4 sur la figure 4.4 pour calculer S.E.C*

- *en Calcul les niveaux de distances à partir du sommet a donc :* $L_1 = \{b, c, d, e, g\}$, $L_2 = \{f, i\}$, $L_3 = \{h\}$
- $G(L_3)$ *est un cographe*
  - $d^-(h) = 2$
  - $G(L_2)$ *est un cographe*
    - $S = \{fFi\}$
    - $S = \{hPi, fFi\}$
  - $d^-(i) = 1$
  - $S = \{iPd, hPi, fFi\}$
  - $G(L_1)$ *est un cographe*
- $S = \{cTb, eFb, dFe, gFb, iPd, hPi, fFi\}$
- $S = \{bPa, cTb, eFb, dFe, gFb, iPd, hPi, fFi\}$

Donc la séquence d'élagage candidate (S.E.C) est $S = bPa, cTb, eFb, dFe, gFb, iPd, hPi, fFi\}$. *Nous allons maintenant vérifier si la séquence d'élagage candidate est une séquence d'élagage de G*

- *Soit* $\{a = x_1, b = x_2, c = x_3, e = x_4, d = x_5, g = x_6, i = x_7, f = x_8, h = x_9\}$
- $j = 8$, fFi, $N(f) \cap \{x_1, x_2, \ldots, x_7\} = N(i) \cap \{x_1, x_2, \ldots, x_7\}$
- $j = 7$, hPi, $d(h) = 1$ *et* $(h, i) \in E$
- $j = 6$, iPd, $d(i) = 1$ *et* $(i, h) \in E$
- $j = 5$, gFb, $N(d) \cap \{x_1, x_2, \ldots, x_4\} = N(e) \cap \{x_1, x_2, \ldots, x_4\}$
- $j = 4$, dFe, $N(c) \cap \{x_1, x_2, x_3\} = N(a) \cap \{x_1, x_2, x_3\}$
- $j = 3$, eFb, $N(g) \cap \{x_1, x_2\} = N(b) \cap \{x_1, x_2\}$
- $j = 2$, cTb, $N(c) \cap \{x_1\} = N(b) \cap \{x_1\}$
- $j = 1$, bPa, $d(b) = 1$ *et* $(b, a) \in E$

La séquence d'élagage associée est :$(bPa, cTb, eFb, dFe, gFb, iPd, hPi, fFi)$. *en peut trouver une autre séquence d'élagage comme :* $(ePa, cTe, gFe, bFg, dFe, fPd, hPf, iFf)$



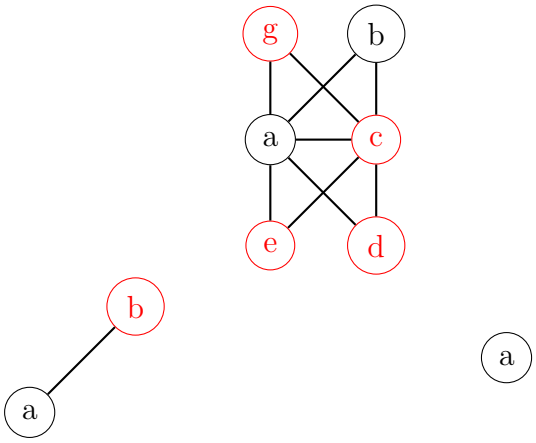


FIGURE 4.4 – Exemple d'un graphe distance héréditaire

4.2 Graphe biparti distance héréditaire (GBDH)

La plupart des problèmes d'optimisation combinatoire sont NP-complets, notamment ceux liés aux graphes. Cependant, il existe des classes de graphes spécifiques où ces problèmes deviennent plus faciles à résoudre. Étant donné que les graphes utilisés dans des contextes pratiques sont souvent particuliers, développer des algorithmes efficaces pour reconnaître ces classes de graphes est essentiel pour simplifier la résolution de problèmes complexes.

4.2.1 Reconnaissance des GBDH

Les graphes biparti distance héréditaire ont un algorithme linéaire pour les reconnaître, utilisons le parcours en largeur lexicographique (ou parcours LexBFS) . Nous montrons d'abord quelques définitions et propriétés importantes.

Définition 4.2.1. *Dans un graphe biparti $G = (X, Y, E)$, une arête $(x, y) \in E$ est dite bisimpliciale si elle satisfait la condition suivante :*

— *Soit $N(x)$ l'ensemble des voisins de x et $N(y)$ l'ensemble des voisins de y, alors pour chaque paire de sommets $u, v \in N(x) \cup N(y)$, il existe une arête entre u et v.*

Proposition 4.2.1. *[33]Soit xy une arête dans un graphe biparti G. Les deux conditions suivantes sont équivalentes :*

1. *xy est une arête bisimpliciale.*
2. *xy appartient à une unique biclique maximale.*

Définition 4.2.2. *Un sommet x est dit bisimplicial si toutes les arêtes incidentes à x sont des arêtes bisimpliciales.*

Exemple 4.2.2. *Dans le graphe G de la figure 4.5, l'arête x_1y_1 est bisimpliciale, car le sous-graphe $G[N(x_1) \cup N(y_1)]$, induit par $N(x_1) \cup N(y_1)$, forme une biclique. De même, x_4y_4 est également une arête bisimpliciale*

Dans le graphe G' de la figure 4.5, on peut vérifier que toutes les arêtes incidentes au sommet y_1 sont bisimpliciales, donc y_1 est un sommet bisimplicial.

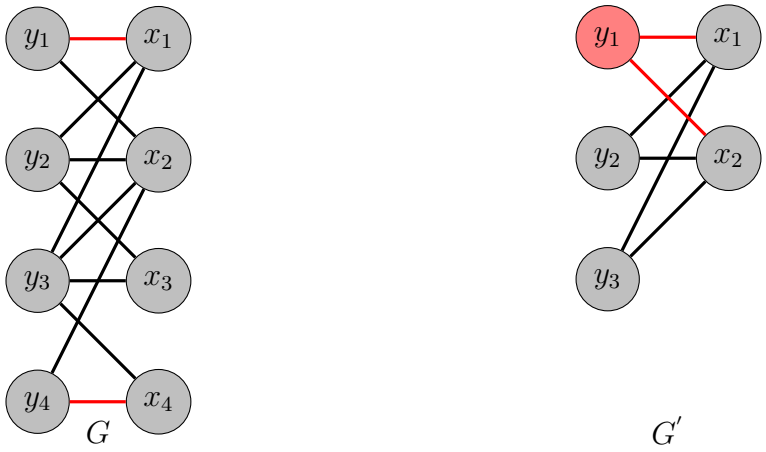


FIGURE 4.5 – Sommet et arête bisimplicial

Étant donné un graphe $G = (V, E)$, rappelons que le carré d'un graphe G est le graphe G^2 ayant les mêmes sommets que G. Deux sommets u et v sont adjacents dans G^2 (c'est-à-dire $u \sim_{G^2} v$) si et seulement si leur distance dans G est majorée par 2 (c'est-à-dire $d_G(u, v) \leq 2$).

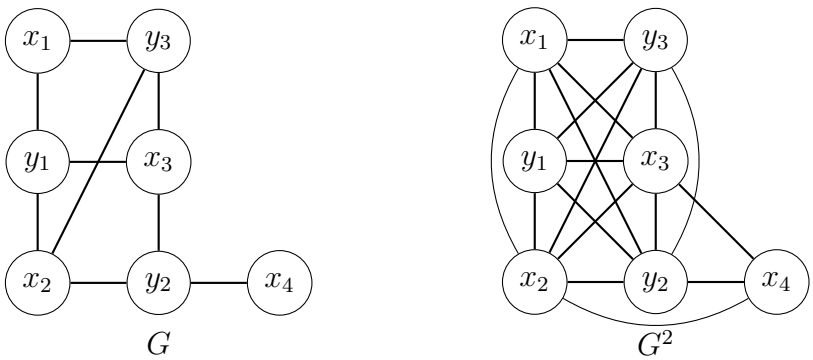


FIGURE 4.6 – Graphe G et son carré

Corollaire 4.2.3. *[33] Un graphe biparti G est distance héréditaire si et seulement si son carré G^2 est un graphe triangulé.*

Lamme 4.2.4. *[33] Soient $G = (V, E)$ un graphe biparti et v un sommet. Les deux assertions suivantes sont équivalentes :*

1. *v est bisimplicial dans G*
2. *v est simplicial dans G^2*

Theorem 4.2.5. *[33] Soient G un graphe biparti et $\sigma = (v_1, v_2, \ldots, v_n)$ un ordre LexBFS sur ses sommets. Les conditions suivantes sont équivalentes :*

1. *G est distance héréditaire ;*
2. *σ est un ordre d'élimination bisimplicial.*

Définition 4.2.3. *[33] Étant donné un graphe biparti distance héréditaire connexe $G = (X, Y, E)$, on définit la séquence de graphes connexes $H_0, H_1, \ldots, H_k$ comme suit :*

1. *$H_0 = G$;*
2. *pour $i > 0$:*

(a) H_i est le sous-graphe induit de H_{i-1} par la suppression d'un sommet bisimplicial x et tous ses voisins, excepté un seul, si x n'est pas pendant dans H_{i-1} ;

(b) H_i est le sous-graphe induit de H_{i-1} par la suppression d'un sommet bisimplicial x, si x est un sommet pendant dans H_{i-1}.

Theorem 4.2.6. *[33] Pour qu'un graphe G admette une séquence de graphes $H_0 = G, H_1, \ldots, H_k$ définie selon la définition 4.2.3 et convergeant vers un graphe $H_k = (V_k, E_k)$ vide, il faut et il suffit que G soit un graphe biparti distance héréditaire.*

Exemple 4.2.7. *On peut vérifier que l'ordre des sommets sur le graphe biparti G de la figure 4.7 est un LexBFS. De plus, cet ordre est un ordre d'élimination bisimplicial, donc G est un graphe biparti distance héréditaire.*

Le sommet x_1 est bisimplicial et ses voisins sont x_3, x_4 et x_5. On laisse x_5 et on supprime x_1, x_3 et x_4 et on obtient le sous-graphe H_1. x_2 est un sommet pendant dans H_1 , on le supprime et on obtient le sous-graphe H_2. En supprimant x_5 dans H_2, on obtient le graphe H_3 réduit au sommet x_6, et donc la séquence s'arrête.

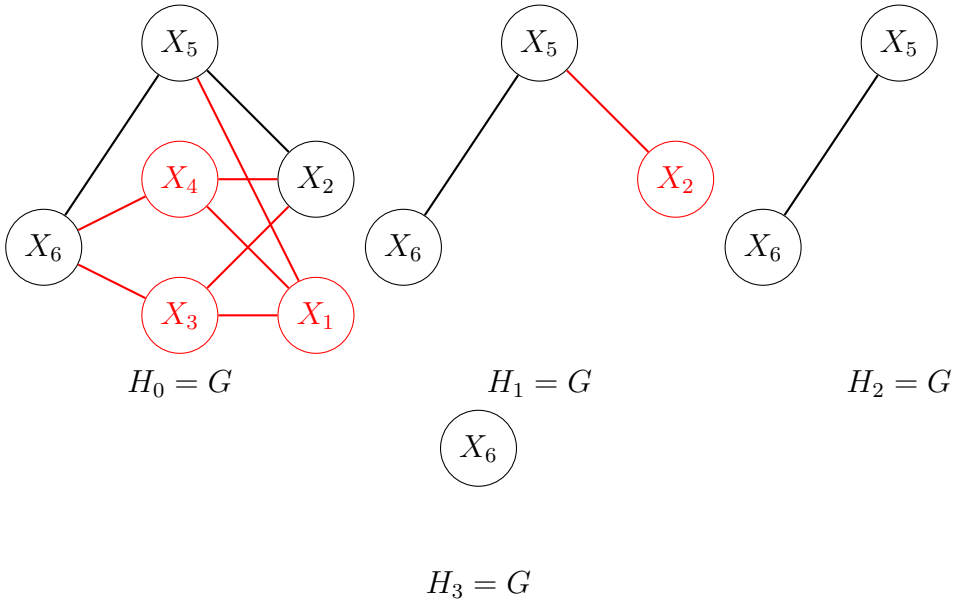


FIGURE 4.7 – Séquence de graphes calculés selon la définition 4.2.3

Lamme 4.2.8. *[33] Soit $G = (V, E)$ un graphe biparti distance héréditaire connexe et ordonné par LexBFS $\sigma = (v_1, v_2, \ldots, v_n)$. Alors, la restriction de σ sur le sous-graphe G' obtenu par la suppression de v_1 ou par la suppression de v_1 et de ses voisins, à l'exception d'un seul, est aussi un LexBFS.*

Algorithm 6 [33] Algorithme de reconnaissance d'un graphe biparti distance héréditaire

Données Un biparti connexe $G = (V, E)$ avec $|V| = n$, $|E| = m$ et un ordre lexBFS $\sigma = (x_1, x_2, \ldots, x_n)$.

Résultat $G = (V, E)$ est un graphe biparti distance héréditaire si et seulement si σ est un ordre d'élimination bisimplicial.

début
$W = V$, $x = x_1 = \min W$, $m(x) \in N^+(x)$ // $m(x)$ est l'un des voisins supérieurs de x
tant que $m(x) \neq x_n$ **faire** // si $m(x) = x_n$ alors W induit la biclique B_x
    si x est pendant **alors**
        $W = W - \{x\}$, $x = \min_\sigma W$ // $\min_\sigma$ = minimum pour σ dans W
    sinon
        pour tout $y \in N^+(x) \setminus m(x)$ **faire**
            si $N_W(y) \neq N_W(m(x))$ **alors**
                G n'est pas un (6, 2)-chordal
            sinon
                $W = W - N^+[x] \cup m(x); x = min_\sigma W$
            fin si
        fin pour
    fin si
fin tant que
Retourner vrai
Fin

L'algorithme 6 teste si LexBFS est un ordre d'élimination bisimplicial, et donc si G est un graphe biparti distance héréditaire, avec une complexité $O(m+n)$.

Exemple 4.2.9. *Dans la figure 4.8, nous avons un graphe biparti d'ordre 8. On peut vérifier facilement que l'ordre des sommets est un LexBFS. En applique maintenant l'algorithme 6 pour voir s'il est biparti distance héréditaire.*

- *On a* $W = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$, $x = x_1$ *et* $N^+(x_1) = \{x_4, x_5, x_6\}$, $m(x) = x_6$ *(le choix de* x_6 *est arbitraire, on aurait pu choisir* x_4 *ou* x_5*, car ils sont des jumeaux). Passons à l'itération 1 :*
- *1.* $m(x) = x_6 \neq x_8$ *; pour* $y \in N^+(x_1) \setminus \{x_6\} = \{x_4, x_5\}$, $N_W(y) = N_W(x_6) = \{x_1, x_2, x_3, x_8\}$*. Ceci implique que le sommet* $x = x_1$ *est bisimplicial. Posons*

$$W = \{x_2, x_3, x_6, x_7, x_8\}, \quad x = x_2 \quad et \quad N^+(x_2) = N_W(x_2) = \{x_6\}$$

et passons à l'itération 2.

- *2. Ceci implique que le sommet* $x = x_2$ *est pendant. Posons*

$$W = \{x_3, x_6, x_7, x_8\}, \quad x = x_3 \quad et \quad N^+(x_3) = N_W(x_3) = \{x_6, x_7\}, \quad m(x) = x_6$$

et passons à l'itération 3.

- *3.* $m(x) = x_6 \neq x_8$ *; pour* $y \in N^+(x_3) \setminus \{x_6\} = \{x_7\}$, $N_W(y) = N_W(x_6) = \{x_3, x_8\}$*. Ceci implique que le sommet* $x = x_3$ *est bisimplicial. Posons*

$$W = \{x_6, x_8\}, \quad x = x_6 \quad et \quad N_W(x_6) = \{x_8\}, \quad m(x) = x_8$$

et passons à l'itération 4.

- *4.* $m(x) = x_8$, $W = B_{x_6}$ *est une biclique et l'algorithme s'arrête pour conclure que le graphe d'entrée est un graphe biparti distance héréditaire.*

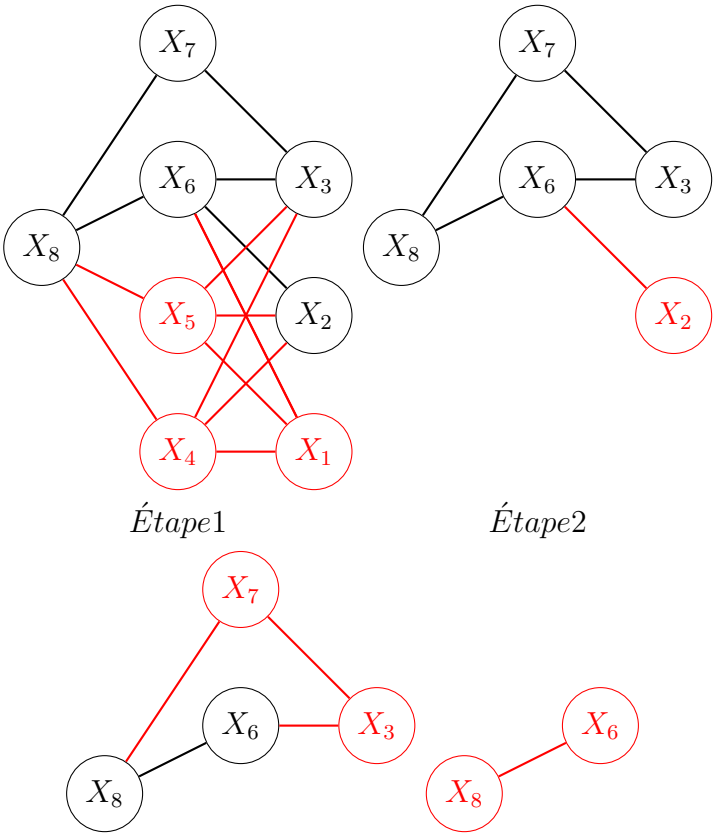


FIGURE 4.8 – LexBFS est un ordre d'élimination bisimplicial

4.2.2 Algorithme d'énumération de bicliques maximales

Dans cette section, nous supposons que $G = (X, Y, E)$ est un graphe biparti distance héréditaire ordonné par LexBFS $\sigma = (v_1, v_2, ..., v_n)$. Rappelons que les sommets de G sont partitionnés en niveaux $N_0, N_1, ..., N_k$ et pour tout i, v_i est un sommet bisimplicial dans le sous-graphe induit G_i, et l'unique biclique maximale dans G_i contenant v_i est noté B_{v_i}.

Remarque 4.2.1. *Notons que :*
- *Pour tout i, le sommet v_i est le plus petit par rapport à σ dans G_i ; plus précisément, $v_i = \min B_{v_i} = \min_\sigma G_i$;*
- *Les voisins supérieurs de tout sommet $v_i, i < n$, dans G Coïncident avec les voisins de v_i dans le sous-graphe G_i, i.e $N_G^+(v_i) = N_{G_i}(v_i)$;*
- *Pour tout $i < n$, $B_{v_i} = \Big(N_G^+(v_i), N_{G_i}(v_j)\Big), i < j$ où $N_{G_i}(v_j)$ représente les voisins de v_j dans G_i.*

Remarque 4.2.2. *On peut voire facilement que si u, v sont deux sommets tels que $d(u, v) > 2$, alors u et v ne peuvent pas appartenir à une même biclique.*

Theorem 4.2.10. *Soit v_i un sommet tel que $v_n \notin N^+(v_i)$. Alors les deux conditions suivantes sont équivalentes*

1. *La biclique $B_{v_i} = \Big(N^+(v_i), N_{G_i}(v_j)\Big), j > i$, est maximale dans G ;*
2. *$\forall t < i, N^+(v_t) \neq N^+(v_i)$.*

Démonstration. S'il existe $t < i$ tel que $N^+(v_t) = N^+(v_i)$, alors on a clairement $B = (N^+(v_i), N_{G_i}(v_j) \cup \{v_t\})$ est une biclique contenant strictement la biclique B_{v_i}, et ceci implique que B_{v_i} n'est pas maximale dans G.

Inversement, notons que puisque $v_n \notin N^+(v_i)$, alors v_i se trouve dans un niveau $N_l, l \geq 2$ et donc $v_j \in N_{l-1}$. De plus, v_j possède au moins un voisin supérieur $v_s \in N_{l-2}$. Notons aussi que les sommets v_i, v_j et v_s sont tous dans B_{v_i}.

Soit $t < i$ tel que $N^+(v_t) \neq N^+(v_i)$. Puisque $t < i$, alors $v_t <_\sigma v_i$ et donc v_t appartient soit à N_l soit à N_r avec $r > l$.
- Si $v_t \in N_r, r > l$, dans ce cas $d(v_t, v_s) \geq 3$, et donc, d'après la remarque 4.2.2, $\{v_t\} \cup B_{v_i}$ n'est pas une biclique ;
- Si $v_t \in N_l$, dans ce cas et puisque $v_t <_\sigma v_i$ et $N^+(v_t) \neq N^+(v_i)$, alors d'après la remarque ci-dessous, $N^+(v_i) - N^+(v_t) \neq \emptyset$, c'est-à-dire, v_t

n'est pas adjacent à au moins un sommet de $N^+(v_i)$ qui constitue une partie de B_{v_i}, d'où $\{v_t\} \cup B_{v_i}$ n'est pas une biclique.

Remarque 4.2.3. *L'ordre σ est un LexBFS si et seulement si pour toute paire de sommets v_i, v_j avec $v_i <_\sigma v_j$ l'une des conditions suivantes est satisfaite*

- $N^+(v_i) = N^+(v_j)$;
- $N^+(v_i) \subset N^+(v_j)$;
- $max\Big(N^+(v_i) - N^+(v_j)\Big) <_\sigma max\Big(N^+(v_j) - N^+(v_i)\Big)$

Par conséquent, la biclique B_{v_i} est maximale dans G. □

Theorem 4.2.11. *Soit v_i un sommet tel que $v_n \in N^+(v_i)$. Alors les deux conditions suivantes sont équivalentes*

1. *B_{v_i} maximale dans G ;*
2. *$B_{v_i} = (N_1, N_0)$ et $N^+(v_{i-1}) \neq N_1 = N^-(v_n)$*

Démonstration. Soit B_{v_i} une biclique maximale. Puisque $v_n \in N^+(v_i)$, alors $v_i \in N_1$, donc B_{v_i} est incluse dans la biclique (N_0, N_1) ; et puisque B_{v_i} maximale, alors $B_{v_i} = (N_0, N_1)$. Ainsi, on a clairement $v_i = \min N_1$; on a aussi $N^+(v_{i-1}) \neq N_1 = N(v_n)$, car dans le cas contraire, on aurait $B_{v_i} \subset B_{v_{i-1}}$.

Réciproquement, si $B_{v_i} = (N_1, N_0)$, alors $v_i = \min N_1$. Par ailleurs, soit $j < i$. On a deux cas

- Cas 1 : si $v_j \in N_l$, avec $l > 2$, alors d'après 4.2.2, les sommets v_j et v_n ne peuvent pas appartenir à la même biclique.
- Cas 2 : si $v_j \in N_2$, et puisque $N^+(v_{i-1}) \neq N_1$, alors d'après observation 2, $N^+(v_j) \neq N_1$ et ceci implique que v_j n'est pas adjacent à au moins un sommet dans le niveau N_1 qui constitue une partie de B_{v_i}, et ceci implique $\{v_j\} \cup B_{v_i}$ n'est pas une biclique. Par conséquent, B_{v_i} est maximale dans G.

□

Le theorem 4.2.10 et le theorem 4.2.11 aboutissent à l'algorithme ci-dessous qui énumère toutes les bicliques maximales d'un graphe biparti distance héréditaire ordonné par LexBFS avec une complexité linéaire. Dans l'algorithme ci-dessous, on a initialisé le nombre de bicliques maximales $\theta = 1$, car B_{v_1} est maximale pour tout graphe biparti distance héréditaire.

Algorithm 7 [33]Énumération de toutes les bicliques maximales

1: **Entrée :** Un graphe biparti à distance héréditaire $G = (V, E)$ ordonné par LexBFS $\sigma = \{v_1, v_2, \ldots, v_n\}$
2: **Sortie :** θ, le nombre de bicliques maximales dans G
3: $i \leftarrow 1$
4: $\theta \leftarrow 1$
5: **tant que** $v_n \notin N^+(v_{i+1})$ **faire**
6: **si** $N^+(v_{i+1}) = N^+(v_i)$ **alors**
7: $i \leftarrow i + 1$
8: **sinon**
9: $i \leftarrow i + 1$
10: $\theta \leftarrow \theta + 1$
11: **fin si**
12: **fin tant que**
13: **si** $v_n \in N^+(v_{i+1})$ **alors**
14: **si** $|N^+(v_i)| \neq |N^-(v_n)|$ **alors**
15: $\theta \leftarrow \theta + 1$
16: **fin si**
17: **fin si**
18: **Retourner** θ

Theorem 4.2.12. *L'algorithme 7 énumère toutes les bicliques maximale de G en $O(m)$.*

Démonstration.

1. **Correction**
   Selon le theorem 4.2.10, dans la ligne 6, θ est incrémenté si et seulement si une nouvelle biclique maximale est détectée. Selon le theorem 4.2.11, θ est incrémenté dans la ligne 11 si et seulement si la biclique $B = (N_1, N_0)$ est maximale.
2. **Complexité**
   Il est claire que la complexité de cet algorithme est dominée par celle de la ligne 3, où pour chaque sommet v_i, on calcule $N^+(v_i)$ au plus une fois. Par conséquent, à la fin de l'algorithme, le nombre d'opérations effectuées est en $O(\sum_1^n |N^+(v_i)|) = O(m)$.

□

Remarque 4.2.4. *Pour calculer une biclique sommet-maximum ou une biclique arête-maximum avec une complexité linéaire dans un graphe biparti distance héréditaire, il suffit de modifier légèrement l'algorithme 7 [33].*

Exemple 4.2.13. *Montrons sur la figure 4.9 comment appliquer l'algorithme 7 pour énumérer toutes les bicliques maximales de G*

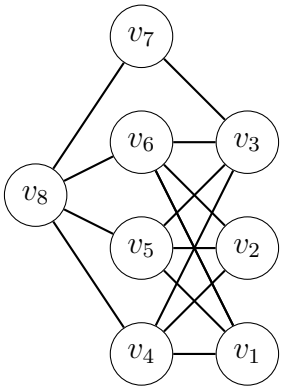


FIGURE 4.9 – Énumération des Bicliques maximales.

1. $i = 1$, $\theta = 1$, $B_{v_1} = (\{v_1, v_2, v_3, v_8\}, \{v_4, v_5, v_6\})$;
2. $i = 2$ $v_8 \notin N^+(v_2)$, $N^+(v_1) = \{v_4, v_5, v_6\} = N^+(v_2)$,// *puisque il y a égalité, on incrémente i seulement.* $\Rightarrow i \leftarrow 3$.
3. $i = 3$, $v_8 \notin N^+(v_3)$, $N^+(v_2) = \{v_4, v_5, v_6\} \neq N^+(v_3) = \{v_4, v_5, v_6, v_7\}$, // *on incrémente* θ *et* $i \Rightarrow \theta = 2$, $B_{v_3} = (\{v_3, v_8\}, \{v_4, v_5, v_6, v_7\})$, $i \leftarrow 4$.
4. $i = 4$, $v_8 \in N^+(v_4)$, $|N^+(v_3)| = 4 = |N^-(v_8)|$
   Ainsi, ce graphe possède 2 bicliques maximales.

4.2.3 Problème de transversal dans un arbre

Dans cette section, nous appliquons LexBFS pour résoudre efficacement le problème de transversal dans un arbre. Mais avant, nous rappelons quelques propriétés d'un arbre. Rappelons qu'un arbre (tree) est un graphe connexe sans cycle. Notons que tout arbre est un biparti, est un triangulé et est un distance héréditaire. Autrement dit, les arbres constitue une sous-classe de l'intersection de ses trois classes de graphes. Les sommets pendants dans les arbres sont les sommets simpliciaux et bisimpliciaux de celui-ci, et vice versa. Ainsi, un arbre peut être caractérisé par les propositions suivantes :

Proposition 4.2.14. *Un graphe G est un arbre si et seulement si il admet un ordre d'élimination sommet pendant.*

Proposition 4.2.15. *Soit un graphe G ordonné par LexBFS σ. Alors, G est un arbre si et seulement si σ est un ordre d'élimination sommet pendant.*

Ainsi, le parcours LexBFS peut aussi être utilisé pour reconnaitre les arbres en un temps linéaire :

Algorithm 8 Reconnaissance d'arbre

Entrée : Un graphe G ordonné par lexBFS σ.
Sortie : G est un arbre si et seulement σ est un ordre d'élimination sommet pendant.
Début
pour $i = 1...n-1$ faire
si $|N^+(v_i)| \neq 1$ **alors**
    retourner faux
fin si
retourner vrai
Fin

Proposition 4.2.16. *Soient G un arbre, v un sommet pendant et w l'unique voisin de v. Alors il existe un transversal minimum T tel que $w \in T$.*

Démonstration. Soit T un transversal minimum de G. Si $w \notin T$, alors T contient forcément v. Par ailleurs, le sommet v couvre la seule arête vw, donc si nous remplaçons v par w dans T, on trouve un autre transversal ayant la même cardinalité que T. D'où le résultat. □

Proposition 4.2.17. *Soient G un arbre, v un sommet pendant et w l'unique voisin de v. Si G' est le sous-graphe obtenu par la suppression de l'ensemble $N(v)$, alors*

$$\tau(G') = \tau(G) - 1$$

.

Démonstration. Soit T un transversal minimum de G qui contient le sommet W (cet transversal existe d'après la proposition ci-dessus). Soit $T' = T - \{w\}$. Il est claire que l'ensemble T' couvre toutes les arêtes de G excepté celles couvertes par le sommet w, donc T' est un transversal dans G'. Par l'absurde, supposons que T' n'est pas minimum, dans ce cas, il existe un autre transversal T" tel que $|T"| < |T'|$. Mais ceci implique que $T" \cup \{w\}$ est un transversal dans G avec une cardinalité strictement inférieure à celle de T, or ceci est impossible car T est minimum. □

Lamme 4.2.18. *[33] Soit $G = (V, E)$ un graphe biparti distance héréditaire connexe et ordonné par LexBFS $\sigma = (v_1, v_2, \ldots, v_n)$. Alors, la restriction de σ sur le sous-graphe G' obtenu par la suppression de v_1 ou par la suppression de v_1 et ses voisins, est aussi un LexBFS.*

Corollaire 4.2.19. *Si v est un sommet pendant (donc bisimplicial) dans un arbre G ordonné par LexBFS σ, alors la restriction de σ sur le sous-graphe obtenu par la suppression de $N[v]$ est un LexBFS.*

Algorithm 9 Calcul la taille d'un transversal minimum pour un arbre

Entrée : Un arbre $G = (V, E)$ ordonné par lexBFS $\sigma = (v_1, v_2, ..., v_n)$.
Sortie : $\tau(G)$, la taille d'un transversal minimum de G.
Début
$v = v_1$; $F = V$, $\tau = 0$.
tant que $F \neq \emptyset$ **faire** ;
    $v = \min F$
    si v est isolé **alors**
        $F = F - \{v\}$;
    sinon
        $F = F - N[v]$;
        $\tau = \tau + 1$
    fin si

Theorem 4.2.20. *L'algorithme ci-dessus calcul le nombre de domination $\tau(G)$ en un temps linéaire pour un arbre.*

Démonstration. **Correction**
D'après les propositions citées ci-dessus, l'algorithme 9 calcul un transversal minimum.
Complexité
Remarquons qu'à chaque itération de l'algorithme, on supprime des sommets, et chaque sommet est supprimé une seule fois, par conséquent, la complexité de l'algorithme est $O(n)$. □

Exemple 4.2.21. *Appliquons l'algorithme 9 au graphe ci-dessous*

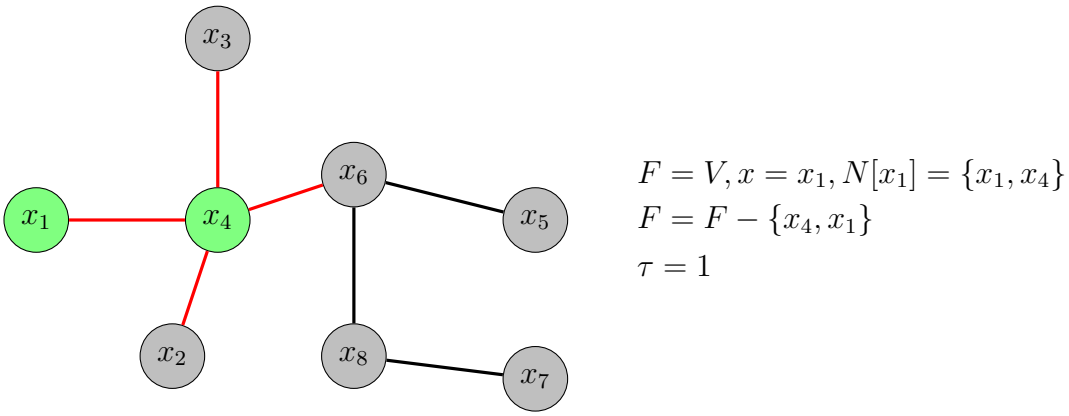


FIGURE 4.10 – Transversal minimum pour un arbre

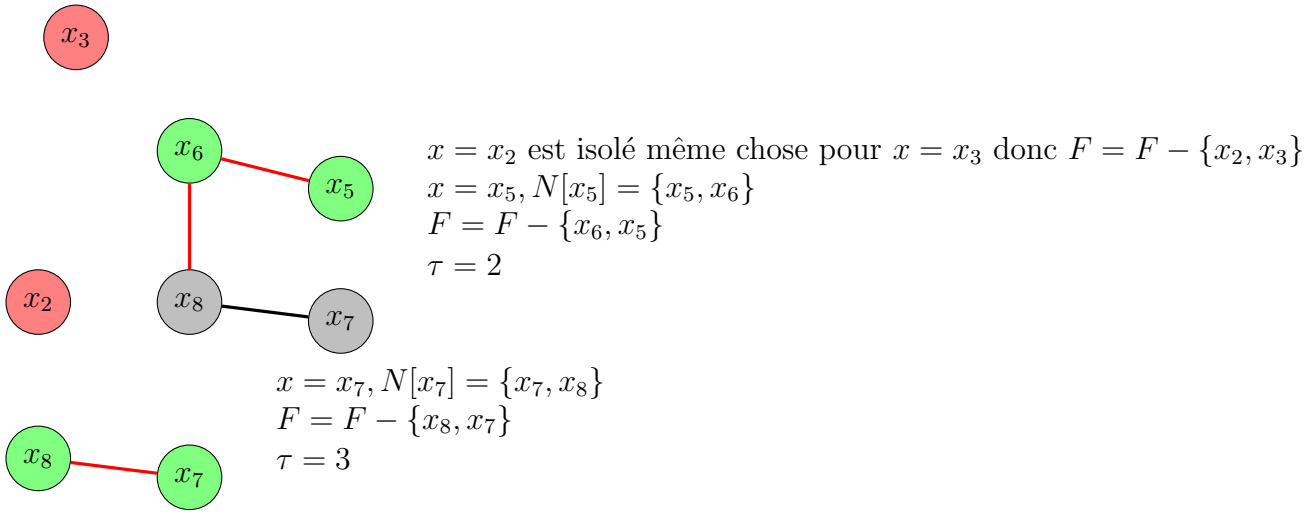


FIGURE 4.11 – Transversal minimum pour un arbre

Conclusion du chapitre 4

Dans ce chapitre, nous avons étudié les graphes bipartis distances héréditaires et proposé des algorithmes efficaces pour leur reconnaissance, l'énumération des bicliques maximales et transversal minimum dans un arbre. En utilisant le LexBFS, nous avons pu identifier les (GBDH) et résoudre des problèmes complexes de manière optimale. Ces résultats démontrent que les graphes GBDH offrent des solutions intéressantes pour des problèmes NP-complets, tout en ouvrant la voie à de futures recherches dans ce domaine.

CONCLUSION GÉNÉRALE

Dans ce mémoire, nous avons étudié le calcul d'invariants dans les graphes distances héréditaires, en particulier dans le contexte des problèmes NP-complets. Nous avons proposé des algorithmes efficaces pour la reconnaissance de ces graphes, ainsi que pour l'énumération de bicliques maximales, en utilisant des techniques telles que le parcours LexBFS.

Les résultats obtenus montrent que, bien que certains problèmes associés aux graphes soient difficiles à résoudre de manière générale, il est possible de concevoir des solutions efficaces pour des classes spécifiques, comme les graphes distances héréditaires. Nos contributions ont notamment permis de résoudre des problèmes complexes comme la couverture de sommets et d'arêtes dans les graphes bipartis, tout en apportant de nouvelles perspectives dans le calcul d'invariants pour des structures plus générales.

Ce travail met en évidence le rôle fondamental des invariants dans la théorie des graphes et leur importance pour l'optimisation algorithmique. Les méthodes proposées pourraient être étendues à d'autres classes de graphes, offrant ainsi de nombreuses pistes pour des recherches futures. En conclusion, les résultats obtenus renforcent la pertinence du calcul d'invariants dans des domaines allant de l'optimisation à la modélisation de structures complexes.

BIBLIOGRAPHIE

[1] *Kenneth I Appel and Wolfgang Haken.* Every planar map is four colorable*, volume 98. American Mathematical Soc., 1989.*

[2] *Sanjeev Arora and Boaz Barak.* Computational complexity : a modern approach*. Cambridge University Press, 2009.*

[3] *Brenda S Baker. Approximation algorithms for np-complete problems on planar graphs.* Journal of the ACM (JACM)*, 41(1) :153–180, 1994.*

[4] *Lounis Belhoucine and Zhour Achour.* Etude synthèse et génération des cliques maximales dans les graphes non orientés*. PhD thesis, UMMTO, 2017.*

[5] *Jean RS Blair and Barry Peyton. An introduction to chordal graphs and clique trees. In* Graph theory and sparse matrix computation*, pages 1–29. Springer, 1993.*

[6] *JA Bondy and USR Murty. Théorie des graphes, 2008.*

[7] *Gregory J Chaitin, Marc A Auslander, Ashok K Chandra, John Cocke, Martin E Hopkins, and Peter W Markstein. Register allocation via coloring.* Computer languages*, 6(1) :47–57, 1981.*

[8] *Olivier Cogis and Eric Thierry. Computing maximum stable sets for distance-hereditary graphs.* Discrete Optimization*, 2(2) :185–188, 2005.*

[9] *Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein.* Introduction to algorithms*. MIT press, 2022.*

[10] *Guillaume Damiand.* Quelques propriétés des graphes distances héréditaires*. PhD thesis, Master's thesis, Université de Montpellier II, LIRMM, 1997.*

[11] Guillaume Damiand, Michel Habib, and Christophe Paul. A simple paradigm for graph recognition : application to cographs and distance hereditary graphs. Theoretical Computer Science*, 263(1-2) :99–111, 2001.*

[12] Rolf Niedermeier Professeur Docteur and Peter Rossmanith Professeur Docteur. Algorithmes exacts et exponentiels pour les problèmes np-difficiles : domination, variantes et généralisations.

[13] Alessandro D'Atri and Marina Moscarini. Distance-hereditary graphs, steiner trees, and connected domination. SIAM Journal on Computing*, 17(3) :521–538, 1988.*

[14] David Easley, Jon Kleinberg, et al. Networks, crowds, and markets : Reasoning about a highly connected world*, volume 1. Cambridge university press Cambridge, 2010.*

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

[16] Alfred Errera. Une contribution au problème des quatre couleurs. Bulletin de la Société Mathématique de France*, 53 :42–55, 1925.*

[17] Louis Fippo Fitime. Modélisation hybride, analyse et vérification quantitative des grands réseaux de régulation biologique*. PhD thesis, École centrale de Nantes, 2016.*

[18] Claude Flament. Théorie des graphes et structures sociales*. Mathématiques et sciences de l'homme, 2. Mouton, 1965.*

[19] Jean-Claude Fournier. Théorie des graphes et applications*. Informatique. hermes, 2011.*

[20] Michael R Garey and David S Johnson. The complexity of near-optimal graph coloring. Journal of the ACM (JACM)*, 23(1) :43–49, 1976.*

[21] Michael R Garey, David S Johnson, and Larry Stockmeyer. Some simplified np-complete problems. In Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63, 1974.*

[22] Sudipto Guha and Samir Khuller. Approximation algorithms for connected dominating sets. Algorithmica*, 20 :374–387, 1998.*

[23] William K Hale. Frequency assignment : Theory and applications. Proceedings of the IEEE*, 68(12) :1497–1514, 1980.*

[24] Alain Hertz and D de Werra. Using tabu search techniques for graph coloring. Computing*, 39(4) :345–351, 1987.*

[25] Haruo Hosoya. Topological index. a newly proposed quantity characterizing the topological nature of structural isomers of saturated hydrocarbons. Bulletin of the Chemical Society of Japan*, 44(9) :2332–2339, 1971.*

[26] Edward Howorka. A characterization of distance-hereditary graphs. The quarterly journal of mathematics, *28(4) :417–420, 1977.*

[27] Richard M Karp. Reducibility among combinatorial problems. *Springer, 2010.*

[28] Clément Lecat. Réduction de l'espace de recherche du Problème de la Somme Coloration Minimum d'un graphe. *PhD thesis, Université de Picardie Jules Verne, 2017.*

[29] Jean Mayer. Le théorème des quatre couleurs : notice historique et aperçu technique. Cahiers du séminaire d'histoire des mathématiques, *3 :43–62, 1982.*

[30] Gregory Morel. Stabilité et coloration des graphes sans P5. *PhD thesis, Université de Grenoble, 2011.*

[31] Owen J Murphy. Computing independent sets in graphs with large girth. Discrete Applied Mathematics, *35(2) :167–170, 1992.*

[32] Donald J Rose, R Endre Tarjan, and George S Lueker. Algorithmic aspects of vertex elimination on graphs. SIAM Journal on computing, *5(2) :266–283, 1976.*

[33] Djamel Talem. Calcul d'invariants dans les graphes et ordres. *PhD thesis, Universite Mouloud MAMMERI Tizi-Ouzou, 2023.*

[34] Jacobo Valdes, Robert E Tarjan, and Eugene L Lawler. The recognition of series parallel digraphs. In Proceedings of the eleventh annual ACM symposium on Theory of computing, *pages 1–12, 1979.*

[35] Feng Wang, Hongwei Du, Erika Camacho, Kuai Xu, Wonjun Lee, Yan Shi, and Shan Shan. On positive influence dominating sets in social networks. Theoretical Computer Science, *412(3) :265–269, 2011.*

[36] Jie Wu and Hailan Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications, *pages 7–14, 1999.*

Résumé / Abstract

Résumé :
Ce mémoire traite du calcul d'invariants dans les graphes distances héréditaires, un sujet central en théorie des graphes. Après une présentation des notions de base et des parcours dans les graphes (comme le BFS, DFS et LexBFS), l'étude se concentre sur les graphes distances héréditaires (GDH). Ces graphes permettent de résoudre plus efficacement certains problèmes NP-complets, en proposant des algorithmes pour leur reconnaissance, l'énumération des bicliques maximales dans les graphes biparti distances héréditaires (GBDH), et le calcul de transversal minimum dans un arbre. Le travail examine également la complexité algorithmique, notamment les classes P et NP, et explore les liens entre invariants de graphes et théorie de la complexité.

Mots clés : *Calcul d'invariants, graphes distances héréditaires, biclique, décomposition de graphe en bicliques...*

Abstract :
This thesis focuses on the computation of invariants in distance-hereditary graphs, a central topic in graph theory. After introducing the basic concepts and graph traversals (such as BFS, DFS, and LexBFS), the study focuses on distance-hereditary graphs (DHG). These graphs allow for more efficient solutions to certain NP-complete problems by providing algorithms for their recognition, the enumeration of maximal bicliques in distance-hereditary bipartite graphs (DHBG), and the computation of the minimum transversal in a tree. The work also examines algorithmic complexity, notably the P and NP classes, and explores the connections between graph invariants and complexity theory.

Keywords : *Invariant computation, distance-hereditary graphs, biclique, biclique decomposition of graphs...*