N°d'ordre: .....

République Algerienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la Recherche scientifique

Université Mouloud Mammeri de Tizi Ouzou
Faculté des Sciences
Département de Mathématiques
Laboratoire LAROMAD

Mémoire de Master

Filière : Mathématiques
Spécialité : Recherche Opérationnelle

Par

Zaouache Narimane

Belhadj ali Ouardia

La résolution des problèmes difficiles de la théorie des graphes

Soutenue le Septembre 2022 devant le jury :

Mr.	AOUANE Mohouhand	UMMTO	Président du jury
Mme.	OUBAKOUK Lynda	UMMTO	Examinatrice
Mr.	TALEM Djamel	UMMTO	Encadreur

Année Universitaire : 2021/2022

Dédicace

Je dédie ce modeste travail à mes chers parents qui m'ont toujours soutenu et épauler pour que je puisse atteindre mes objectifs.

À mes sœurs, mes frères, ma belle sœur et mon beau frère pour leurs soutiens moral et leurs conseils tout au long de mes études.

À mes chers grands parents paternel que dieu les accueille dans son vaste paradis, et à ma grande mère maternelle.

À mon neveu Aksel et mes nièces Thilelli, Yasmine, Sofia et Isabella.

À Yahia Merabet pour son encouragement et l'aide qu'il m'a toujours accordé.

À tous mes ami(e)s, en particulier à Syphax Mammou, et à ma binôme et sa famille.

Narimane

Dédicace

Je dédie ce modeste travail à mes chers parents qui ont toujours été la pour moi à m'encourager afin d'atteindre mes objectifs.

À mes deux frère, à ma sœur et mon beau frère qui m'ont soutenu et encouragé durant ces années d'études.

À mon adorable neveu Mayas et à ma belle nièce Célia.

À toute ma famille et mes ami(e)s, notamment à ma binôme et sa famille.

Ouardia

Remerciements

On remercie dieu le tout puissant de nous avoir donné suffisamment la santé, la volonté et le courage afin de pouvoir accomplir ce modeste travail.

Je tiens à remercier en premier lieu Talem Djamel d'avoir accepter de nous encadrer. Il ne saura jamais assez notre gratitude pour tout ce qu'il nous a apprit en théorie des graphes, mais aussi pour ses précieux conseils, sa disponibilité et surtout sa patience.

Nous remercions également les membres du jury, Mr AOUANE MOHOUHAND et Mme OUBAKOUK LYNDA pour avoir accepté d'évaluer ce travail.

Nous apportons aussi nos vifs remerciements aux personnels enseignants qui, par leur enseignement et leur aide ont contribué à notre formation durant toutes notre étude.

Enfin nous tenons à remercier tout ceux qui nous ont apporté, de prés ou de loin, de l'aide et des conseils pour la réalisation de ce travail.

Tizi-Ouzou, le 24 octobre 2022.

TABLE DES FIGURES

TABLE DES MATIÈRES

INTRODUCTION

La recherche opérationnelle (RO) peut se définir comme la mise en œuvre des méthodes scientifiques, essentiellement mathématiques et statistiques, en vu de prendre la meilleure décision possible. Depuis longtemps, les mathématiques ont connu des problèmes dont l'énoncé était simple à présenter, en revanche leur résolution n'était pas évidente et nécessitait de nouveaux outils pour pouvoir les aborder. C'est pour cela on fait appel à la théorie des graphes. De manière générale, la théorie des graphes peut être considérée comme un outil de modélisation permettant de représenter une structure d'un objet complexe en exprimant les relations entre ses éléments. La programmation linéaire et les principaux algorithmes de graphes sont souvent enseignés dans les parcours mathématiques ou informatiques des universités et grandes écoles. Ils sont souvent complétés par d'autres notions théoriques comme la théorie de la complexité qui permet d'avoir une meilleure compréhension des problèmes qui peuvent être résolus efficacement.

L'histoire des graphes est apparue au milieu du XXe siècle, le célèbre physicien hongrois Eugène Wigner parle de « La déraisonnable efficacité des mathématiques dans les sciences de la nature. » En effet, la modélisation mathématique facilite la compréhension d'un problème car elle détermine un seul vocabulaire formel pour différentes situations. La modélisation mathématique peut atteindre un niveau d'abstraction permettant le développement d'une théorie précise sur le modèle, indépendante de la réalité, tout en gardant de nombreuses applications réelles. De manière générale, un graphe permet de représenter les connexions d'un ensemble complexe en exprimant les relations entre ses éléments : réseau de communication, réseaux routiers ...

Dans ce mémoire, nous allons aborder des méthodes de résolution des problèmes difficiles ; nous avons à ce titre déjà mentionné la théorie de la complexité, cette dernière définit la classe des problèmes NP-difficiles à laquelle appartiennent la plupart des problèmes pratiques de la Recherche Opérationnelle. Comme il semble peu probable de pouvoir résoudre efficacement ces problèmes de manière exacte, les informaticiens ont développé un grand nombre de méthodes permettant d'obtenir, en des temps de calcul raisonnables, de bonnes solutions à ces problèmes. Beaucoup de ces méthodes sont génériques même si, dans la plupart des cas, un travail conséquent est nécessaire pour mettre en œuvre de manière performante une telle méthode sur un problème donné.

CHAPITRE 1

DÉFINITIONS ET NOTATIONS

Dans ce premier chapitre nous allons présenter quelques notions de base sur la théorie des graphes qui seront utiles tout au long de ce mémoire.

1.1 Graphe et digraphe

1.1.1 Graphe

Un graphe fini qui n'est pas vide $G = (V, E)$ est un couple constitué par un ensemble fini $V = \{v_1, v_2, \ldots ...v_n\}$ dont les éléments sont des **sommets** et d'un ensemble $E = \{e_1, e_2, \ldots .., e_m\}$ dont les éléments sont des **arêtes**. Une arête e est définie par une paire de sommets $\{x, y\}$ non ordonnés, appelés les extrémités de e. On écrit alors $e = xy$ et on dit que e est **incidente** à x et y.

1.1.2 Digraphe

En donnant une direction aux arêtes d'un graphe, on obtient un **digraphe** (ou graphe orienté). Un digraphe fini $G = (V, U)$ est défini par l'ensemble fini $V = \{v_1, v_2, ..., v_n\}$ dont les éléments sont appelés sommets, et par l'ensemble fini $E = \{e_1, e_2, ..., e_m\}$ dont les éléments sont appelés **arcs**. Un arc e de l'ensemble U est défini par une paire ordonnée de sommets (x, y). On écrit, $e = (x, y)$, et on dit que l'arc e va de x vers y. On dit aussi que x est l'extrémité initiale de l'arc e, et y son extrémité finale.
On dit qu'un graphe G est **simple** si entre deux sommets il existe au plus une arête et il est sans boucle. une **boucle** est une arête dont les extrémités sont confondues.

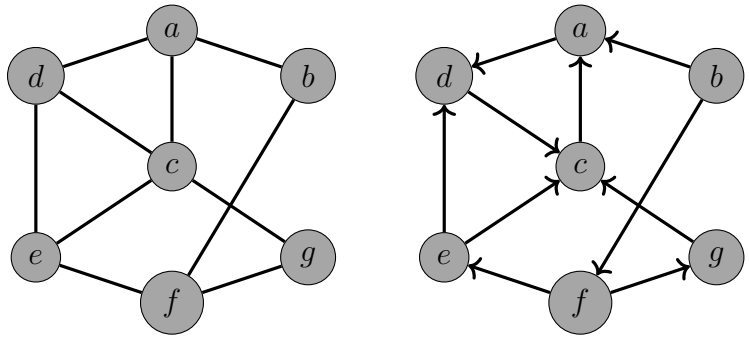


FIGURE 1.1 – Graphe et digraphe

1.1.3 Adjacence, voisinage, degré

Soit $G = (V, E)$ un graphe.

1. Deux sommets u et v sont **adjacents** ou **voisins** si $uv \in E$.
2. Deux arêtes sont adjacentes si elles ont au moins une extrémité commune.
3. Le **voisinage** d'un sommet v dans G est l'ensemble des sommets adjacents à v, il est noté $N_G(v) = \{u \in V, uv \in E\}$.
4. Le **degré** d'un sommets v dans un graphe simple G est le nombre de ses voisins. Il est noté $d_G(v) = |N(v)|$. Autrement dit le degré d'un sommet v est le nombre d'arêtes incidentes à ce sommet.

1.2 Sous-graphe

Soit $G = (V(G), E(G))$ un graphe. On appelle **sous-graphe** de G tout graphe $H = (V(H), E(H))$ tels que $V(H) \subseteq V(G)$ et $E(H) \subseteq E(G)$. Un sous-graphe H est dit **graphe partiel** de G, si $V(H) = V(G)$ et $E(H) \subseteq E(G)$. Autrement dit, le graphe partiel est obtenu par la suppression des arêtes sans toucher aux sommets.
Un sous-graphe H est un sous-graphe **induit** par l'ensemble de sommets A, si $V(H) = A$ et $E(H)$ est l'ensemble de toutes les arêtes de G ayant leurs extrémités dans A. Autrement dit, le sous graphe induit par l'ensemble A est obtenu par la suppression des sommets de $V - A$.

1.3 Graphe connexe et fortement connexe

Une **chaine** allant d'un sommet x vers un sommet y est une suite finie de n sommets $(x_0, x_1; \ldots, x_n)$ tels que $x_0 = x$ et $x_n = y$ et pour tout i dans

$\{0, 1, \ldots, n-1\}$, x_i et x_{i+1} sont adjacents. Un **chemin** est une séquence de sommets $(x_1, x_2, ...x_n)$ telle que (x_i, x_{i+1}) est un arc. Une chaine (resp. chemin) est dite **élémentaire** si elle ne rencontre pas le même sommets deux fois. Une chaine élémentaire passant par tous les sommets de G est dite **hamiltonienne**. Un **cycle** (resp. **circuit**) est une chaine (resp. chemin) ayant l'extrémité initiale confondue avec l'extrémité terminale.

On dit qu'un graphe G est **connexe**, s'il existe une chaine entre toute paire de sommets. Si G n'est pas connexe, il contient au moins, deux sous-graphes connexes, maximaux au sens de l'inclusion, appelés **Composantes connexes**. Un graphe orienté $G = (V, E)$ est dit **fortement connexe**, si quels que soient les sommets u et v de V, il existe un chemin orienté allant de u à v et un autre chemin allant de v à u. C'est-à-dire il existe un circuit passant par les deux sommets. Un graphe qui n'est pas fortement connexe est constitué par des composantes fortement connexes.

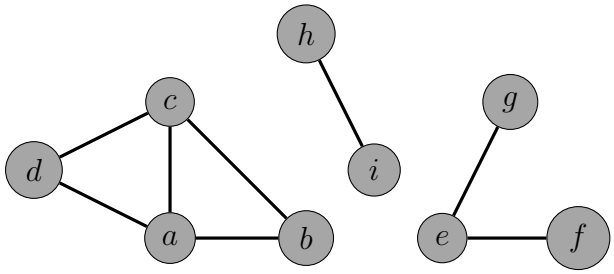
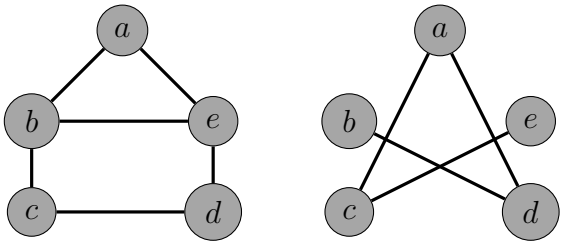


FIGURE 1.2 – Graphe non connexe avec trois composantes connexes : $C_1 = \{a, b, c, d\}, C_2 = \{e, f, g\}$ et $C_3 = \{i, h\}$.

1.4 Graphe complémentaire

Soit $G = (V, E)$ un graphe. Le graphe G est **complet** si les sommets de G sont deux à deux adjacents. Un **stable** de G est un ensemble de sommets deux à deux non adjacents. Une **clique** est un sous-graphe induit complet. Le graphe **complémentaire** d'un graphe $G = (V, E)$, est le graphe $G^c = (V, E^C)$, avec $E^c = \{uv \text{ tel que } uv \notin E\}$, c'est à dire E^c est l'ensemble des arêtes qui manque à G pour être complet.



G G^c

FIGURE 1.3 – Un graphe G et son complémentaire G^c

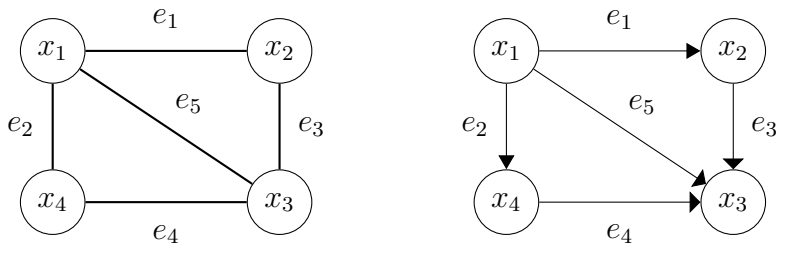
1.5 Représentation d'un graphe

Les graphes peuvent être considérés comme une structure de données. C'est pourquoi, il est fondamental de s'intéresser à comment les présenter. Plusieurs modes de représentation peuvent être envisagées selon la nature des traitements que l'on souhaite appliquer au graphe considéré.

1. **Représentation par matrice d'adjacences :** La matrice d'adjacence d'un graphe simple $G = (V, E)$ d'ordre n est la matrice $M = (m_{ij})_{1 \leq i,j \leq n}$ de dimension $(n \times n)$ (une ligne pour chaque sommet et une colonne pour chaque sommet) telle que :

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

Exemple 1.1. Soient le graphe non orienté G et le graphe H obtenu par l'orientation de G. Ces deux graphes ont la même matrice d'adjacence. La matrice d'adjacence des deux graphes est la suivante :



G H

FIGURE 1.4

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

2. **Représentation par la matrice d'incidence :** Soit $G = (V, E)$ un graphe simple, et la matrice $B = (n \times m)$, une matrice ayant n ligne et m colonne, c'est à dire une ligne pour chaque sommet et une colonne pour chaque arête telle que :
   (a) pour un graphe orienté, $b_{ij} = 1$ si le sommet i est l'extrémité initiale de l'arc j, $b_{ij} = -1$ si i et l'extrémité terminale de l'arc j et $a_{ij} = 0$ sinon.
   (b) pour un graphe non orienté, $b_{ij} = 1$ si le sommet i est l'extrémité de l'arête j, $b_{ij} = 0$ sinon.

   Pour un graphe orienté, la matrice d'incidence est définie :

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

   Pour un graphe non orienté, la matrice d'incidence est définie par :

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

   Exemple 1.2. La matrice d'incidence associée au graphe orienté H de la figure 1.4 est la suivante :

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

   La matrice d'incidence associée au graphe non orienté G de la figure 1.4 est la suivante :

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

1.6 Exploration d'un graphe

Pour déterminer l'ensemble des sommets accessibles, il est nécessaire de parcourir le graphe de façon systématique. Principalement, nous utilisons deux types de parcours, le **parcours en largeur** (BFS) et le **parcours en profondeur** (DFS). Soit $G = (X, E)$ un graphe non orientée. L'objectif de ces deux parcours est d'explorer le graphe G à partir d'un sommet de départ s, appelée racine ou source, et puis, parcourir un à un tous les sommets du graphe accessibles à partir de s par un chemin. Les arêtes de G utilisés pour faire le parcours forment une arborescence des plus cours chemin allant de la source s et les sommets accessibles à partir de celui-ci.

1.6.1 Parcours en largeur BFS

L'algorithme de parcours en largeur (BFS) permet de parcourir un graphe de la manière suivante : on commence par explorer un sommet source, puis ses successeurs, puis les successeurs non explorés des successeurs et ainsi de suite. L'algorithme de parcours en largeur utilise la structure de données **file** qui utilise le principe, les premiers arrivés sont les premiers sortir (FIFO). Les étapes d'un parcours en largeur sont résumés comme suite :

1. Mettre le sommet source dans la file.
2. Retirer le sommet de la tête de la file pour l'examiner (i.e explorer ses voisins non explorés) ;
3. Mettre tous ses voisins explorés pour la première fois à la fin de la file ;
4. Si la file n'est pas vide, aller en 2.

Pour bien comprendre le principe de la recherche en largeur, on utilise des couleurs au cours de la recherche. Dans notre cas un sommet est coloré en jaune s'il n'est pas encore exploré, un sommet est en vert s'il est exploré (donc il figure dans la file), et un sommet est rouge si tous ses voisins sont explorés.

Exemple 1.3. On montre le déroulement du parcours en largeur BFS à partir du sommet 1

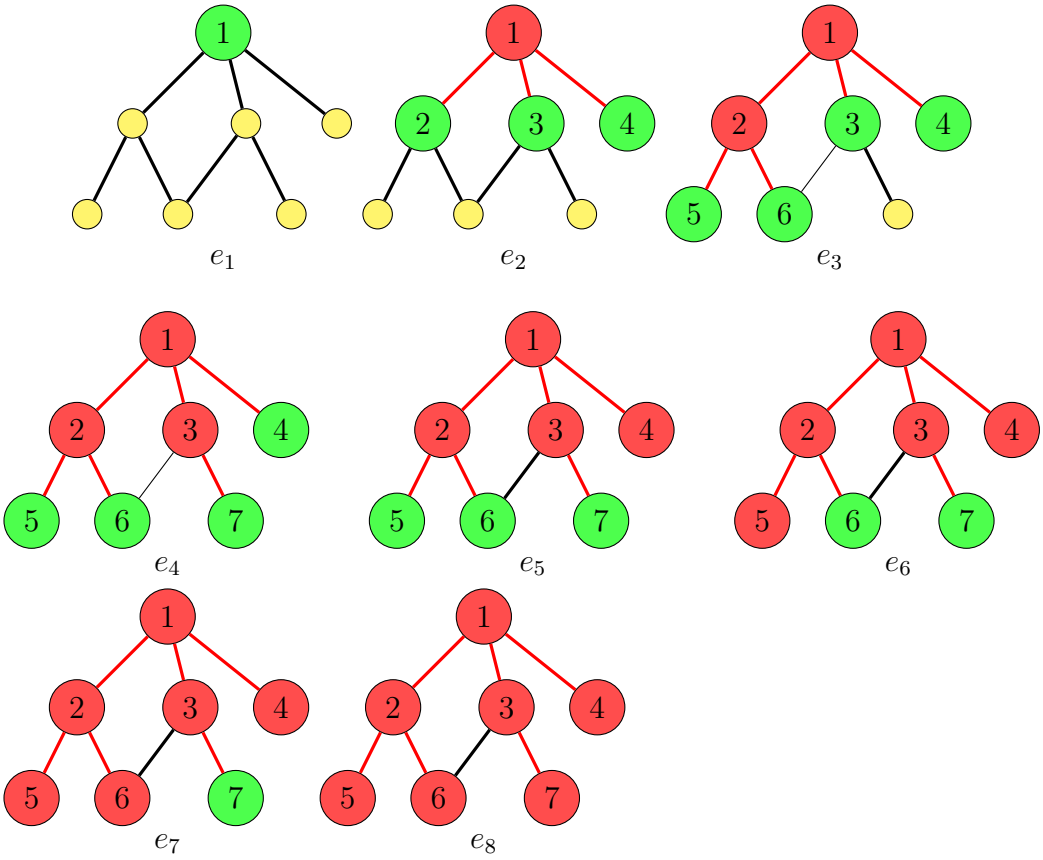


FIGURE 1.5 – Illustration d'un parcours en largeur

1.6.2 Parcours en profondeur DFS

L'algorithme de parcours en profondeur parcoure un graphe à partir du sommet source s. On descend plus profondément dans le graphe à chaque fois que ceci est possible, à partir de s, on explore un voisin y de s, puis on explore un voisin non exploré de ce voisin, et si ce dernier a un voisin non exploré, on l'explore, ainsi de suite. On avance en profondeur pour chercher des files à explorer. Ce parcours utilise une pile : c'est une structure de donnée qui se base sur le principe "dernier marqué, premier sorti (LIFO)". Les étapes d'un parcours en profondeur sont résumés comme suite :

1. On marque le sommet de départ s comme exploré, puis on choisit un sommet voisin quelconque ;
2. Si le sommet voisin n'a pas déjà été visité, on le marque comme exploré ;
3. Si ce dernier n'est pas fermé, c'est-à-dire, qu'il existe au moins un

voisin qui n'est pas encore visité, on l'explore, et on relance à nouveau ce processus à chaque fois qu'on trouve un sommet non fermé jusqu'à ce que tout les sommets seront explorés.

Pour voir clairement comment fonctionne cet algorithme, on utilise trois couleurs : tous les sommets sont blanc au départ. dés qu'un sommet est exploré, il sera mit dans la pile et sa couleur devient grise, et dés qu'on tombe sur un sommet fermé, ce dernier quitte la pille et sa couleur passe de grise à verte.

Exemple 1.4. On montre comment dérouler le parcours en profondeur DFS à partir du sommet e.

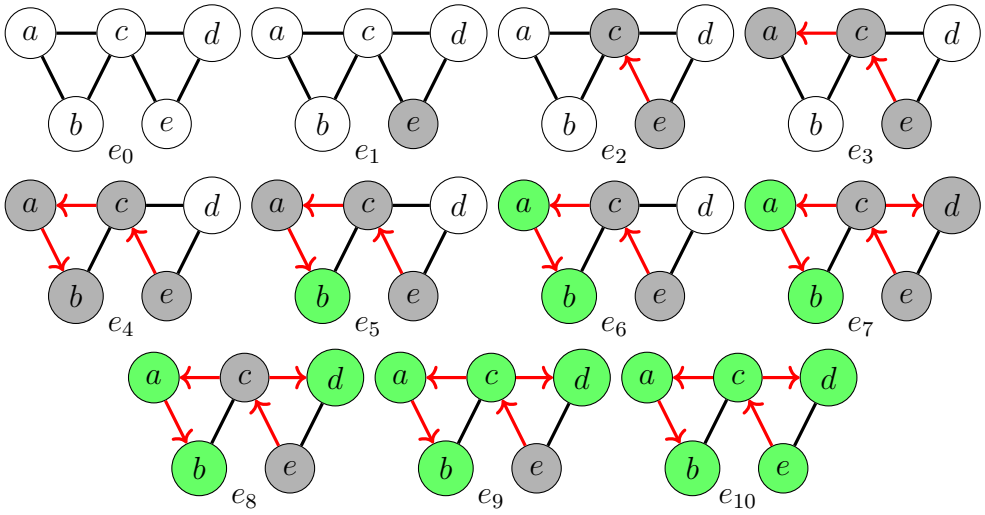


FIGURE 1.6 – Illustration d'un parcours en profondeur

1.6.3 Parcours en largeur lexicographique

C'est un raffinement de l'algorithme de parcours en largeur BFS, ce parcours est très utile pour étudier certaines classes de graphes et pour obtenir des algorithmes de reconnaissance rapides de ces classes, comme la reconnaissance des graphes triangulés.

Soit $G = (V, E)$ un graphe avec $|V| = n$. On associe à chaque sommet v une étiquette (ou un vecteur) $L(v)$ qui est vide au départ. Une fois qu'elles sont pas vides, les étiquettes sont ordonnées par un ordre décroissant, c'est à dire si $L = (a_1, a_2, ...a_k)$ alors $a_1 \leq a_2 \leq ... \leq a_k$.

L'application de l'algorithme LexBFS sur les sommets de G, permit de

définir une bijection

$$\begin{cases} \sigma : \{1, 2...n\} \to V \\ i \to \sigma(i) = v_i \end{cases}$$

On énumère les sommets de G de n à 1 à partir d'un sommet quelconque $s = \sigma(n) = v_n$, le sommet s est la source et donc il reçoit le numéro n comme suite : à chaque étape si i est le plus grand entier non encore attribué, lexBFS énumère par i un sommet v ayant $L(v)$ maximale lexicographiquement, et pour chaque sommet w appartenant à $N(v)$ et qui n'est pas encore numéroté, on concatène l'entier i a la fin de l'étiquette $L(w)$.

Exemple 1.5. On montre comment dérouler le parcours lexicographique LexBFS

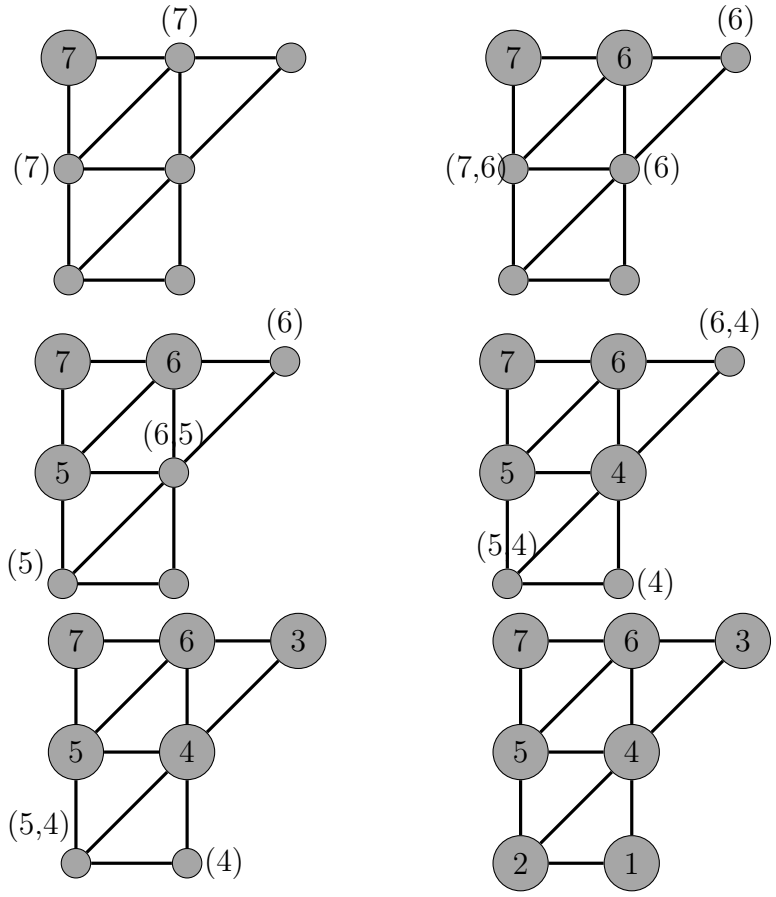


FIGURE 1.7 – Illustration d'un parcours lexicographique

1.7 Classes des graphes

1.7.1 Graphe triangulé

Un graphe est dit triangulé s'il ne contient pas de trou comme sous graphe induit. Un **trou** est un cycle élémentaire de longueur supérieure à 3 sans corde. Une **corde** dans un cycle est une arête joignant deux sommets non consécutifs. Notons que tout sous-graphe induit d'un graphe sans trou est aussi sans trou, donc tout sous-graphe d'un graphe triangulé est triangulé.

Un sommet x est dit *simplicial*, si $N(x)$ induit une clique. Un sommet simplicial appartient à une unique clique maximale. On dit que $\sigma = (x_1, x_2, ..., x_n)$ est un *ordre d'élimination simplicial* si pour $i = 1...n$, x_i est simplicial dans G_i, le graphe induit par les sommets $x_i, x_{i+1}, ..., x_n$.

Pour reconnaitre si un graphe donné G est triangulé, on applique les étapes suivantes :

1. On applique LexBFS pour calculer un ordre σ sur les sommets ;
2. On test si σ est un ordre d'élimination simplicial.

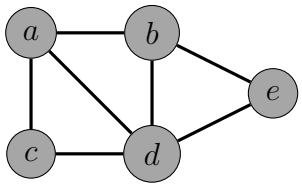


FIGURE 1.8 – Graphe triangulé

1.7.2 Graphe de comparabilité

Un graphe $G = (V, E)$ est de **comparabilité** s'il existe une orientation O transitive et anti-symétrique des arêtes de G, c'est-à-dire, $\forall x, y, z \in V$

$$(x, y) \in O, (y, z) \in O \Rightarrow (x, z) \in O$$

Un graphe de comparabilité $G = (V, E)$ est dit graphe de **co-comparabilité** si son complémentaire G^c est aussi un graphe de comparabilité.

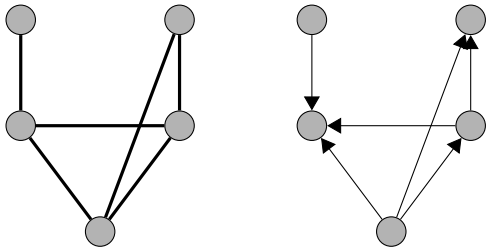


FIGURE 1.9 – Graphe de comparabilité

1.7.3 Graphe d'intervalles

Un graphe d'**intervalles** est le graphe d'intersection d'un ensemble d'intervalle de la droite réelle : chaque sommet représente un intervalle, et une arête relie deux sommets lorsque les intervalles correspondants s'intersectent. Plus précisément, soit $I = \{I_1, I_2, ..., I_n\}$ un ensemble d'intervalles dans $\mathbb{R}$, le graphe d'intervalle correspondant est $G = (V, E)$, où $V = \{I_1, I_2, ...I_n\}$ et

$$I = \{I_x, I_y\} \in E \Longleftrightarrow I_x \cap I_y \neq \varnothing$$

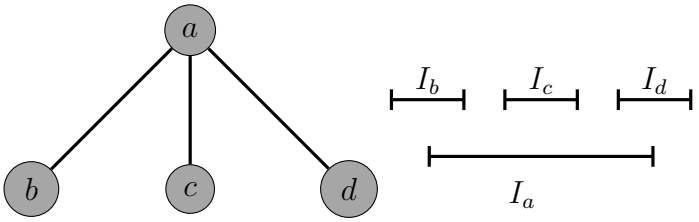


FIGURE 1.10 – Graphe d'intervalles

CHAPITRE 2

NOTIONS SUR LA COMPLEXITÉ ALGORITHMIQUE

Dans ce chapitre, nous allons définir quelques notions sur la complexité ainsi que tous les outils liés à ce concept (problème algorithmique, classes de complexité, problèmes NP-complets,...)

La notion d'algorithme est connu depuis l'antiquité. Autant de dire que les algorithmes sont connus et utilisés bien avant les débuts de l'informatique. Le premier algorithme non trivial est l'algorithme d'*Euclide* (300 avant JC) qui permet de trouver le P.G.C.D de deux entiers. La théorie de la complexité est un domaine théorique des mathématiques et de l'informatique qui ne s'intéresse aux problèmes dont la solution est l'écriture d'un algorithme. Elle se présente comme l'étude de la difficulté intrinsèque de ces problèmes et vise à les classifier selon leurs difficultés. Ici, les mots complexité et difficulté font référence à la quantité de ressources qu'un algorithme utilise pour résoudre un problème : les ressources consistent en le temps d'exécution (la complexité temporelle) et l'espace mémoire requis (sa complexité spatiale). Le principal avantage de la théorie de la complexité est que ces grandeurs sont exprimées indépendamment de tout dispositif physique concret, elles sont indépendantes du matériel (le hardware), du compilateur et du langage (le software) utilisés.

2.1 Concepts de base

2.1.1 Premières définitions

Un problème

Un problème est une question générale qui cherche une réponse et qui possède des paramètres dont la valeur n'est pas connue. Il est composé d'une entrée ou une instance et d'une question ou d'une tache à réaliser dont la sortie est codée par un mot. Une **instance** (certificat) est obtenue en affectant une valeur a chacun de ses paramètres, elle est caractérisée par sa **taille** qui désigne généralement la quantité de cases mémoires nécessaires pour décrire les paramètres.

Exemple 2.1. Le problème du voyageur de commerce (TSP) est défini comme suit : Étant donné un ensemble de villes séparées par des distances connues, on cherche à trouver le plus court chemin qui relie toutes les villes, en ne passant qu'une seule fois par chaque ville.
L'entrée de ce problème est donc un ensemble de n points (représentant les villes). Et la sortie consiste au plus court chemin reliant ces villes.

Dans la littérature il existe deux sortes de problèmes :

1. **Problème de décision** Un problème de décision est une question qui demande une réponse par oui ou non.

   Exemple 2.2. le problème de primalité est un problème de décision, qui consiste à déterminer si un entier naturel est premier.
   Entrée : un entier N ;
   Question : Déterminer si l'entier N est premier.

2. **Problème d'optimisation combinatoire** consiste à déterminer la meilleure solution parmi toutes les solutions réalisables.

   Exemple 2.3. Problème de stable maximum qui consiste à calculer un stable de cardinalité maximum est un problème d'optimisation, il est défini ainsi :
   Entrée : Un graphe $G = (V, E)$;
   Question : Déterminer un stable maximum dans G.

Notons qu'à tout problème d'optimisation correspond un problème de décision.

Exemple 2.4. On a le problème d'optimisation suivant :
Entrée : Un graphe $G = (V, E)$;
Question : Déterminer une clique maximum dans G.

La version décision de ce problème est
Entrée : Un graphe $G = (V, E)$, et un entier k ;
Question : Existe-t-elle une clique ayant au moins k sommets dans G ?

Un algorithme

Un **algorithme** est une suite finie d'opérations de calcul élémentaires permettant de résoudre un problème, il accepte n'importe quelle instance en entrée et renvoie une solution au problème en sortie.
Un algorithme peut se caractériser par le temps et l'espace qu'il utilise pour arriver à la solution. Ces deux notions de temps et d'espace sont des indicateurs sur l'efficacité ou non d'un algorithme, l'étude de cette caractéristique est connue par le nom de la complexité algorithmique.
La notion d'efficacité d'un algorithme demande d'ordinaire à considérer l'ensemble des ressources mises en œuvre pour résoudre un problème ; souvent ce qui nous intéresse est de minimiser le temps d'exécution requis par l'algorithme ou l'espace de stockage nécessaire à l'algorithme durant son exécution.

2.2 Complexité d'un algorithme

On distingue deux type de complexité :

1. **La complexité temporelle** : elle correspond au nombre d'instructions élémentaires effectuées au cours de l'exécution de l'algorithme.
2. **La complexité spatial** : correspond au nombre de cases mémoires occupées par les données manipulées par l'algorithme au cours de son exécution.

En général, le temps d'exécution dépend de la taille de l'instance du problème, en particulier plus l'instance est grande, plus le problème demandera plus de temps pour sa résolution. Par exemple si l'on considère un algorithme de tri d'un tableau d'entiers, le nombre d'instructions et l'espace occupé ne seront pas les mêmes si on travaille sur un tableau de 10 entiers ou sur un tableau de 10 000 entiers. On exprime donc le temps en fonction de la taille d'une instance générale du problème considéré, souvent notée n.
Notons aussi que le temps d'exécution (ou la complexité temporelle) d'un algorithme peut varier sur des instances de même taille. En effet, pour chercher l'existence d'un élément dans un tableau contenant n entiers, il est claire que l'élément cherché peut être trouvé à la premier position, donc la complexité dans ce cas est 1 ; comme on peut ne pas trouver l'élément cherché ou le trouver à la dernière position, dans ce cas il faut exécuter n testes de

comparaisons.
Pour que notre calcul de la complexité soit fiable, on définit généralement une complexité en considérant la pire instance possible parmi toutes les instances de taille n (la complexité dans les pires des cas). D'où la définition suivante :

Définition 2.1. Soient P un problème et A un algorithme qui résout P. Notons $I_{(P,n)} = \{x/x$ est une instance de P et $|x| = n\}$ et Φ_A est une application qui à toute instance x fait associer le temps d'exécution de A sur x. **La fonction de complexité** de l'algorithme A est une application C_A définie sur l'ensemble des entiers naturels par :

$$C_A(n) = \max_{I_{(P,n)}} \Phi(n)$$

$C_A(n)$ est le nombre maximum d'opérations élémentaires pris par A sur les instances de taille n.

Remarque 2.1. Il existe aussi d'autres types de complexité, la complexité dans le meilleur des cas et la complexité en moyenne. Cependant, la complexité dans le meilleur des cas est souvent peu porteuse d'informations utiles, et la complexité en moyenne, qui nécessite de connaitre la loi de distribution, est souvent trop difficile a calculer. Pour ces raisons, on se contente généralement de la complexité dans le pire des cas.

2.3 Notation grand O, Ω, Θ

En pratique une analyse précise d'un algorithme même dans le pire des cas est presque impossible (sauf pour un algorithme simple). Donc La solution est d'établir *une approximation asymptotique du temps de calcul* de l'algorithme.

2.3.1 La notation grand O

Lors de l'évaluation de la complexité d'un algorithme, il est habituel d'utiliser *la notation O* qui permet d'exprimer une borne supérieure asymptotique sur le temps d'exécution de l'algorithme.
La notation O, dite aussi symbole de Landau, décrit le comportement asymptotique d'une fonction, exprimé à l'aide d'une autre fonction généralement plus simple.

Définition 2.2. $f(n) = O(g(n))$ ($\ll f(n)$ est en grand O de $g(n) \gg$) quand $n \to \infty$ si et seulement si $\exists c > 0$, et un entier n_0 tels que $\forall n \geq n_0$, $|f(n)| \leq c|g(n)|$.
Intuitivement, ceci signifie qu'à partir de n_0 et à un facteur constant près, f ne croît pas plus rapidement que g.

Exemple 2.5. Soit $f(n) = 6n^4 - 2n^3 + 5$. Choisissons $n_0 = 1$. Alors pour tout $n \geq n_0$, on a

$$\begin{aligned} |6n^4 - 2n^3 + 5| &\leq 6n^4 + |2n^3| + 5 \\ |6n^4 - 2n^3 + 5| &\leq 6n^4 + 2n^4 + 5n^4 \\ &= 13n^4 \\ &= 13|n^4| \end{aligned}$$

Ainsi, en prenant $c = 13$, on a $f(n) = O(n^4)$. Autrement dit, à un facteur constant près, $f(n)$ ne croît pas plus rapidement que n^4.

Il est facile de voir qu'un polynôme $P(n)$ de degré k est toujours en $O(n^k)$.

2.3.2 Notation Ω

De façon similaire, la notation Ω permet de donner une borne asymptotique inférieure sur le temps d'exécution d'un algorithme sur une entrée de taille n.

Définition 2.3. Soient f et g deux fonctions $f; g : \mathbb{N} \to \mathbb{R}_+$.
On note $f = \Omega(g)$ lorsqu'il existe un entier n_0 et une constante réelle positive $c\prime$ tel que pour tout $n \geq n_0$, $f(n) \geq c\prime g(n)$.

2.3.3 Notation Θ

Définition 2.4. Soient f et g deux fonctions $f; g : \mathbb{N} \to \mathbb{R}_+$.
On dit que $g(n)$ est une borne approchée asymptotique pour $f(n)$ et l'on note $f = \Theta(g(n))$ lorsqu'il existe un entier n_0 et deux constantes strictement positives c et $c\prime$ tel que :

$$c\prime g(n) \leq f(n) \leq cg(n)$$

Ceci revient à dire que $f(n)$ est égale à $g(n)$ à un facteur constant prés.

Remarque 2.2. Si une fonction $f(n)$ est à la fois en $O(g(n))$ et en $\Omega(g(n))$, on dit que $f(n)$ est en $\Theta(g(n))$.

Pour compléter ces trois notations, Woeginger [35] a introduit la notation O^* qui autorise la suppression de tous les termes bornés par un facteur polynomial dépendant de la taille de l'entrée. Ainsi une complexité de la forme $O(f(n) \cdot poly(n))$, où n est la taille de l'entrée et $poly(n)$ un polynôme dépendant de n, se notera $O^*f(n)$. Cela nous permettra en particulier d'insister sur la croissance exponentielle de la fonction qui borne le temps

d'exécution. Par exemple un temps d'exécution $2 + n^3 \cdot 1.8788n$ se réécrira $O^*(1.8788n)$. Cette nouvelle notation asymptotique se justifie par le fait que $2 + n^3 \cdot 1.8788n$ est compris entre $n^3 \cdot 1.8788n$ et $n^3 \cdot 1.8789n$ pour toute valeur de n supérieure à 26. Par conséquent, la fonction $2 + n^3 \cdot 1.8788n$ pourra également se réécrire simplement sous la forme $O^*(1.8789n)$.

2.4 Les Classes de Complexité

Nous nous intéressons maintenant à la complexité des problèmes, et on tentera de les classifier selon la complexité des algorithmes les résolvant.
La complexité d'un problème est la complexité du meilleur algorithme qui permet de le résoudre. Si la complexité de cet algorithme est majorée par un polynôme en la taille des données, le problème est dit polynomial (ou facile) et l'algorithme qui le résout algorithme efficace (ou polynomial tout court), autrement le problème est exponentiel (ou difficile).

2.4.1 La classe P

La classe de complexité P contient l'ensemble des problèmes de décisions pour lesquels il existe un algorithme déterministe qui s'exécute au pire des cas en temps polynomial pour les résoudre. Typiquement, ce temps d'exécution est de la forme $O(n^k)$ où n représente la taille de l'entrée et k est une constante (qui dépend de l'algorithme considéré). Bien souvent, chercher un algorithme dont la constante k est la plus petite possible pour résoudre un certain problème est un véritable défi.
Les problèmes de cette classe sont dits faciles. Ce sont ceux que l'on sait résoudre efficacement.

Remarque 2.3. Notons aussi les deux versions de décision et d'optimisation d'un même problème ont une même complexité temporelle.

2.4.2 La classe NP

La classe de complexité NP (Not Determinist Ploynomial) contient l'ensemble des problèmes de décision pour lesquels il existe un algorithme non déterministe qui permet de les résoudre en temps polynomial. Autrement dit, pour lesquels il existe un algorithme polynomial qui permet de vérifier une solution (certificat) au problème en temps polynomial en la taille des données ou encore, les problèmes de la classe NP sont ceux que l'on peut résoudre en énumérant l'ensemble des solutions possibles et en les testant à l'aide d'un algorithme polynomial.

Il est évident que $P \subseteq NP$ car, si l'on peut trouver la solution en temps polynomial, on peut forcement la vérifier en temps polynomial.
La grande question ouverte en informatique est de savoir si $P = NP$ ou si $P \neq NP$. De nombreuse personnes pensent que l'inclusion est stricte, mais cela reste à prouver. L'intuition est fondée sur le fait qu'il est plus facile de vérifier si un certificat est une solution que de trouver la solution elle même.

2.4.3 La classe NP-complet

La raison principale qui laisse penser que $P \neq NP$ est l'existence de cette classe des problèmes NP complets. Elle est constituée par les problèmes les plus difficiles de NP. Cette famille à une propriété très particulière c'est que ces problèmes sont tous équivalents en termes de difficultés, si l'on savait résoudre l'un d'eux en temps polynomial, alors on pourrait tous les résoudre en temps polynomial, et par conséquent on obtiendrait la relation $P = NP$. Jusqu'à présent, de nombreux problèmes ont été montrés NP-complets et pour aucun d'entre eux un algorithme polynomial n'a été trouvé.

2.5 La réduction polynomiale :

Une approche usuelle pour résoudre un problème donné consiste à le transformer en un autre dont la solution est déjà connue, ensuite convertir cette solution en une solution du problème original. Bien sûr, cette approche n'est réalisable que si la transformation peut se faire rapidement. Le concept de réduction polynomiale formalise cela.
La théorie de la NP-complétude fait appel aux réductions polynomiales pour montrer qu'un problème est NP-complet ou pour augmenter cette liste des problèmes NP-Complet.

Soit deux problèmes π_1 et π_2, on dit que π_1 se réduit en temps polynomial (ou se réduit polynomialement) à π_2, noté $\pi_1 \leq \pi_2$, s'il existe un algorithme qui réduit toute instance x_1 de π_1 en une instance x_2 de π_2, admettant la même réponse que x_1.

Remarque 2.4. La relation $\leq$ est transitive et $\pi_1 \leq \pi_2$ signifie que π_1 n'est pas plus difficile que π_2. Ainsi, "π_1 est polynomialement réductible à π_2" signifie que si l'on connait un algorithme polynomial résolvant π_2, alors il existe un algorithme polynomial pour π_1, on en déduit que $\pi_1 \in P$. En effet, on traduit les instances de π_1 en instances de π_2, puis on résout π_2 et enfin on retraduit les solutions de π_2 en solutions de π_1, tout cela se fait en temps polynomial.

Pour montrer qu'un problème est NP-complet, on a le théorème suivant :

Théorème 2.1. *Un problème π est NP-complet si et seulement si :*

1. $\pi \in NP$.
2. *Pour tout problème* $\pi_1 \in NP$, $\pi_1 \leq \pi$.

Ce théorème permet donc de montrer commodément la NP-complétude d'un problème. Pour pouvoir l'utiliser il faut néanmoins connaître au moins un problème NP-complet. Et cela a été le travail de Cook [8] qui a montré en 1971 que le problème SAT est NP-complet.

SAT
Entrée : Une formule booléenne sur un ensemble de variables booléennes X.
Question : Existe-t-il une affectation de valeurs de vérité pour X tel que la formule soit satisfaite ?

A partir de SAT, on montre progressivement la NP-complétude d'autres problèmes qui, à leur tour, peuvent être utilisés pour démontrer la NP-complétude de nouveaux problèmes, et ainsi de suite. Par exemple, à partir de SAT, ils ont pu montrer la NP-complétude de $3 - SAT$. Et grâce à $3 - SAT$ ont démontré que MIS (problème du stable maximum) est NP-complet (Karp(1972)).

Depuis la démonstration de ce résultat, de nombreux problèmes ont été montrés NP-complets et, en 1979, Garey et Johnson en recensent déjà plus de 300 dans un ouvrage qui fait encore référence [25]. À ce jour, aucun algorithme s'exécutant en temps polynomial pour résoudre un problème NP-complet n'a été trouvé. Ce sont ces problèmes qui ont motivés de larges travaux (algorithmes d'approximation, algorithmes à paramètres fixés, études de décompositions de graphes, ...).

CHAPITRE 3

EXEMPLE DE PROBLÈMES DIFFICILES

Dans ce chapitre, nous allons voir comment utiliser la notion de réduction polynomiale pour montrer la NP-complétude de nouveaux problèmes.

3.1 Problème de Clique

Problème de Clique :
Soit un graphe $G = (V, E)$, et un entier k tel que $1 \leq k \leq |V|$.
Existe-t-il une clique de taille k dans G ?

Définition 3.1. Une variable booléenne est une variable qui peut prendre deux valeurs, 0 (faux) où 1 (vrai). On appelle **un littéral** une variable booléenne x où sa négation $\overline{x}$, et **une clause** une disjonction de littéraux. On note **une disjonction** de deux littéraux "p" et "q" par $p \vee q$ (p OU q) et **une conjonction** $p \wedge q$ (p ET q). Toute conjonction de clauses disjonctives est appelée une formule *sous forme normale conjonctive*, ou tout simplement CNF.
Une formule booléenne P est dite *satisfiable*, s'il y a une affectation de vérité à ses variables pour laquelle la valeur de la formule est 1. Autrement dit, une affectation satisfait P si et seulement si pour chaque clause de P, au moins un littéral est égal à 1. Dans ce cas, nous disons que la formule est satisfaite par l'affectation.

Exemple 3.1. Prenons un ensemble de 3 variables booléennes $X = \{x_1, x_2, x_3\}$, et une formule $f := (x_1 \vee x_2) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_2} \vee \overline{x_1})$ composée donc de 3 clauses, $C_1 = (x_1 \vee x_2)$, $C_2 = (\overline{x_1} \vee x_3)$ et $C_3 = (\overline{x_2} \vee \overline{x_1})$, elle est

satisfiable par l'assignation $x_1 = vrai$, $x_2 = faux$, $x_3 = vrai$. En effet comme $x_1 = 1$, la clause C_1 est vérifiée ; comme $x_3 = 1$, la clause C_2 est vérifiée ; et comme $x_2 = 0$ et $\overline{x_2} = 1$ la clause C_3 est vérifiée, donc f est satisfaite.

Mais toutes les formules booléennes ne sont pas satisfiables. Par exemple, aucune assignation ne rend vraie la formule suivante : $g := (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee x_1) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$
Ainsi, une formule $3-CNF$ est une formule sous forme normale conjonctive où toutes les clauses ont exactement 3 littéraux. Et $3-SAT$ est le problème qui correspond a la satisfiabilité d'une formule $3-CNF$.

3.1.1 Réduction polynomiale

Théorème 3.1. *Le problème de CLIQUE est NP-complet.*

Démonstration. Afin de prouver que le problème clique est NP-difficile, nous devront réduire un problème NP-difficile connu à ce problème. Choisissons le problème $3-SAT$ pour effectuer cette réduction.

Théorème 3.2. *[6] Le problème $3-SAT$ est NP-complet.*

CLIQUE $\in NP$

Certificat : liste des sommets de la clique.
On peut vérifier en temps quadratique, que chaque paire de sommets du certificat, sont relier par une arête appartenant a E. Ainsi, le problème clique est inclut dans NP.

$3-SAT \leq$ **Clique**

Soit f une instance de $3-SAT$, c'est-à-dire une formule sous forme conjonctive avec p clauses de taille 3.

$$f = (l_1 \vee l_2 \vee l_3) \wedge (l_4 \vee l_5 \vee l_6) \wedge ... \wedge (l_{3p-2} \vee l_{3p-1} \vee l_{3p})$$

On construit un graphe non orienté tel que si f est satisfiable si seulement si ce graphe contient une clique de taille k , soit G ce graphe, dont l'ensemble des sommets est l'ensemble $V = \{l_1, ..., l_{3p}\}$ de tous les littéraux de f. Deux sommets de G sont reliés par une arête s'ils ne font pas partie de la même clause et s'ils ne sont pas contradictoires (l'un n'est pas égal à la négation de l'autre).

Exemple 3.2. $f = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3})$

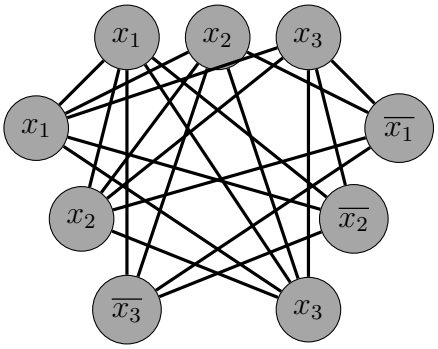


FIGURE 3.1 – La réduction polynomial du problème 3-SAT au problème de CLIQUE

Le graphe G est un graphe à $3p$ sommets, c'est donc une instance de taille polynomiale en la taille de f. De plus deux littéraux d'une même clause ne sont jamais reliés par une arête, une clique peut donc contenir au plus un littéral par clause et elle est de taille au plus p. On fixe alors dans notre instance $k = p$, et on montre alors l'équivalence $3 - SAT \equiv$ Clique et ceci revient à montrer : f satisfiable $\Leftrightarrow$ G contient une clique de taille k

— Supposons d'abord que f est satisfiable. Il existe donc une affectation des variables telle que f vaille 1. Ceci signifie qu'au moins un littéral par clause vaut la valeur 1. Choisissons un tel littéral dans chacune des clauses pour former un ensemble de k littéraux. Comme tous ces littéraux sont vrai, deux d'entre eux ne peuvent pas être antagonistes (contradictoires) donc ils sont reliés par une arêtes. Ainsi ces k littéraux forment une clique de taille k dans G.

— Supposons maintenant que G contienne une clique de taille k.
Comme les littéraux d'une même clause ne sont pas reliés, cette clique contient un littéral exactement dans chaque clause. $f = (\_ \vee l_1 \vee \_) \wedge (l_2 \vee \_ \vee \_) \wedge ... \wedge (\_ \vee l_k \vee \_)$ on peut affecter à 1 chaque littéral de la clique , cela est possible car G ne peut pas avoir des arêtes entre deux littéraux contradictoire.
Montrons alors qu'il existe une affectation qui rend tous ces littéraux (les sommets de la clique) l_i $(i = 1...k)$ vrai.
Chaque littéral de cette clique est égal à x_i ou à $\overline{x_i}$. Pour que ce littéral vaille 1, on impose la valeur 1 ou 0 à la variable correspondante x_i. Et comme tous les littéraux de la clique sont reliés par une arête, ils ne sont pas contradictoires deux à deux. Ceci signifie que deux littéraux quelconques de la clique ils concernent deux variables distinctes x_i et x_j avec $i \neq j$ ou alors, ils concernent la même variable x_i mais ils imposent la même valeur à cette variable .

En affectant la valeur 1 à chaque variable appartenant à la clique,et n'importe quelle valeur à chacune des autres variables, on obtient une affectation qui rend la formule f satisfiable.

Ceci complète la réduction car nous avons montré que la transformation est valable dans les deux sens , ainsi donc le problème de CLIQUE est NP-complet. □

3.2 Couverture des sommets (Vertex Cover)

Une couverture des sommets VC dans un graphe $G = (V, E)$ est un sous-ensemble de sommets $V' \subseteq V$ tel que chaque arête de G a au moins une extrémité dans V'. (Voir la figure 3.2)
Le problème de couverture de sommets VC se formule comme suit :
Entrée : Un graphe G et un entier k.
Question : Existe-t-il une couverture de sommets de taille au plus k ?

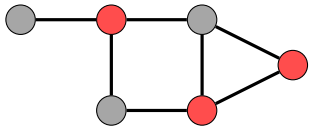


FIGURE 3.2 – Couverture de sommets

3.2.1 Réduction polynomiale

Théorème 3.3. *[25] Le problème vertex cover est NP-complet*

Démonstration. 1. $VC \in NP$

Étant donné une instance $G = (V, E)$ et un entier positif k, le certificat pour le probléme de couverture de sommets est un sous-ensemble V' de V, qui contient les sommets de couverture. On peut vérifier en temps polynomial que l'ensemble V' est une couverture de sommets de taille k, en vérifiant que chaque arête de G est incidente à l'un de ces sommets. Ainsi, le probléme de couverture de sommets est dans la classe NP.

2. *Clique* $\leq_p$ *VC*

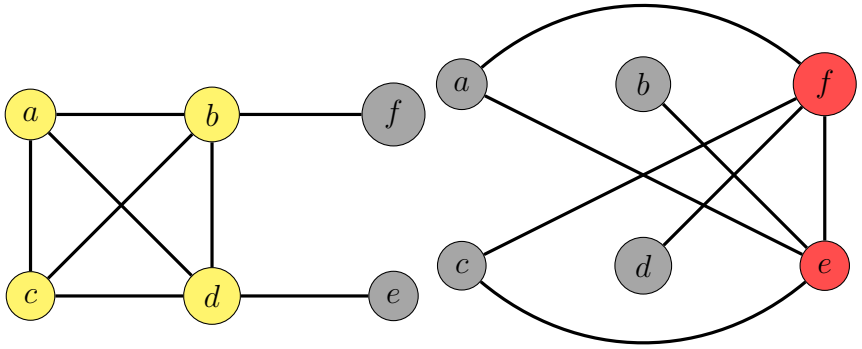
   Pour prouver que VC est NP-difficile, nous prenons un problème qui a déjà été prouvé comme étant NP-difficile, et montrons que ce problème peut être réduit au problème de vertex cover. Pour cela, nous considérons le problème de la clique, qui est NP-complet.
   Ici, le problème est de savoir s'il existe une clique de taille k dans le graphe donné. Par conséquent, une instance du problème de la clique est un graphe $G = (V, E)$ et un entier positif k, et nous devons vérifier l'existence d'une clique de taille k dans G.
   Maintenant, nous devons montrer que toute instance (G, k) du problème de la clique peut être réduit à une instance du problème de couverture de sommets. Considérons le graphe $\overline{G}$ qui consiste en toutes les arêtes non dans G, mais dans le graphe complet en utilisant tous les sommets de G. Appelons cela le complément de G. Maintenant, le problème de trouver si une clique de taille k existe dans le graphe G est le même que le problème consistant a trouver s'il existe une couverture de sommets de taille $|V| - k$ dans $\overline{G}$. Nous devons montrer que c'est bien le cas.

   $\Rightarrow$ Supposons qu'il existe une clique de taille k dans G. Soit V' l'ensemble des sommets de la clique. Cela signifie que $|V'| = k$. Dans le graphe complémentaire $\overline{G}$, choisissons une arête quelconque (u, v). Alors au moins un sommet parmi u ou v doit être dans l'ensemble $V - V'$. En effet, si u et v appartenaient tous deux à l'ensemble V', alors l'arête (u, v) appartiendrait à V', ce qui signifierait à son tour que l'arête (u, v) est dans G. C'est impossible puisque (u, v) n'est pas dans G. Ainsi, toutes les arêtes de $\overline{G}$ sont couvertes par des sommet dans l'ensemble $V - V'$.

   $\Rightarrow$ Supposons maintenant qu'il existe une couverture de sommet V'' de taille $|V| - k$ dans $\overline{G}$. Cela signifie que toutes les arêtes de $\overline{G}$ sont connectées à un sommet de V''. Par conséquent, si nous choisissons une arête (u, v) de $\overline{G}$, les deux sommets ne peuvent pas être en dehors de l'ensemble V''. Cela Signifie, que toutes les arêtes (u, v) telle que u et v sont en dehors de l'ensemble V'' appartiennent au graphe G, c'est à dire que ces arêtes constituent une clique de taille k. Ainsi, on peut dire qu'il existe une clique de taille k dans le graphe G si et seulement s'il existe une couverture de sommets de taille $|V| - k$ dans $\overline{G}$, et donc, toute instance du problème de la clique peut être réduite à une instance du problème de couverture de sommets. Ainsi, on conclu

que la couverture des sommets est NP-difficile.

Étant donné que la couverture de vertex appartient à la fois aux classe NP et NP-difficile, d'où le problème vertex cover est NP-complet.
Pour comprendre la preuve, considérons l'exemple de graphe suivant et son complémentaire : □



Graphe G Graphe $\overline{G}$

$V' = \{a, b, c, d\}$ $V'' = \{e, f\}$

FIGURE 3.3

3.3 Problème d'ensemble dominant (Dominating set)

Ensemble dominant

Un ensemble dominant dans un graphe non orienté $G(V, E)$ est un sous ensemble de sommets $D \subseteq V$, tel que chaque sommet de G est soit dans D, soit adjacent à au moins un des sommet de D.(Voir la figure 3.4)
La version décision du problème de l'ensemble dominant est :
Entrée : Un graphe $G(V, E)$ et un entier k.
Question : Existe-t-il un ensemble dominant de taille au plus k dans le graphe G ?

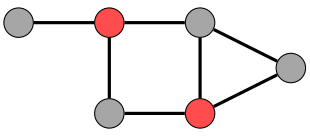


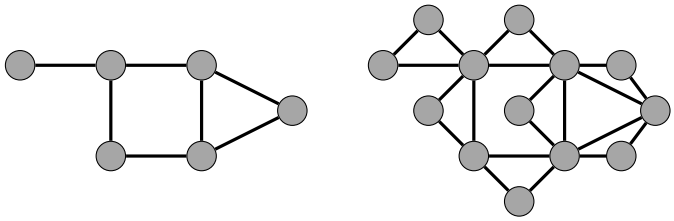
FIGURE 3.4 – Ensemble dominant (Les sommets colorés en rouge)

Afin de prouver que le problème d'ensemble dominant D est NP-difficile, nous devons réduire un problème NP-difficile connu à ce problème. Choisissons le problème vertex cover pour effectuer cette réduction.
Pour prouver cela nous allons suivre les étapes suivante :

— Montrer que l'ensemble dominant $D \in NP$
— Montrer que le problème vertex cover se réduit polynomialement au problème d'ensemble dominant.

1. **L'ensemble** $D \in NP$
   Il est facile de vérifier si un ensemble D de taille k est un ensemble dominant, il suffit de vérifier si les sommets $\{V - D\}$ sont adjacent au sommets de l'ensemble D et cela se fait en temps polynomial. Ainsi l'ensemble dominant est dans NP

2. **Cover vertex** $\leq_p$ **D**
   Pour passer d'un problème de couverture de sommet à un problème d'ensemble dominant nous avons besoin de construire un nouveau graphe à partir d'un graphe donné. Soit $< G, k >$ une instance du problème de couverture de sommets, on construit un nouveau graphe G', en ajoutant à G de nouveaux sommets, pour représenter les arcs du graphe initial. Plus précisément, pour chaque arête (u, v) de G, ajoutons un sommet uv et deux autres arête (u, uv) et (v, uv) pour former un triangle. Ce processus est illustré sur la figure 3.5 ci dessous. Le nouveau graphe G' peut-être obtenu en temps polynomial,



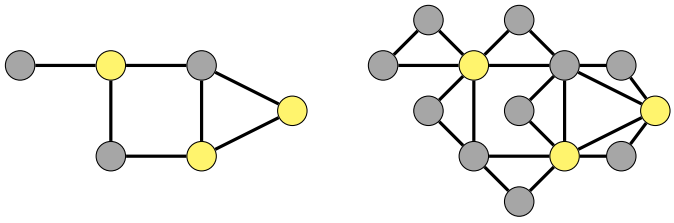
Graphe G *Graphe G′*

FIGURE 3.5

en ajoutant de nouvelles arêtes correspond au nouveau sommet.
Pour établir l'exactitude de la réduction, nous devons montrer que G à une couverture de sommets de taille k si G' admet un ensemble dominant de taille k.

L'ensemble dominant de taille k **dans** $G \Leftrightarrow$ **Vertex cover de taille de taille** k .

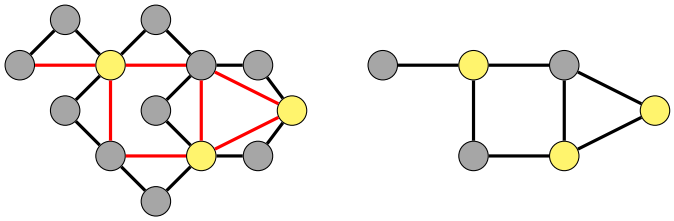
$\Rightarrow$ Supposons que C est une couverture de sommets de taille k dans G.
Chaque arête de G a un des sommets appartenant à C. Par conséquent, pour chaque arête e, constituée de sommets $\{u, v\}$, au moins u ou v fait partie de cover vertex. Donc si u est inclut dans C, alors le sommet adjacent est v, il est aussi couvert par un élément de C.
Maintenant, pour les sommets nouvellement ajoutés pour construire G', ils sont aussi couvèrent par C, car comme l'on sait déjà si (u, v) une arête de G on ajoute un sommet uv pour construire G', tel que ce nouveau sommet sera adjacent à u et adjacent à v, dont l'un est au moins inclut dans C. Alors le sommet nouvellement ajouté uv sera aussi couvert par son voisin qui est dans C, ainsi, donc il est dominé par ce voisin. Par conséquent, les sommets supplémentaires pour toutes les arêtes sont également couverts par C, ils sont donc dominé par les sommets de C. On conclu que l'ensemble des sommets formant vertex cover de taille k dans G forme l'ensemble dominant de même taille dans G' (Voir la figure 3.6) .



Vertex cover *Ensemble dominant*

FIGURE 3.6

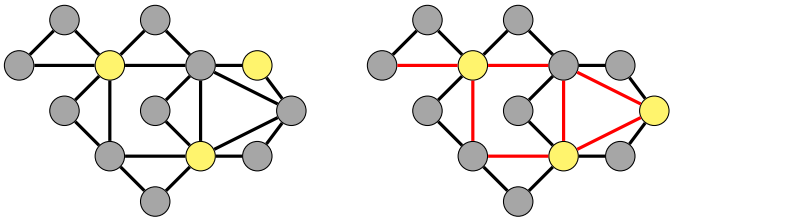
$\Rightarrow$ Supposons que D est un ensemble dominant de taille k dans G'.
Deux possibilité peuvent se présenter, soit le sommet dans D est un sommet d'origine, soit il appartient au sommet nouvellement ajouté uv pour chaque arête (u, v).
Dans le premier cas, puisque D est un ensemble dominant dans G', donc, chaque sommet $v \in \{V' - D\}$ est dominé par au moins un sommet de D, particulièrement les anciens sommets, et couvrent toutes les arêtes du graphe G contenu dans G', autrement dit chaque arête de G à au moins une extrémité dans D. Ainsi, on conclu que D est une couverture de sommets dans le graphe G.(Voir la figure 3.7)



Ensemble dominant D *Cover vertex C*

FIGURE 3.7

Dans le second cas, puisque chaque nouveau sommet est relié aux deux sommets de l'arête (u, v) de G, il peut donc remplacé par u ou v et puisque ces 3 sommets forment un triangle, par conséquence même en le remplaçant par u ou v, on peut couvrir tout les sommets qui ont été couvèrent avant de le remplacer. Cela conduira à l'élimination de tous les sommets nouvellement ajoutés, tout en couvrant toutes les arêtes du graphe G'.
Les sommets de l'ensemble dominant modifié D' dominent tous les sommets nouvellement ajoutés et couvrent toutes les arêtes du graphe G (Voir la figure 3.8), avec au moins un sommet dans D' pour chaque arête du graphe G. On déduit donc, si G' admet un ensemble dominant de taille k, le graphe G aura une couverture de sommet de taille au plus k .



Ensemble dominant D *Ensemble dominant modifié D′*

FIGURE 3.8

Ainsi on peut dire que le graphe G' contient un ensemble dominant si seulement si graphe G contient une couverture de sommet. Par conséquent, toute instance du problème de l'ensemble dominant peut-être réduite à une instance du problème de la couverture des sommets. Ainsi l'ensemble dominant est également NP-difficile.

Remarque 3.1. Si un graphe G contient un sommet isolé u alors, ce sommet

ne peut être dominé sauf s'il est inclut dans l'ensemble dominant et comme il n'est incident a aucune arête, il n'a pas besoin d'être dans la couverture de sommet. Notons V_I l'ensemble des sommets isolés dans G et n_I le nombre de ces sommets isolés. Le nombre de sommets nécessaire pour l'ensemble dominant sera est égale à $k' = k + n_I$.
Pour établir une correction pour la réduction, on a besoin de montrer que G possède une couverture de sommets de taille k si G' contient un ensemble dominant de taille k'.
Pour prouver cela, il suffit d'ajouter les sommets isolé à l'ensemble dominant, et suivre la même procédure dans le cas d'un graphe connexe pour démontrer la NP-complétude de problème de dominant.

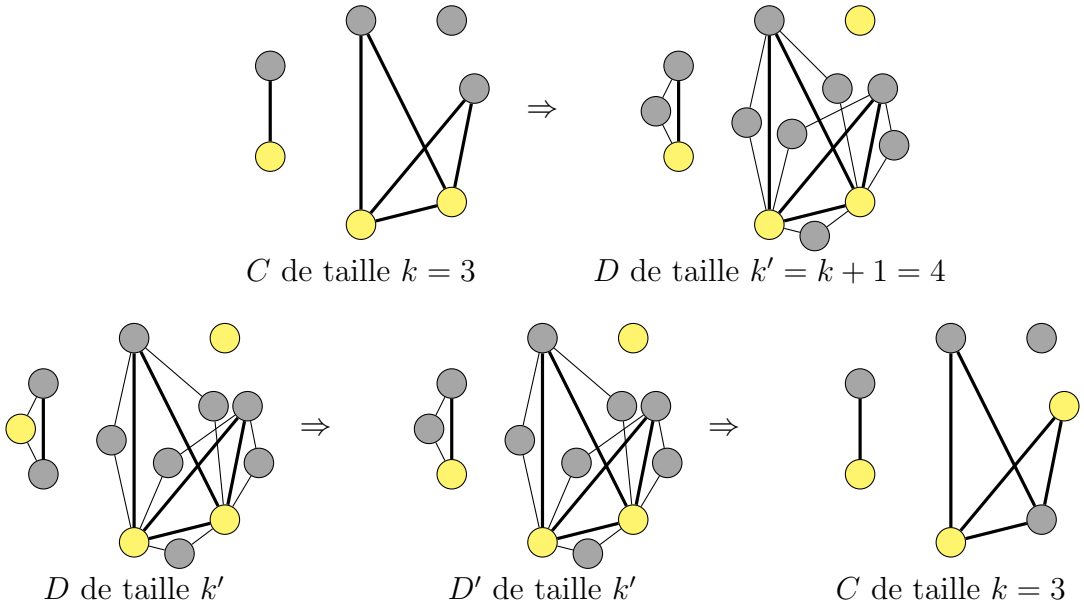


FIGURE 3.9 – Correction de la réduction de vertex cover à l'ensemble dominant (avec $k = 3$ et $I = 1$)

CHAPITRE 4

FAIRE FACE À UN PROBLÈME DIFFICILE

Lorsque l'on est confronté à un problème difficile, on ne sait pas construire d'algorithme qui puisse le résoudre de façon exacte et en temps polynomial. Sinon, cela signifierait que $P = NP$. Dans ce cas plusieurs approches sont possibles afin de pouvoir traiter le problème tout de même :
- Utiliser des algorithmes exacts et à temps d'exécution exponentiel.
- Utiliser des algorithmes d'approximation.
- Utiliser des heuristiques et méta-heuristiques.
- Se restreindre à des classes de graphes.

Nous présentons dans ce chapitre, les algorithmes exacts qui permettent de trouver des solutions optimales en temps exponentiel. Puis nous introduirons les algorithmes d'approximation qui permettent de trouver des solutions dont la qualité est garantie, en temps polynomial. En suite les heuristiques qui, bien que la qualité de leurs solutions n'est pas garantie, trouvent souvent en temps acceptable des solutions de bonnes qualités. Enfin, nous présenterons la restriction à des classes de graphes, qui permet de trouver des solutions optimales dans des cas très particuliers.

4.1 Les algorithmes exacts à temps d'exécution exponentiel

Résoudre un problème NP-complet exactement et efficacement est l'un des défis les plus importants des quarante dernières années en informa-

tique. Un algorithme polynomial pour résoudre un tel problème établirait immédiatement la relation $P = NP$. Malheureusement, à ce jour il n'existe aucun algorithme permettant d'accomplir cette tâche. De plus, de nombreux chercheurs conjecturent que $P \neq NP$ et donc qu'un tel algorithme est fortement improbable à obtenir. C'est pourquoi l'étude des algorithmes exponentiels est devenue un domaine de recherche conséquent où aussi bien la conception que l'analyse du temps d'exécution demandent d'importants efforts pour obtenir des algorithmes meilleurs que les simples algorithmes naïfs.

Dès lors, une approche possible pour résoudre un problème de façon exacte est de le résoudre de manière exhaustive. Cette attaque brute, triviale mais simple, permet effectivement de trouver une solution exacte. Souvent cette approche n'a d'intérêt que si l'entrée considérée n'est pas trop grande puisque le temps d'exécution croît exponentiellement en fonction de la taille de l'entrée. De toute manière, sous l'hypothèse $P \neq NP$, on ne peut pas espérer obtenir un algorithme polynomial et seul un algorithme exponentiel pourrait résoudre exactement le problème. Ainsi, nous cherchons à obtenir un algorithme pour résoudre exactement un problème dont le temps d'exécution au pire des cas est significativement meilleur que celui de l'algorithme naïf réalisant cette tâche par recherche exhaustive. Ce nouvel angle d'attaque a depuis quelques années motivé des chercheurs du domaine à concevoir, mais aussi à analyser, de tels algorithmes. Comme déjà indiqué, il est préférable d'avoir un algorithme dont le temps d'exécution est en $O(1.1^n)$ qu'un algorithme en $O(2^n)$ pour résoudre un certain problème. Bien sûr le temps d'exécution reste exponentiel en la taille de l'entrée, mais rappelons que notre objectif principal est de résoudre des problèmes pour lesquels un algorithme polynomial est improbable, sauf si $P = NP$.

Ces dernières années ont également vu la puissance des machines continuer à augmenter, les ordinateurs proposés aujourd'hui au grand public sont capables d'effectuer un grand nombre d'opérations : un processeur cadencé à 3GHz effectue trois milliards d'opérations par seconde. Un espoir est alors d'utiliser un ordinateur plus rapide.
La célèbre « loi de Gordon Moore » indique que la puissance des ordinateurs devrait doubler tous les 18 mois. Et donc l'intuition serait qu'avec une machine plus performante, une plus grande entrée pourrait être traitée. Cela est vrai mais si l'on dispose d'un algorithme exponentiel, le gain apporté par un ordinateur deux fois plus rapide ne permet pas de traiter une entrée beaucoup plus grande. Si on dispose d'un algorithme en 2^n qui en une certaine unité de temps t permet de traiter une entrée de taille N_E, alors un ordinateur cent fois plus rapide ne permet que de traiter une entrée de taille

$N_E + 6.64$ (Voir [24]). On ne gagne qu'un facteur additif ! Un algorithme polynomial permet quant à lui de gagner un facteur multiplicatif.
D'un autre coté on peut pas négliger que la taille des instances des problèmes a aussi augmenté au fil de temps. Avec la mondialisation, certains problèmes locaux, se posent maintenant à l'échelle nationale voire internationale. De plus, le développement d'algorithmes de plus en plus complexes et/ou de plus en plus spécifiques à certaines propriétés de certains graphes rend la partie opérationnelle (l'implémentation du code) relativement difficile. Il n'est donc pas étonnant de trouver très peu de résultats expérimentaux dans la littérature récente. L'espoir fondé sur une augmentation de la puissance des machines est donc vain.

Maintenant regardons de plus près la complexité des algorithmes. Supposons qu'au lieu d'avoir un algorithme en 2^n pour résoudre un certain problème, on dispose d'un algorithme en $\sqrt{2^n} \approx 1.41^n$ pour le résoudre. Alors, si avec l'algorithme en 2^n on était capable de traiter une entrée de taille N en une certaine unité de temps, alors avec l'algorithme en 1.41^n, on peut traiter une entrée de taille $2N$ dans cette même unité de temps. Il est donc particulièrement important de concentrer nos efforts sur l'obtention de bons algorithmes exponentiels, c'est-à-dire des algorithmes pour lesquels le temps d'exécution, bien qu'exponentiel, soit le meilleur possible. Pour cela il faut concentrer notre attention sur la base de l'exponentielle apparaissant dans le temps d'exécution. On note qu'une réduction de la base de l'exponentielle permet d'augmenter la taille de l'entrée que l'on peut résoudre en une certaine unité de temps par un facteur multiplicatif.

4.1.1 Techniques principales de conception des algorithmes exponentiels

Dans le domaine des algorithmes exponentiels, Il existe plusieurs techniques de conception permettant d'obtenir des algorithmes meilleurs que les algorithmes naïfs. Les trois méthodes principales sont (pour plus de détails voir [24] [35]) :
- La méthode de programmation dynamique.
- La méthode d'inclusion-exclusion.
- La méthode de branchement et réduction.

Bien entendu, chaque méthode ou technique a ses avantages et ses inconvénients, et pour chaque méthode, il existe des astuces et des évolutions permettant d'essayer de contourner quelque peu ses faiblesses. Il est aussi

possible de combiner les différentes méthodes, comme ont pu le faire F. Fomin et A. Stepanov qui ont combiné la méthode de branchement et réduction avec celle de programmation dynamique. Ou encore J. Van Rooij et al. qui ont combiné la méthode d'inclusion-exclusion avec celle de branchement et réduction.
Nous allons maintenant présenter les idées générales de ces trois méthodes, ainsi que certaines propriétés typiques des algorithmes obtenus par ces méthodes.

La programmation dynamique

La programmation dynamique est une technique classique de conception d'algorithmes pour résoudre des problèmes en temps polynomial, un tel algorithme est basé sur une ou plusieurs récurrences et applique une approche ascendante. Son principe est simple : il est plus facile de résoudre un problème d'une certaine taille, si on connait déjà la solution de tous les sous-problèmes de taille inférieure. c'est-à-dire qu'on commence par résoudre les sous-problèmes les plus petits, et donc les plus faciles, pour ensuite résoudre des problèmes de plus en plus grands, jusqu'à finalement déterminer une solution du problème initial. Pour obtenir un bon temps d'exécution, chaque sous-problème n'est résolu qu'au plus une fois, et une fois la solution d'un sous-problème calculée, elle est conservée dans une structure de données. (Voir Cormen [9] pour plus de détails).
Pour construire un algorithme à temps d'exécution exponentiel, le principe reste le même. On cherche à résoudre localement les problèmes de petites tailles pour pouvoir utiliser les résultats plus tard. Seulement, si le problème est difficile, alors le nombres de sous-problèmes à résoudre est lui-même typiquement exponentiel.
Le principal avantage de cette approche est que l'on évite, le cas échéant, d'avoir à rechercher plusieurs fois la solution à un même problème. L'inconvénient majeur étant que cette méthode est gourmande en espace mémoire, puisqu'il faut enregistrer et conserver les solutions de tous les sous-problèmes résolus, il est également nécessaire de disposer d'un espace exponentiel.
L'un des algorithmes exponentiels qui utilise la méthode de programmation dynamique est l'algorithme de M. Held et R.M. Karp [16] qui permet de résoudre le problème du voyageur de commerce avec un temps d'exécution en $O^*(2^n)$ sur un graphe de n sommets.

L'inclusion-exclusion

Le principe d'inclusion-exclusion est un principe mathématique. Il permet de compter le nombre d'éléments présents dans une réunion d'ensembles finis, à partir du nombre d'éléments de chaque ensemble et de leurs intersections respectives. En informatique, ce principe peut être utiliser pour compter le nombre de solutions d'un problème à condition que l'on formule ce dernier sous forme de problème de réunion d'ensembles.(Voir [19] [24] pour plus de détails)
L'idée générale de cette formule a été reprise par deux groupes d'auteurs, Björklund et Husfeldt [4] et Koivisto [20], pour obtenir des algorithmes exponentiels en $O^*(2^n)$ résolvant des problèmes de partitionnement fondamentaux, notamment le problème Nombre Chromatique d'un graphe. D'ailleurs les trois auteurs ont uni leurs travaux lors de la publication journal [5]. On peut aussi citer d'autres problèmes comme Steiner Tree [29], ou encore Hamiltonian Path dans un graphe orienté [12] que cette méthode permet d'appréhender.

Branchement et réduction

Le paradigme Brancher et Réduire est particulièrement utilisé pour concevoir des algorithmes exponentiels. On le retrouve sous d'autres dénominations : algorithme à arbre de recherche, algorithme à retour en arrière. Le terme d'arbre de recherche fait référence au fait qu'une exécution peut être assimilée à un arbre de fait qu'il utilise des appels récursifs sur des instances plus petites pour calculer une solution optimale. Le principe de cette méthode est simple : il s'agit de prendre un des éléments du problème et de lui imposer successivement toutes les valeurs qu'il pourrait avoir dans une solution. Cela permet alors de déterminer un certain nombre de sous-problèmes, un par valeur imposée, dont il nous suffira de comparer les résultats lorsqu'ils seront eux-même résolus.
Dans des problèmes de graphe, comme Ensemble Stable ou Ensemble Dominant, cela se traduit par choisir si un sommet sera ou non dans l'ensemble que l'on recherche. En général, faire un tel choix a un impact immédiat sur le reste du graphe. Choisir qu'un sommet sera dans un ensemble stable nous interdit immédiatement tous les voisins de ce sommet dans l'ensemble stable. Ainsi nous n'éliminons pas forcément qu'un seul sommet en faisant notre choix. Pour chaque étape, selon que le nombre de sous-problèmes créés à cette étape, on parlera de branchement ou de réduction [19].
Cette modélisation d'une exécution sous la forme d'un arbre de recherche est particulièrement utile pour analyser la complexité temporelle d'un algo-

rithme. Elle dépend directement du nombre de sous-problèmes à résoudre. Celui-ci correspond au nombre de nœuds dans l'arbre de recherche.

4.1.2 Principaux résultats connus

Plusieurs problèmes réputés difficiles ont suscité un engouement depuis les années soixante et soixante-dix afin de proposer des algorithmes exponentiels pour leurs résolution. On trouve en premier lieu, le problème SAT qui a fait l'objet d'intenses recherches consacrée à l'étude de celui-ci. Dans [11], [10] sont données les premiers résultats qui visent à proposer un algorithme exponentiel. Ce n'est que récemment, qu'un algorithme s'exécutant en $O^{\star}(2^n)$ à donné lieu pour déterminer le nombre chromatique d'un graphe [4], [20].
De manière tout aussi attrayante, le problème Domination a fait naître depuis 2004 plusieurs articles. En moins de deux ans, pas moins de quatre algorithmes ont été proposé. Et de nombreux d'autres problèmes ont fait l'objet de travaux, pour en savoir plus, voir [24].

4.2 Méthodes d'approximations

Lorsqu'un problème est NP-complet, il n'existe à priori aucun algorithme efficace pour résoudre ce problème. Partant de ce constat, cette section présente une alternative classique à la résolution de ce genre de problèmes, c'est une méthode qui permet d'obtenir un algorithme efficace qui consiste (dans le cas de problème d'optimisation) à chercher un résultat approché (à un facteur près). Nous présentons brièvement l'idée derrière ce principe ainsi que les définitions qui y sont rattachées.

4.2.1 Approximation polynomiale

Les algorithmes d'approximation permettent de garantir la qualité des solutions trouvées. Un algorithme d'approximation est une approche de résolution en temps polynomial, dont on arrive à borner la taille des solutions trouvées. Le plus souvent associé aux problèmes NP-complétude, ces algorithmes permettent de les résoudre en temps polynomial tout en préservant une certaine qualité des solutions trouvées. Il est naturel de s'intéresser à la qualité de ces solutions fournies par notre algorithme. Dans cet ordre d'idée, on voudrait avoir une garantie quant à cette qualité. C'est là la problématique de l'approximation polynomiale : construire des algorithmes qui fournissent en un temps polynomial des solutions réalisables ayant une certaine qualité.

Ceci implique la définition d'une mesure de la qualité d'une solution : c'est le rôle du rapport d'approximation.
En effet, on considère généralement que l'article fondateur du domaine est l'article *Approximation algorithms for combinatorial problem* écrit par Johnson en 1974. Johnson y introduit le rapport d'approximation standard et étudie des problèmes classiques pour lesquels il propose des algorithmes (polynomiaux) offrant des garanties de performance. Suite à cet article s'est ensuite développé le domaine de recherche de l'approximation polynomiale. Un très grand nombre de problèmes d'optimisation combinatoire classiques, NP-complets dans leur version décision, ont été étudiés de ce point de vue.

4.2.2 Rapport d'approximation

Tout d'abord, il est à noter que les algorithmes approchés traitent de problèmes d'optimisation. Nous rappelons qu'un problème d'optimisation P est un quadruplet $P = (\mathcal{I}, Sol, m, but)$ où :
– $\mathcal{I}$ est l'ensemble des instances ;
– Sol une fonction telle que pour tout $I \in \mathcal{I}$, $Sol(I)$ représente l'ensemble des solutions réalisables de I ;
– m une fonction telle que pour tout $I \in \mathcal{I}$ et $x \in Sol(I)$, $m(I, x)$ désigne la valeur de la solution x de l'instance I ;
– but $\in$ {min, max} précisant si P est un problème de minimisation ou de maximisation.

Un optimum d'un problème d'optimisation est un élément $x^* \in Sol(I)$ tel que :

$$\begin{cases} m(I, x^*) = min\{m(I, x), x \in Sol(I)\} \text{ si but } = min \\ m(I, x^*) = max\{m(I, x), x \in Sol(I)\} \text{ si but} = max \end{cases}$$

Résoudre le problème sur l'instance I consiste à déterminer la valeur d'une solution optimale x^* pour I, telle que $x^* \in argbut\{m(I, x), x \in Sol(I)\}$.
Dans ce qui suit la valeur de la fonction m pour l'optimum sera noté $opt(I)$.

Le compromis des algorithmes approchés concerne l'exactitude de la solution. ce paradigme s'intéresse à des algorithmes polynomiaux retournant une solution proche d'une solution optimale ayant une bonne qualité garantie de performance. La problématique est d'établir ce qu'est une solution de bonne qualité, c'est-à-dire définir une mesure de la qualité d'une solution.

Le rapport d'approximation standard d'une solution

Cette mesure consistant à comparer toujours la valeur de la solution et la valeur optimale mais en regardant non plus l'erreur absolue mais l'erreur relative. C'est en mesurant la qualité d'une solution à l'aide de ce rapport qu'a été menée l'immense majorité des travaux sur l'approximation.

Définition 4.1. soit un problème P, x une solution réalisable d'une instance I . Le rapport d'approximation standard de x sur I est :

$$\rho_{P(I,x)} = \frac{m(I,x)}{opt(I)}$$

Pour un problème de maximisation ce rapport est dans $[0,1]$, et dans $[1,\infty[$ pour un problème de minimisation. Dans les deux cas, plus il est proche de 1, plus la solution est meilleure.

4.2.3 Algorithme d'approximation

Pour les problème d'optimisation NP-durs intéressants en pratique le mieux qu'on puisse raisonnablement espérer d'un algorithme polynomial est qui renvoie toujours une solution admissible dont la valeur n'est pas trop loin de la valeur optimale.

Maintenant qu'on a donné la définition de rapport d'approximation et la mesure de la qualité d'une solution. Nous pouvons introduire précisément la notion d'algorithmes approchés.

Définition 4.2. Soit P un problème d'optimisation, on dit que A est un algorithme ρ-approché pour une certaine fonction $\rho : \mathcal{I} \longrightarrow \mathbb{R}_+$ si A est un algorithme polynomial en la taille de l'instance, qui renvoie pour toute instance I de P, une solution approchée garantie $A(I)$.

Exemple 4.1. Considérons le problème *Min Couverture De Sommets* où, étant donné un graphe $G = [V, E]$, nous voulons trouver un sous-ensemble de sommets V' couvrant toutes les arêtes du graphe, c'est-à-dire tel que pour toute arête $e = (v_1, v_2) \in E$, $v_1 \in V'$ ou $v_2 \in V'$. Il s'agit alors de minimiser la taille de V'.
Une solution possible consiste à prendre tous les sommets du graphe ; on sera alors sûr de couvrir toutes les arêtes. Supposons maintenant que l'on cherche une solution à ce problème ayant le moins de sommets. Prenons par exemple le graphe de la figure 4.1.

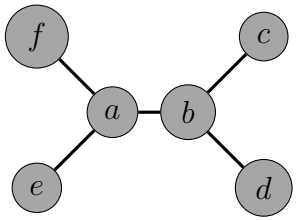


FIGURE 4.1 – Graphe papillon

Sur ce petit graphe, il est facile de voir que les deux sommets centraux forment la solution de taille minimum (ici de taille 2) et couvrant toutes les arêtes. Considérons alors l'algorithme très classique suivant (Algorithme glouton Edges Deletion) : tant que le graphe n'est pas vide, nous choisissons une arête au hasard, ajoutons ses deux extrémités à la solution, et enlevons du graphe ces deux sommets et les arêtes qu'ils couvrent. Sur les graphes de la figure 4.2 , nous avons représenté avec des arêtes en rouge toutes les solutions retournées par Edges Deletion pour le graphe papillon. On constate que ces solutions contiennent au plus 4 sommets, soit deux fois plus que la solution optimale. Pour ce problème et dans tous les graphes en général et sans prétendre connaître la solution optimale, nous obtenons bien une couverture à la fin de l'algorithme, celui-ci étant de toute évidence polynomial. Ainsi, on démontre que l'algorithme proposé est donc une 2-approximation pour Min Couverture De Sommets.

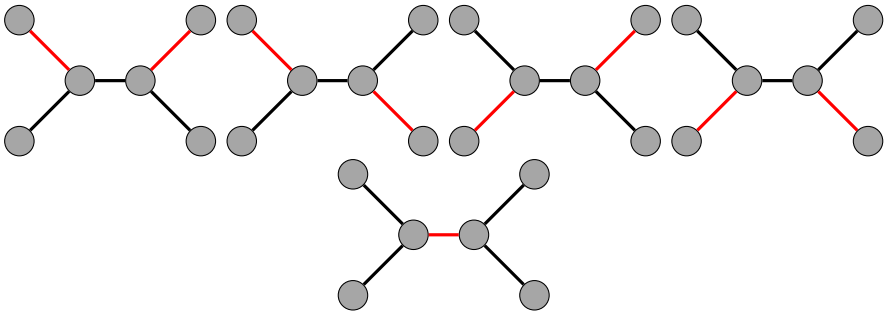


FIGURE 4.2 – Dessins des solutions du graphe papillon

4.2.4 Classes d'approximation

Il est très vite apparu, dans l'étude de l'approximation polynomiale des problèmes avaient des comportements très différents en termes d'approximabilité. Si pour certains problèmes on obtenait des algorithmes à rapport constant, pour d'autres la tâche semblait plus ardue. Pour différencier les

problèmes selon leurs propriétés d'approximabilité, vient alors l'idée de les regrouper dans des classes reflétant un niveau d'approximabilité. Ont ainsi émergé des classes naturelles regroupant ces problèmes.

La classe APX

On remarque que dans la définition 4.2 , le rapport d'approximation ρ est donné comme une fonction dépendant de l'instance. Cependant, pour des raisons évidentes, les algorithmes ρ-approchés les plus intéressants seront lorsque ρ est une fonction constante $\rho(I) = c$ pour tout $I \in \mathcal{I}$. Dans ce cas là, on dira simplement que l'algorithme est c-approché.
L'ensemble des problèmes admettant un algorithme c-approché (le cas d'une minimisation) et $1/c$-approché (le cas d'une maximisation), avec c une constante est appelé APX. La variable c est appelée facteur d'approximation. Ainsi une 1-approximation est un algorithme exact en temps polynomial.
A partir de cette définition, le but dans ce domaine, est donc de trouver un algorithme d'approximation avec une constante c le plus proche de 1, pour un problème donné.

La classe log-APX

Un problème P est dans $log - APX$ s'il existe une constante c telle que P admet un algorithme $c \times \ln(\cdot)$-approché si P est une minimisation, et $\frac{1}{c \times \ln(\cdot)}$-approché si P est une maximisation.

La classe poly-APX

Un problème P est dans poly-APX s'il existe un polynôme p (à valeurs strictement positives) tel que P est p-approché si c'est un problème de minimisation et $\frac{1}{p}$-approché si P est une maximisation.

La classe PTAS (pour Polynomial-Time Approximation Scheme)

Étant donné un problème d'optimisation P, on dit que P est dans $PTAS$, s'il existe un algorithme polynomial A_ε qui est $(1+\varepsilon)$-approché le cas d'une minimisation, et $(1-\varepsilon)$-approché le cas d'une maximisation.
Ainsi, un temps d'exécution typique de PTAS est $O^*(|I|^{\frac{1}{\varepsilon}})$ [33], celui-ci devenant « mauvais » plus la précision demandée diminue.

Une famille $(A_\varepsilon)_{\varepsilon>0}$ d'algorithmes $1-\varepsilon$ (ou $1+\varepsilon$ suivant les cas) approchés sera appelée schéma d'approximation pour P [7].

La classe FPTAS

Soit un problème P , on dit que P est dans $FPTAS$ s'il existe un schéma d'approximation standard $(A_\varepsilon)_\varepsilon > 0$ tel que la complexité de A_ε est polynomiale en la taille de l'instance et en $\frac{1}{\varepsilon}$.

4.2.5 Exemple de problèmes approchés

Comme on l'a déjà vu les problèmes NP-complets ne peuvent être résolus de manière exacte en temps polynomial, en raison de leurs complexité exponentielle. L'objectif est alors de concevoir des algorithmes avec la plus faible complexité exponentielle possible. Nous présentons dans cette sous-section un problème bien connu : l'**indépendant dominant de taille minimum**. Après avoir formellement défini le problème, nous présenterons les premiers résultats sur une famille particulière de graphes : *les line graphes*.

Le problème de l'*indépendant dominant de taille minimum* (minimum Independent Dominating Set : mIDS) est un problème d'optimisation qui consiste à trouver un ensemble indépendant et dominant (IDS) de taille minimum dans un graphe quelconque.
La figure 4.3 montre deux graphes tels que le graphe (a) est un sous-ensemble de type IDS (les sommets foncés) et le graphe (b) est un mIDS.

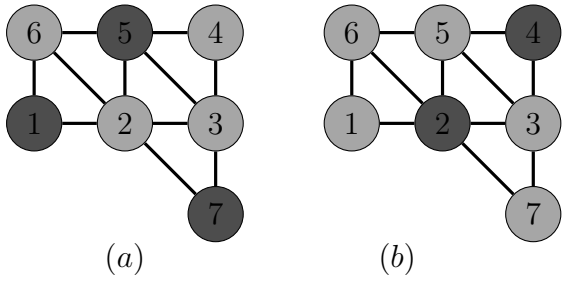


FIGURE 4.3 – Indépendant dominant dans un graphe

Le problème $mIDS$ est connu comme problème NP-difficile [25]. Ce problème n'admet pas une solution approchée sous un facteur $n^{1-\varepsilon}$ avec $\varepsilon > 0$ et en temps polynomial dans les graphes généraux, à moins que $P = NP$, c'est ce que Magnús M. Halldórsson a montrer dans [23]. Dans la section suivante, nous allons montrer que ce problème peut être approximé avec un facteur 2 dans les line graphes.

Facteur d'approximation de 2 dans les line graphes

Dans cette sous-section, nous exhibons une famille de graphes particulière *les line graphes* dans laquelle ce problème est approximable, avec un facteur d'approximation de 2. Pour ce faire, nous passons par le problème du couplage maximal de taille minimum.

Définition 4.3. Soit $G = (V, E)$ un graphe non orienté, un **couplage** (ou matching) $M \subseteq E$ est un ensemble d'arêtes deux à deux non adjacentes, c'est-à-dire n'ayant pas de sommets en communs.
Un couplage est maximal au sens de l'inclusion (MM), s'il n'existe pas d'ensemble strictement plus grand contenant le couplage et qui soit aussi un couplage.

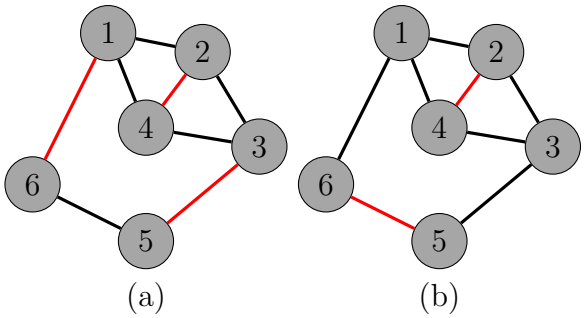


FIGURE 4.4 – Deux couplages dans un graphe

Le graphe (a) de la figure 4.4 montre un couplage maximal (arêtes en rouge), et le graphe (b) montre un couplage maximal de taille minimum (minimum Maximal Matching : mMM).
Construire un MM se fait en temps polynomial avec un algorithme glouton alors que pour le problème d'optimisation, c'est-à-dire le problème de mMM est approximable avec un facteur d'approximation de 2 [26]. Et n'importe quel MM est une 2-approximation du mMM [3].

Définition 4.4. Soit $G = (V, E)$ un graphe non orienté, le **line graphe** de G, noté $L_G = (L_V, L_E)$, est un graphe dont les sommets sont les arêtes de G et 2 sommets sont adjacents si et seulement si leurs arêtes correspondantes dans G sont adjacentes.

La figure 4.5 montre un graphe G et son line graphe L_G, telle que les arêtes de G correspondent aux sommets dans L_G. Ainsi on montrera que chercher un mMM dans G revient à chercher un $mIDS$ dans L_G.

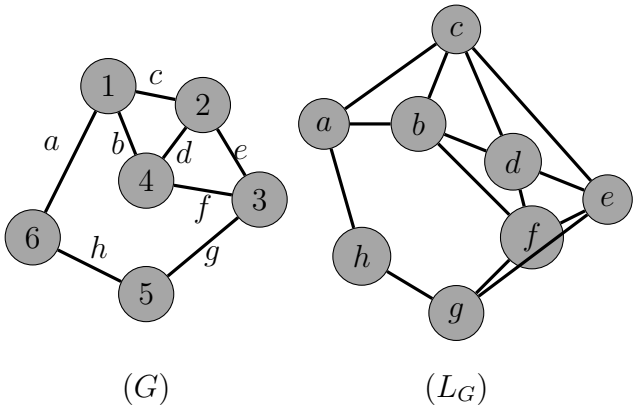


FIGURE 4.5 – Graphe G et son line graphe L_G

Le graphe G est appelé graphe racine de L_G et il est connu qu'à chaque graphe correspond un unique line graphe ; mais tous les graphes ne sont pas nécessairement des line graphes.

Après avoir formellement donné toutes les définitions nécessaires, considérons un line graphe $L_G = (L_V, L_E)$ et le graphe racine correspondant $G = (V, E)$. Nous montrons que quel que soit un indépendant dominant S_{IDS} dans L_G, sa taille est forcément au plus deux fois la taille d'un $mIDS$.

Théorème 4.1. *[32] soit $G = (V, E)$ un graphe racine et $L_G = (L_V, L_E)$ son line graphe, $S_{IDS} \subseteq L_V$ est un indépendant dominant dans L_G si et seulement s'il existe un couplage maximal $M_{MM} \subseteq E$ de G tels que les arêtes de M_{MM} correspondent aux sommets de S_{IDS}. Ainsi on a $|S_{IDS}| = |M_{MM}|$.*

Démonstration. Soit un S_{IDS} indépendant dominant de L_G. Construisons M en prenant les sommets de S_{IDS} correspondent aux arêtes de M dans G. On a $|S_{IDS}| = |M|$. La figure 4.6 montre une telle transformation, tel que le graphe (a) représente le sous-ensemble de sommets S_{IDS} dans L_G et (b) le sous-ensemble d'arêtes M dans G. On remarque que la solution M est bien un couplage, sinon dans le cas contraire, il existerait deux arrêtes qui seraient adjacentes ce qui impliquerait l'existence de deux sommets liées dans S_{IDS} ce qui serait absurde du fait que S_{IDS} est un indépendant.
Ce couplage est maximal, sinon il existerait une arête $e \in E$ qui n'est adjacente à aucune autre arête de M. Ainsi, le sommet associé a l'arête e dans L_G ne serait lié à aucun sommet de S_{IDS}, ce qui est absurde puisque S_{IDS} est un dominant.
Réciproquement, prenons un couplage maximal M_{MM} de G et construisons S dans L_G a partir des arêtes de M_{MM} en prenant les sommets dans L_G qui correspondent aux arêtes de M_{MM} ($|S| = |M_{MM}|$).

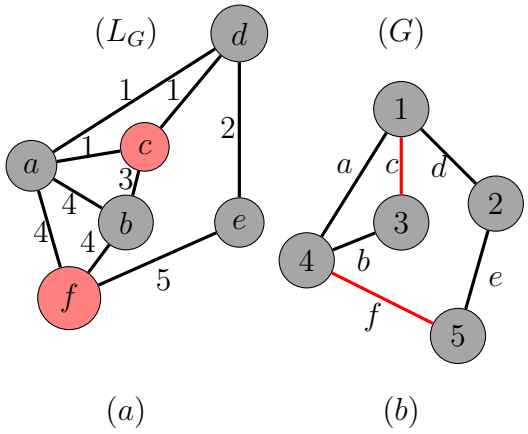


FIGURE 4.6 – (a) IDS dans un line graphe L_G. (b) MM dans le graphe racine associé à L_G

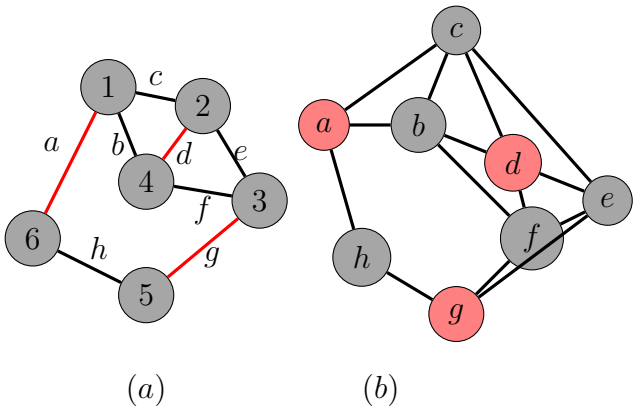


FIGURE 4.7 – Construction d'un IDS à partir d'un MM

La figure 4.7 montre cette construction, telle que le graphe (a) représente le MM dans le graphe G et (b) l'IDS dans le graphe L_G.

Il est facile de voir sur le graphe (b) que S est un indépendant. Si ce n'était pas le cas il existerait deux sommets dans S qui seraient liés. Ainsi, leurs arêtes associées dans M_{MM} seraient adjacentes ce qui est absurde car M_{MM} est un couplage. S est aussi dominant, dans le cas contraire il existerait un sommet dans L_G qui n'est lié a aucun sommet de S ce qui impliquerait l'existence d'une arête associée a ce sommet dans G qui ne serait adjacente à aucune arêtes de M_{MM}, c'est absurde car M_{MM} est maximal. □

corollaire 4.2. *[32] S_{mIDS} est un $mIDS$ d'un line graphe L_G si et seulement si il existe un couplage maximal de taille minimum $M_{mMM} \subseteq E$ de G tels*

que ses arêtes correspondent aux sommets de S_{mIDS}. Ainsi on a $|S_{mIDS}| = |M_{mMM}|$.

corollaire 4.3. *[32] Dans un line graphe L_G, le problème $mIDS$ est un problème approximable avec un facteur d'approximation de 2.*

Démonstration. Si on prend un graphe G=(V,E), $S_{IMS} \subseteq V$ un indépendant maximale, c'est-à-dire il n'existe pas d'indépendant strictement plus grand contenant S_{IMS}. Dans ce cas, S_{IMS} est aussi dominant sinon il existerait un sommet $s \in V$ tel que s n'est pas dominé, c'est-à-dire ni s ni aucun de ses voisins ne sont dans S_{IMS}. Ainsi $S_{IMS} \cup \{s\}$ est indépendant , ce qui absurde car S_{IMS} est maximal. Finalement, S_{IMS} est un indépendant dominant, que nous renommons S_{IDS}.
D'après le théorème 4.1, tout IDS dans le line graphe correspond un MM dans le graphe racine de la même taille ; il en est de même pour les $mIDS$ et mMM par le corolaire 4.2. Or que $|M_{MM}| \leq 2|M_{mMM}|$ pour tout graphe. Ainsi on en déduit que dans les line graphes $|SIDS| \leq 2|SmIDS|$. Comme la construction d'un indépendant maximal se fait en temps polynomial par un algorithme glouton, le problème est dans APX. De plus, nous avons montré que la procédure pour obtenir un line graphe à partir d'un graphe est simple. Le mMM se réduit donc de manière polynomiale au $mIDS$ et nous avons notre résultat. □

Par ces démonstrations, le problème du $mIDS$ a un rapport d'approximation de 2 dans les line graphes. Autrement dit, quel que soit un IDS dans un line graphe sa taille est au plus deux fois celle du $mIDS$. Malheureusement, le problème n'est pas approximable dans le cas général.

4.3 Heuristiques et méta-heuristiques

4.3.1 Heuristiques

Contrairement aux algorithmes exacts et aux algorithmes d'approximations, les heuristiques ne nécessitent pas d'analyses théoriques pour démontrer ou borner la qualité des solutions qu'elles retournent [32].
D'après ce qu'on a déjà vu, on ne connaît pas d'algorithmes polynomiaux pour les problèmes NP-difficiles, et la conjecture $P \neq NP$ fait qu'il est peu probable qu'il en existe. Or, de nombreux problèmes d'optimisation combinatoire sont NP-difficiles et ne pourront donc pas être résolus de manière exacte dans un temps raisonnable. Cependant certains travaux théoriques ont été réalisés et notamment [14] qui déterminent des conditions nécessaires

et suffisantes pour démontrer la convergence de certaines heuristiques vers les solutions optimales. Certaines d'entre elles sont aussi très efficaces et peuvent résoudre des problèmes de très grande taille.
Le mot heuristique, dérivé de la langue grec, vient du verbe heuriskein qui signifie « je trouve ». L'idée de tels algorithmes provient souvent de l'observation de la nature ou de la vie courante. On y distingue les heuristiques, les méta-heuristiques et plus récemment les hyperheuristiques. Cette méthode approximative, est un algorithme qui fournit rapidement (en temps polynomial) une solution réalisable, pas nécessairement optimale, pour un problème d'optimisation NP-difficile. On oppose les méthodes approchées aux méthodes exactes, qui trouvent toujours l'optimum si on leur en laisse le temps. L'usage d'une heuristique est efficace pour calculer une solution approchée d'un problème et ainsi accélère le processus de résolution exacte. Les approches heuristiques, contournent le problème de l'explosion combinatoire en faisant délibérément des impasses et n'explorent qu'une partie de l'espace des combinaisons. Ce type de méthodes traduit une stratégie en s'appuyant sur la connaissance du problème. Une heuristique est spécifique au problème et ne peut pas être généralisée, elles peuvent être classées en deux catégories :

1. Méthodes constructives qui génèrent des solutions à partir d'une solution initiale en essayant d'en ajouter petit à petit des éléments jusqu'à ce qu'une solution complète soit obtenue.
2. Méthodes de fouilles locales qui démarrent avec une solution initialement complète (probablement moins intéressante), et de manière répétitive essaie d'améliorer cette solution en explorant son voisinage.

4.3.2 Méta-heuristique

Les méta-heuristiques sont très utilisées de nos jours. On parle de méta-heuristique pour les méthodes approximatives générales, pouvant s'appliquer à différents problèmes.
Le mot méta-heuristique est composé de deux mots grecs : méta et heuristique. Le mot méta est un suffixe signifiant au-delà c'est-à-dire de niveau supérieur. En effet, ces algorithmes se commandent des méthodes génériques pouvant optimiser une large gamme de problèmes différents sans nécessiter de changements profonds dans l'algorithme employé, c'est une idée de schéma global applicable et adaptable à différents problèmes. En plus de posséder différents outils pour guider et améliorer la qualité des solutions. Les méta-heuristiques sont en général non-déterministes, elles peuvent ne pas trouver la solution optimale. On peut distinguer les méta-heuristiques qui font évoluer une seule solution sur l'espace de recherche à chaque itération et les

méta-heuristiques à base de population de solutions. En général, les méta-heuristiques à base de solution unique sont plutôt axées sur l'exploitation de l'espace de recherche, on n'est donc jamais sûr d'obtenir l'optimum optimorum. Les méta-heuristiques à base de population sont plutôt exploratoires et permettent une meilleure diversification de l'espace de recherche [17]. les méta-heuristiques peuvent combiner différentes heuristiques. Le travail consiste alors à bien maîtriser ces outils et les heuristiques pour savoir comment les combiner pour chaque type de problème. L'essaim particulaire, les colonies de fourmis, l'algorithme génétique ne sont que quelques méta-heuristiques connues parmi les nombreuses qui sont utilisées.

4.4 Algorithme efficace sur des classes des graphes particulières

Afin de pouvoir trouver des algorithmes plus rapides que dans le cas général, on utilise les structures et les propriétés particulières des classes de graphe. D'une certaine manière, les restrictions définies sur les familles de graphes permettent de simplifier les problèmes.

Ainsi, à chaque fois que l'on détermine un certain nombre de particularités, nous permettant de définir une famille de graphes plus restreinte que dans le cas général, nous avons une chance d'avoir, dans le même temps, défini une famille de graphes sur laquelle un problème difficile habituellement devient beaucoup plus simple.

Considérons le problème d'ensemble dominant de taille minimum. Dans le cas général, ce problème est un problème NP-complet. Le meilleur algorithme à ce jour pour le résoudre est celui de Y. Iwata [18]. Supposons maintenant que l'on choisisse de résoudre ce problème sur une famille de graphes dont une propriété particulière est que tous les graphes de cette famille sont des cliques. Dans ce cas, chaque sommet dominant l'intégralité du graphe, le problème devient excessivement simple, et même trivial. Bien que la famille des graphes de cet exemple soit vraiment trop restreinte pour rendre l'étude de ce problème intéressante, elle nous permet de montrer que, lorsque les restrictions sont suffisantes, un problème qui est NP-complet sur le cas général peut devenir solvable en temps polynomial.

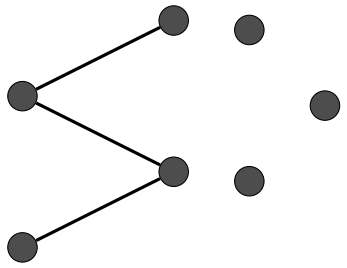
Il existe dans la littérature beaucoup de documents traitant de la complexité d'un problème donné sur une classe de graphe particulière. on citera dans cette section, certains résultats de problème de stable maximum et de problème de domination.

Le problème de stable maximum, est un des problèmes NP-complets les

plus connus et les plus étudiés. Selon la classe de graphes considérée, il peut devenir un problème solvable en temps polynomial.

En 2002 V.E. Alekseev [1] à pu démontrer que dans un graphe sans $S_{1,1,2}$, ce problème peut être résolu en temps polynomial. Un graphe $S_{i,j,k}$ constitué d'un sommet central qui serait le point de départ de trois chemins de longueurs respectivement i, j et k.

J.François Couturier à démontrer dans sa thèse [19] que sur les graphes sans $P_4 + sP_1$, le problème d'ensemble stable peut être aussi résolu en temps polynomial, tel que le graphe $P_4 + sP_1$ est un graphe constitué d'un chemin de longueur 4, associé à une collection de s sommets isolés. C'est-à-dire une union disjointe entre un P_4 et un ensemble stable de taille s.



$P_4 + 3P_1$

FIGURE 4.8

Sur les graphe sans cycle ou les graphes triangulés ce problème peut-être résolu en temps linéaire [15], il est de même pour les graphes d'intervalles, car d'après le théorème de Gilmore et Hoffman chaque graphe d'intervalle est un graphe triangulé [30]. Ainsi le problème de stable maximum est aussi linéaire sur les graphes d'intervalles.
V.V. Lozin et R. Mosca, on pu aussi démontrer en 2005 dans [22] que ce problème peut être résolu en temps polynomial si le graphe ne possède pas de sous-graphe induit $K_{1,3} + P_2$.

Nous allons maintenant nous intéresser au problème de l'ensemble dominant. Ce problème est bien connu pour être NP-complet dans le cas général. Voyons maintenant comment ce problème se comporte sur les différentes classes de graphes.
Les graphes sans P_4 est un graphe pour lequel il n'y a pas de sous-graphe induit qui soit un chemin de quatre sommets. Sur cette classe de graphes, le problème de domination peut être résolu en temps polynomial. De même sur les graphes sans $P_4 + sP_1$ [19].

Ces constatations nous amènent alors à une série de questions naturelles : où se trouve la limite à partir de laquelle un problème n'est plus NP-difficile ?

Ou encore, pourrait-on, grâce aux classes de graphes, définir une sorte de frontière entre l'ensemble des problèmes déterministes polynomiaux et l'ensemble des problèmes non-déterministes polynomiaux ?

CHAPITRE 5

RESTRICTION DE PROBLÈME DIFFICILE SUR UNE CLASSE DE GRAPHES

5.1 Problème du stable maximum dans un graphe triangulé

Les problèmes de stable maximum et de partition minimum en cliques des sommets sont parmi les problèmes très connus pour être dans la classe NP-complet ; il est aussi connu que grâce à l'algorithme LexBFS, ces problèmes peuvent être résolus en un temps linéaire quand ils sont considérés sur les graphes triangulés. Rappelons que le nombre d'éléments dans un stable maximum d'un graphe G est noté $\alpha(G)$, et le nombre de cliques dans une partition minimum des sommets en cliques est noté $\theta(G)$. Il est facile de voir que pour un graphe G quelconque

$$\alpha(G) \leq \theta(G)$$

et si G est triangulé on a

$$\alpha(G) = \theta(G)$$

La version décision du problème du stable est
Problème de décision associé au problème du stable
Entrée : Un graphe $G = (V, E)$ et un entier k.
Question : Existe-t-il une stable de taille au moins k dans G ?

Théorème 5.1. *[25] Le problème du stable maximum est NP-complet.*

Théorème 5.2. *[34] Soit G un graphe. Les trois conditions suivantes sont équivalente*

1. *G est triangulé*
2. *G admet un ordre d'élimination simplicial*
3. *Tout ordre LexBFS sur G est un ordre d'élimination simplicial.*

Lemme 5.3. *Si x est un sommet simplicial et S est un stable maximum, alors*

$$|S \cap N[x]| = 1$$

Démonstration. Puisque x est simplicial, $N[x]$ est une clique et donc S et $N[x]$ ont au plus un sommet en commun. Par l'absurde, supposons que $|S \cap N[x]| = 0$. Mais ceci implique que $S \cup N[x]$ est un stable, et ceci est une contradiction avec le fait que S est maximum. □

Lemme 5.4. *Si x est un sommet simplicial dans un graphe $G = (V, E)$, alors*

$$\alpha(G) = \alpha(G - N[x]) + 1$$

Démonstration. Soit S un stable maximum dans G. Alors $S \cap (V - N[x])$ est un stable dans G', le sous graphe induit par $V - N[x]$. On a

$$S \cap (V - N[x]) = S - S \cap N[x] \Rightarrow |S - S \cap N[x]| = |S| - |S \cap N[x]| \leq \alpha(G')$$

c'est à dire

$$\alpha(G) - 1 \leq \alpha(G')$$

par conséquent

$$\alpha(G) \leq \alpha(G') + 1$$

Inversement, soit S' un stable maximum dans le sous graphe G' induit par $V - N[x]$. Par définition de G', x n'a aucun voisin dans S', donc $S' \cup \{x\}$ est un stable dans G. On en déduit que

$$|S' \cup \{x\}| = |S'| + 1 = \alpha(G') + 1 \leq \alpha(G)$$

D'où

$$\alpha(G) = \alpha(G') + 1$$

□

Définition 5.1. Soit v un sommet dans un graphe G ordonné par LexBFS σ.

1. On appelle voisins supérieurs de v par rapport à σ, l'ensemble

$$N^+(v) = \{w \in V, v < w, vw \in E\}$$

2. La restriction de LexBFS σ sur un sous ensemble de sommets A est un LexBFS si $\forall u, v \in A$,

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

Lemme 5.5. *Soit G un graphe triangulé ordonné par LexBFS $\sigma = (v_1, v_2...v_n)$. Alors la restriction de σ sur le sous graphe induit par $V - N[v_1]$ est un LexBFS.*

Démonstration. Soient $u, v \in V' = V - N[v_1]$. Par définition, puisque σ est un LexBFS, alors

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

et ceci implique

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

Par conséquent, σ est un LexBFS sur $V - N[v_1]$. □

Les lemmes 5.4 et 5.5 conduisent à un algorithme linéaire de calcul d'un stable maximum dans la classe des graphes triangulé.

Algorithme 5.1. *(Algorithme glouton pour le stable maximum)*
Entrée : *Un graphe triangulé et un ordre LexBFS.*
Sortie : *Un stable S de cardinalité maximale.*

1. *Début*
2. $W \leftarrow V$; $S \leftarrow \emptyset$;
3. *Tant que* $W \neq \emptyset$ *faire ;*
4. $v = \min W$, $S \leftarrow S \cup \{v\}$, $W \leftarrow W - N^+[v]$// $N^+[v] = N^+(v) \cup \{v\}$.
5. *Fin tant que*
6. *Afficher S*
7. *Fin*

Remarque 5.1. Notons que si $S = \{s_1, s_2...s_k\}$ l'ensemble des sommets renvoyé par l'algorithme ci-dessus, où $s_k = \min W$ est le sommet calculé à la dernière étape, alors $W = N^+[s_k]$, sinon s_k ne serait pas le dernier élément.

Théorème 5.6. *L'algorithme 5.1 calcul un stable maximum en $O(n)$ dans un graphe triangulé, avec n est le nombre de sommets.*

Démonstration.

1. **La correction** Il est claire qu'à chaque itération, le sommet v rajouté à l'ensemble S n'a aucun voisin dans S, donc S est un stable. D'après les lemmes 5.4 et 5.5,

$$\alpha(G) = k + \alpha\Big(G - \cup_1^k N^+[s_i]\Big) = k + 0 = k$$

car d'après la remarque 5.1, $V = \cup_1^k N^+[s_i]$. Par conséquent, S est maximum.

2. **La complexité** Chaque sommet est examiné une seule fois, soit pour le mettre dans S ou pour le supprimer. Par conséquent, la complexité temporelle est en $O(n)$.

□

5.1.1 Exemple d'application

L'institut Pasteur doit stocker 13 vaccins dans des réfrigérateurs. Chaque vaccin à une température de conservation. Et chaque réfrigérateur devra être réglé à une température précise. Il s'agit pour l'institut de stocker les vaccins dans un nombre minimum de réfrigérateurs.
Quel est le nombre minimum des réfrigérateurs pour stocker ces vaccins ?

A : vaccin BCG [-11, 8]
B : vaccin Hépatite [0, 8]
C : vaccin Rougeole [12, 40]
D : vaccin Pfizer-BIONT [0, 7]
E : vaccin Johnson & johnson [-80, -70]
F : vaccin AstraZeneca [-20, -11]
G : vaccin mRNA-1273 [-20, -15]
H : vaccin contre la rage [-71, -30]
I : vaccin Grippe saisonnière [2, 8]
J : vaccin BEXSERO [-31, -18]
K : vaccin ERVEBO [9, 10]
L : vaccin HEXYON [4, 13]
M : vaccin dcaT-VPI [0, 6]

Pour trouver le nombre minimum de réfrigérateurs, on associe le graphe G comme suit : à chaque vaccin on lui associe un sommet, et deux sommets sont adjacents si et seulement si les vaccins peuvent être mis dans

un même réfrigérateurs. Autrement dit, si les deux intervalles associés à ces deux sommets s'intersectent. Ainsi, les vaccins qui se trouvent dans un même réfrigérateurs sont deux à deux adjacents, c'est à dire forment une clique, et le nombre minimum de réfrigérateurs est donné par une partition minimum des sommets en clique. Le graphe (a) de la figure 5.1 illustre cette modélisation.

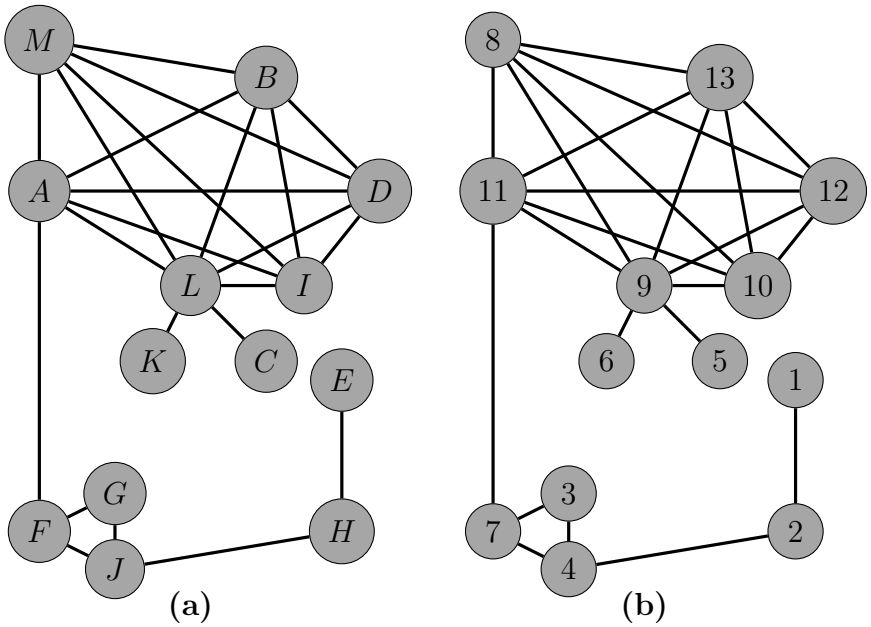
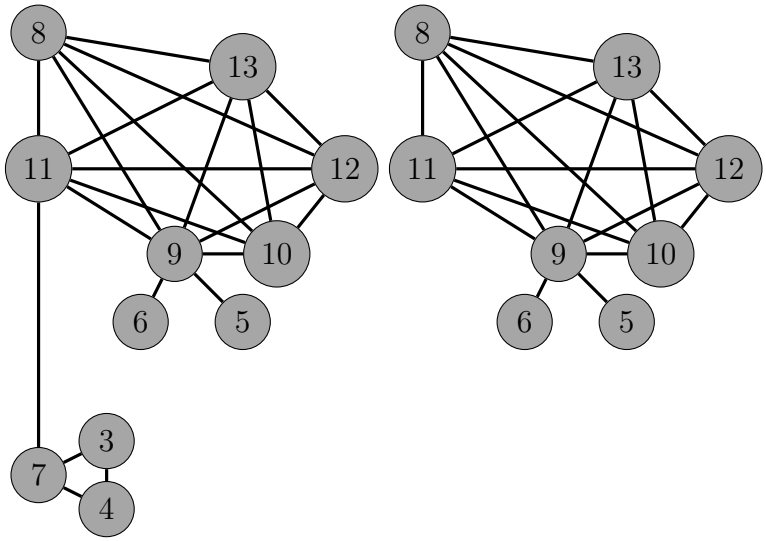


FIGURE 5.1 – L'ordre LexBFS sur les sommets de G

Par définition, le graphe G est d'intervalle, donc triangulé. Il en résulte que le nombre minimum de cliques est égale au nombre de sommets dans un stable maximum. Ci-dessous, nous appliquons l'algorithme 5.1 pour calculer le nombre de stabilité $\alpha(G)$ et donc le nombre minimum de réfrigérateurs.

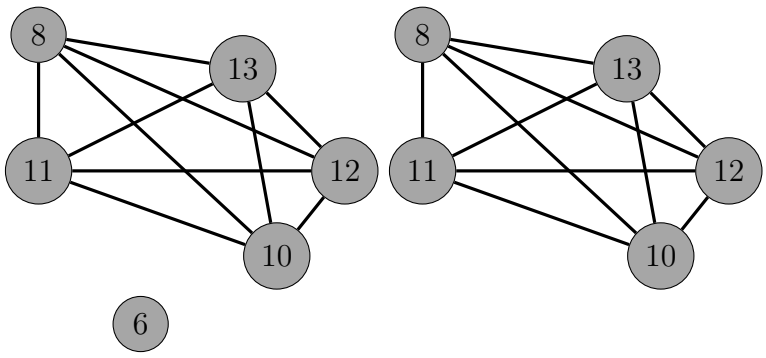
Après l'application de l'algorithme de LexBFS sur le graphe G, on obtiendra l'énumération comme il est indiqué dans le graphe (b) la figure 5.1.

1. **Initialisation :** $S = \emptyset$, $W = V$;
2. **Itération 1 :** $W \neq \emptyset$, $\min W = \min V = 1$; $S = \{1\}$, $W \leftarrow V - N[1] = \{3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\}$.
3. **Itération 2 :** $W \neq \emptyset$, $\min W = 3$; $S = \{1, 3\}$, $W \leftarrow V - N^+[3] = \{5, 6, 8, 9, 10, 11, 12, 13\}$.
4. **Itération 3 :** $W \neq \emptyset$, $\min W = 5$; $S = \{1, 3, 5\}$, $W \leftarrow V - N^+[5] = \{6, 8, 10, 11, 12, 13\}$.



e_1 e_2

5. **Itération 4 :** $W \neq \emptyset$, $\min W = 6$; $S = \{1, 3, 5, 6\}$,
   $W \leftarrow V - N^+[6] = \{8, 10, 11, 12, 13\}$.



e_3 e_4

6. **Itération 5 :** $W \neq \emptyset$, $\min W = 8$; $S = \{1, 3, 5, 6, 8\}$,
   $W \leftarrow V - N^+[8] = \emptyset$.
7. **Itération 6 :** $W = \emptyset$, alors $S = \{1, 3, 5, 6, 8\}$ est maximum.

Par conséquent, la conservation des vaccins se fait comme suit : Dans le premier réfrigérateur, on met les vaccins $\{1, 2\}$=$\{$johnson, le vaccin contre la rage$\}$; dans le deuxième, on met les vaccins $\{4, 3, 7\}$=$\{$vaccin mRNA-1273, vaccin AstraZeneca, vaccin BEXSERO$\}$; le troisième réfrigérateur contient les vaccins $\{5, 9\}$=$\{$vaccin Rougeole, vaccin HEXYON$\}$; le quatrième réfrigérateur contient le vaccin $\{6\}$=$\{$ vaccin ERVEBO$\}$; le cinquième réfrigérateur contient les vaccins $\{8, 10, 11, 12, 13\}$=$\{$vaccin dcaT-VPI, vaccin Grippe saisonniere, vaccin BCG, vaccin Hepatite, vaccin Pfizer-BIONT$\}$.

5.2 Colorer un graphe triangulé

Une coloration propre d'un graphe $G = (V, E)$ est une application c sur l'ensemble V à valeurs dans l'ensemble $\mathcal{C} = \{c_1, c_2.....\}$ tel que deux sommets adjacents ne doivent pas recevoir la même couleur. Autrement dit

$$uv \in E \Leftrightarrow c(u) \neq c(v)$$

; le nombre chromatique du graphe G est noté $\chi(G)$, il est donné par le nombre minimum de couleurs dans une k-coloration propre. Colorer un graphe par k couleurs c'est équivalent à partitionner ses sommets en k stables.

Remarque 5.2. On peut représenter les couleurs à l'aide des entiers de l'ensemble $\{1, 2, 3, ..., k\}$ tel que l'entier 1 représente la première couleur, 2 la deuxième couleur, ainsi de suite.

Problème de coloration
Entrée : Un graphe $G = (V, E)$ et un entier k.
Question : Existe-t-il une coloration de G utilisant au plus k couleurs ?
Le problème de coloration est l'un des premiers problèmes qui a été démontré NP-complet.

Théorème 5.7. *[25] Le problème de coloration est NP-complet.*

Pour calculer une coloration optimale dans un graphe triangulé, on applique l'algorithme de coloration gloutonne à l'ordre inverse de LexBFS. Une **coloration Gloutonne** d'un graphe est un algorithme qui choisit à chaque itération un sommet non coloré et lui attribuer la plus petite couleur non utilisé par ses voisins déjà colorés.

5.2.1 Exemple d'application

Une entreprise qui fabrique 12 sortes de produits chimiques différents (notés P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}, P_{12}) doit en assurer le transport par train. Ces produit sont en petite quantité, mais ne peuvent être tous placés dans le même wagon pour des raisons de sécurité (le contact entre certains de ces produits peut provoquer des réactions explosives). Plus précisément :
P_1 ne peut être transporté avec P_2, P_3 ou P_4
P_2 ne peut être transporté avec P_1, P_3 ou P_5
P_3 ne peut être transporté avec P_1, P_2 ou P_4
P_5 ne peut être transporté avec P_2, P_6 ou P_8
P_6 ne peut être transporté avec P_5, P_{11} ou P_{12}
P_7 ne peut être transporté avec P_8, P_9 ou P_{10}
P_8 ne peut être transporté avec P_5, P_7 ou P_{10}
P_{10} ne peut être transporté avec P_7, P_8, P_{11} ou P_{12}
P_{11} ne peut être transporté avec P_6, P_{10} ou P_{12}
Combien de wagons sont ils nécessaires au transport des 12 produits ?

Pour trouver le nombre nécessaires de wagons, on doit d'abord modéliser ce problème sous forme d'un graphe G, tel que à chaque produit on lui associe un sommet, et deux sommets sont adjacent si et seulement si les deux produits chimiques correspondent a ces deux sommets sont incompatibles. Autrement dit les deux produits peuvent pas être dans le même wagon. Le graphe de la figure 5.2 illustre ce processus.

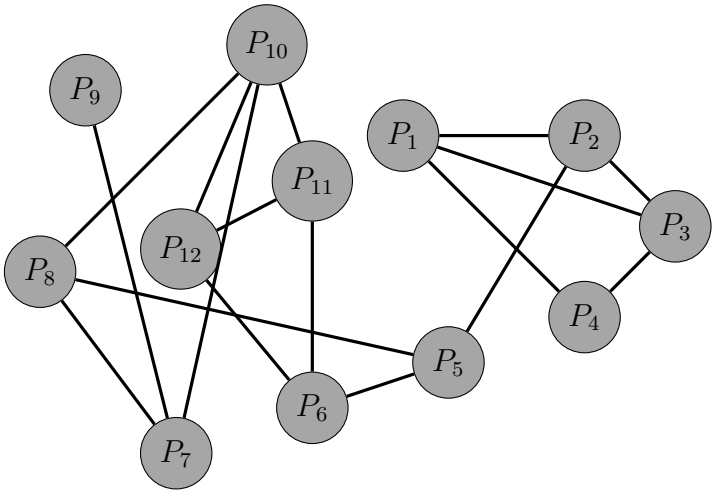


FIGURE 5.2

Ainsi donc on cherchera une partition minimum en stable, tel que les sommets de chaque stable correspond aux produits qui peuvent être transporté dans le même wagon.
Pour trouver cette partition en stable, on appliquera l'algorithme de coloration Gloutonne, sur l'ordre inverse de lexBFS.

Application de l'algorithme lexBFS

Après l'application de l'algorithme lexBFS on obtiendra l'énumération suivante :

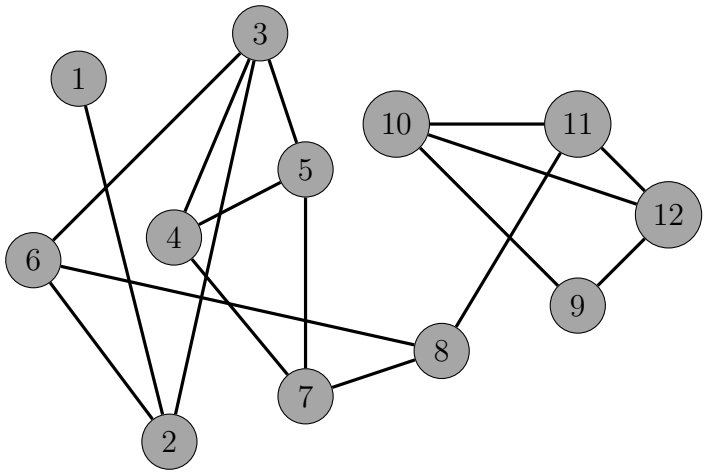
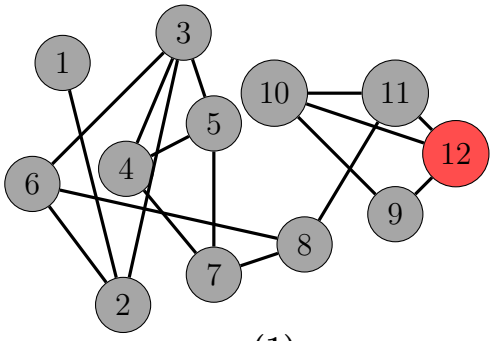


FIGURE 5.3 – Graphe ordonné par l'algorithme lexBFS

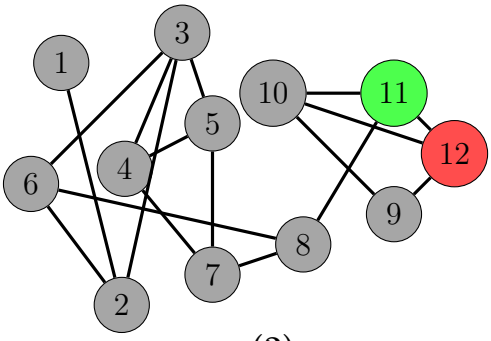
Il est remarquable que ce graphe ne contient pas de trou, et il admet un ordre d'élimination simpliciale. On conclu alors qu'il est triangulé.
Après avoir ordonné tous les sommets de ce graphe par l'algorithme lexBFS, on passera a l'étape de coloration de gloutonne.

Application de l'algorithme de coloration Gloutonne

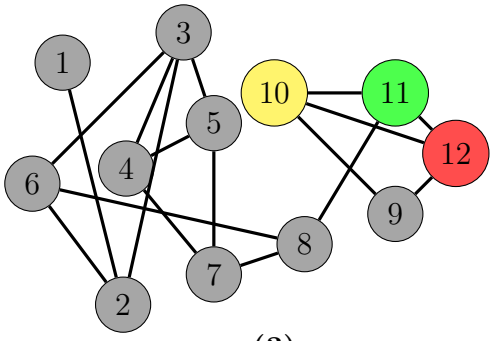
Dans cette sous-section nous allons appliqué l'algorithme de coloration glouton sur l'ordre inverse fourni par LexBFS. Le graphe **(12)** de la figure 5.4 montre le résultat de cette coloration gloutonne.



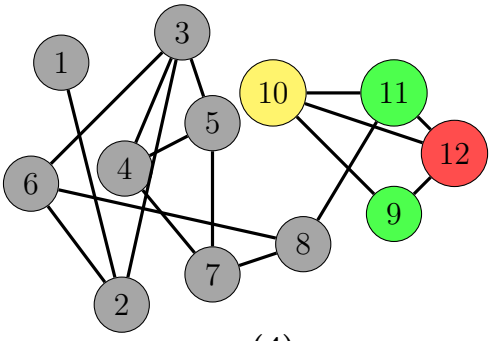
(1)



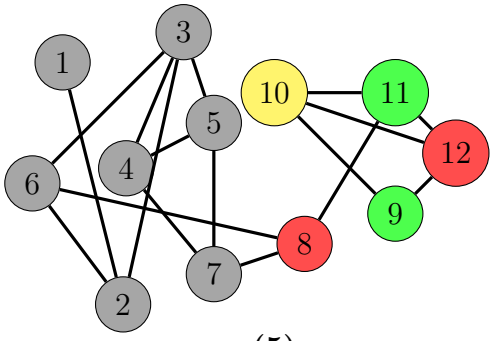
(2)



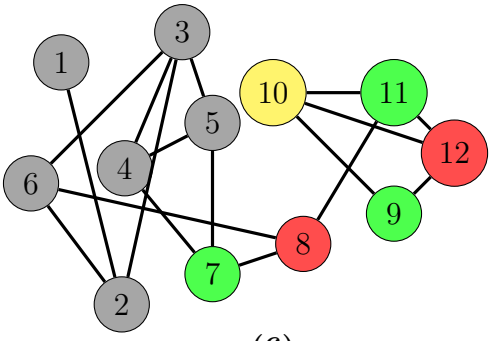
(3)



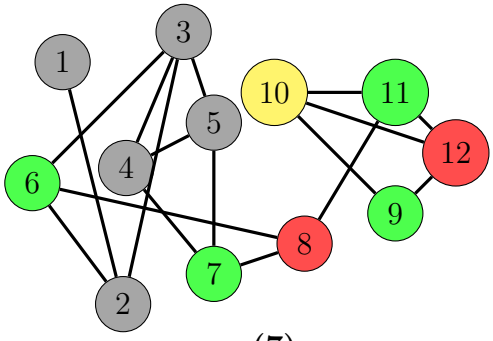
(4)



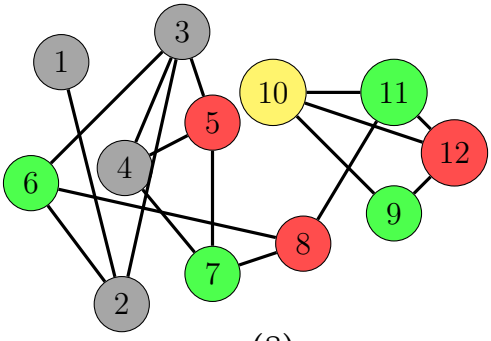
(5)



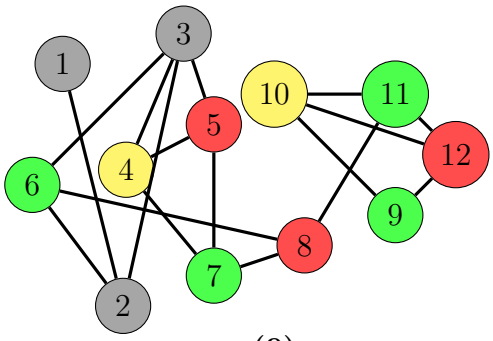
(6)



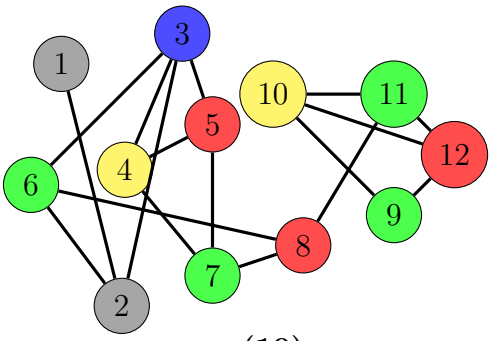
(7)



(8)



(9)



(10)

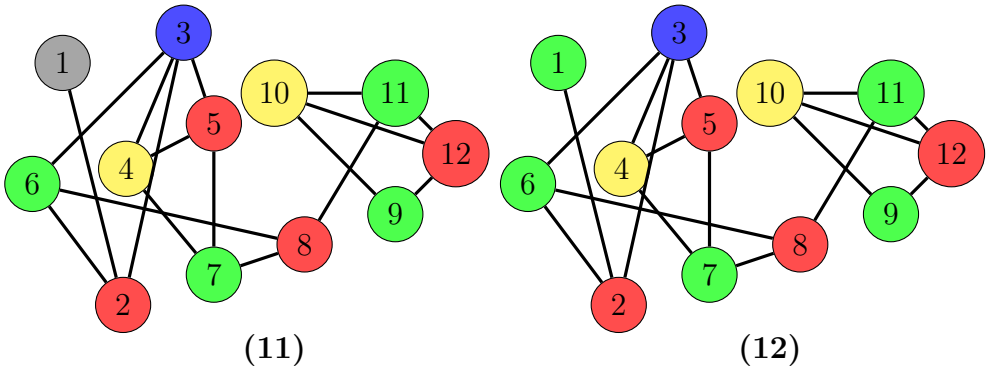


FIGURE 5.4 – Graphe coloré par l'algorithme de coloration Gloutonne

Le nombre minimum de couleur nécessaire pour colorer ce graphe est bien 4. Ainsi, le nombre de stable qui le partitionne est 4. Les sommets de chaque stable correspondent aux produits chimiques qui peuvent être transporté dans le même wagon. On conclu donc que le nombre nécessaire de wagons qui va assuré cette transportation est 4 tel que :

1. Le premier wagon transportera 4 produits qui correspondent à : $\{p_3, p_5, p_{11}, p_7\}$
2. Le deuxième wagon 5 qui correspondent à : $\{p_2, p_4, p_6, p_8, p_9\}$
3. Le troisième wagon 2 produits qui correspondent à : $\{p_1, p_{12}\}$
4. Le quatrième wagon 1 qui correspondent à : $\{p_{10}\}$.

CONCLUSION GÉNÉRALE

Le travail réalisé dans ce mémoire présente une vision globale sur la théorie des graphes qui englobe un sujet très intéressant, la résolution des problèmes difficiles. Les graphes constituent l'un des instruments les plus courants et les plus efficaces pour représenter les problèmes discrets, posés en recherche opérationnel.

Notre travail est essentiellement motivé par la notion de la complexité et de la réduction polynomiale pour prouver la NP-complétude de certains problèmes. Nous avons, dans le premier chapitre aborder un aperçu générale sur les notions de base de la théorie des graphes. Dans le second et le troisième chapitre sont consacrés respectivement à la notion de complexité et la réduction polynomiale dont le but de prouver que les problèmes de décision peuvent être classés dans des classes selon leurs difficultés intrinsèque. Le chapitre 4 présente les différentes méthodes utilisées pour résoudre un problème difficile, à savoir, garder le caractère d'exactitude de la solution optimale en essayant d'améliorer la complexité des algorithmes existant ou en s'orientant vers des classes de graphes pour confectionner des algorithmes efficaces, ou bien renoncer à la solution exacte et se diriger vers des algorithmes d'approximation polynomiaux ou des heuristiques méta-heuristiques..... Enfin, dans le dernier chapitre, nous avons présenté deux exemples de problèmes concrets que nous avons modélisé par des graphes triangulés et chercher ensuite les solutions optimales par des algorithmes efficaces.

BIBLIOGRAPHIE

[1] V. Alekseev, Polynomial algorithm for finding the largest independent sets in graphs without forks, Discrete Applied Mathematics,135 (2004), page [3 – 16].

[2] Anne Benoit ;Benjamin Depardon, Christophe Mouilleron, Clément Resvoy Algorithmique I - Cours et Travaux Dirigés L3, École Normale Supérieure de Lyon 2009 [pages 103, 104]

[3] G. Ausiello, P. Crescenzi, V. Kann, Marchetti-Sp, Giorgio Gambosi et Alberto M. Spaccamela. Complexity and approximation : Combinatorial optimization problems and their approximability properties. Springer, jan 2000.

[4] Björklund A., T. Husfeldt, Inclusion-exclusion algorithms for counting set partitions, Proceedings of the 47th IEEE Symposium on Foundations of Computer Science, FOCS 2006 , p. 575–582. [p. 39, 41, 43, 54, 55, 67, 126, 175]

[5] A. Björklund, T. Husfeldt, and M. Koivisto, Set partitioning via inclusion-exclusion, SIAM Journal on Computing, 39 (2009), pp. 546–563.

[6] J.A. Bondy et U.S.R. Murty ;Traduit de l'anglais par F. Havet Théorie des Graphes 2008 p.202[pages 184, 193, 202]

[7] Bruno, Escoffier ; Giorgio, Ausiello ; Approximation polynomiale de problèmes d'optimisation : aspects structurels et opérationnels Université Paris IX UFR Sciences des Organisations ,France, thèse 2005 [pages 27, 28, 29, 31].

[8] Cook, S., the Complexity of Theorem Proving Procedures, Proceedings of the third Annual ACM Symposium on Theory of Computing, ACM, New York, 2570, p. 151–158

[9] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, Introduction To Algorithms, MIT Press, 2001.

[10] Davis,M., G. Logemann, D. Loveland, A machine program for theorem-proving, Communications of the ACM 5 (1962), p. 394–397. [p. 4, 39, 41, 44]

[11] Davis,M., H. Putnam, A computing procedure for quantification theory, Journal of the ACM 7 (1960), p. 201–215. [p. 4, 39, 41]

[12] Dynamic programming meets the principle of inclusion and exclusion, Operations Research Letters, 1 (1982), pp. 49 – 51.

[13] Eglantine, Camby ; Différents problèmes en théorie des graphes université de bruxellex ; Bruxelles 136.

[14] Gérard Fleury. Méthodes stochastiques et déterministes pour les problèmes NP-difficile. Thèse Université Blaise Pascal, 1993. Page 13.

[15] M. C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, vol. 57 of Annals of Discrete Mathematics, Elsevier, 2004.

[16] M. Held and R. Karp, A dynamic programming approach to sequencing problems, SIAM Journal on Applied Mathematics, 10 (1962), pp. 196–210.

[17] Ilhem, Boussaid ;Perfectionnement de métaheuristiques pour l'optimisation continue,L'université des sciences et de la téchnologie HOUARI BOUMEDIENE thèse Algérie 2013 [pages 8,9].

[18] Y. Iwata, A faster algorithm for dominating set analyzed by the potential method, in IPEC, 2011, pages [41–54].

[19] Jean-François, Couturier ; Dieter, Kratsch ; Algorithmes exacts et exponentiels sur les graphes : énumération, comptage et optimisation, l'Université de Lorraine, thèse, France 2012, pages [17 - 122].

[20] Koivisto, M., An $O^*(2^n)$ Algorithm for Graph Coloring and Other Partitioning Problems via Inclusion-Exclusion, Proceedings of the 47th IEEE Symposium on Foundations of Computer Science, FOCS 2006 , p. 583–590. [p. 39, 41, 43, 54, 55, 67]

[21] Lionel, Lagarde ; Société française de recherche opérationnelle ; France 7,8

[22] V. V. Lozin and R. Mosca, Independent sets in extensions of $2k_2$-free graphs, Discrete Applied Mathematics, 146 (2005), page [74–80].

[23] Magnús M. Halldórsson. Approximating the Minimum Maximal Independence Number. Inf. Process. Lett., vol. 46, no. 4, pages 169-172, 1993 [pages 20 et 22].

[24] Mathieu, Liedloff ; Dieter, Kratsch ; Algorithmes exacts et exponentiels pour les problèmes NP-difficiles :domination, variantes et généralisations, Université Paul Verlaine, thèse, Metz 2007, [pages 11, 12, 13, 14, 38, 40, 41, 42, 51, 54, 55]

[25] Michael R. Garey et David S. Johnson. Computers and intractability : A guide to the theory of NP-completeness (series of books in the mathematical sciences). W.H. Freeman Co Ltd, first edition, Janvier 1979 [pages 8, 20 et 102].

[26] Mihalis Yannakakis et Fanica Gavril. Edge dominating sets in graphs. SIAM J. Appl. Math., vol. 38, pages 364-372, 1980. (Cité en pages 11, 22 et 26.)

[27] Mustapha, kchikech ; Introduction à la Complexité Algorithmique 2016 Université Cadi Ayyad ; [pages 8,9]

[28] Mustapha,Kchikech ; Introduction à la théorie des graphes, Université Cadi Ayyad, Maroc 2018 [pages 5,6,11]

[29] J. Nederlof, Fast polynomial-space algorithms using möbius inversion : Improving on steiner tree and related problems, in ICALP (1), vol. 5555 of Lecture Notes in Computer Science, Springer, 2009, pp. 713–725.

[30] Paul C. Gilmore et Alan J. Hoffman, A characterization of comparability graphs and of interval graphs , Canad. J. Math, vol. 16, n^{os} 539 - 548, 1964, page 4.

[31] Press, W H., B.P. Flannery, S.A. Teukolsky,Numerical Recipes in C : The Art of Scientific Computing. 2^e édition. Cambridge University Press, 1992. [p. 45]

[32] Raksmey,Phan ; Pascal,Berthomé ; Méthodes exactes et approchées par partition en cliques de graphes ; Université Blaise Pascal école doctorale Spisciences pour l'ingénieur, thèse France 2013 [pages 10, 11, 13, 14, 22, 24, 25, 26].

[33] Rémi, Watrigant ; Frédéric, Havet ; Vangelis, Th. Paschos ; Approximation et complexité paramétrée de problèmes d'optimisation dans les graphes : partitions et sous-graphes l'Université Montpellier II thése France 2014 [pages 34, 36].

[34] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. SIAM J. Comput., 5(2) :266–283, 1976

[35] Woeginger, G.J., Exact algorithms for NP-hard problems : A survey, Combinatorial Optimization - Eureka, You Shrink !, LNCS 2570, (2003), p. 185–207. [p. 3, 12, 30, 38, 41, 42, 126, 136, 173]

على تقدير نماذج الانحدار الذاتي

في هذا العمل نحن مهتمون بالمشكلات الصعبة في نظرية الرسم البياني. أولاً ، قمنا بتعريف مفهوم التعقيد وفئات المشكلات ، ولا سيما فئة ما يسمى بالمشكلات الصعبة. ثم ركزنا على حل هذه المشاكل. لكن ، لسوء الحظ ، لا نعرف كيف نبني خوارزميات يمكنها حلها تمامًا وفي وقت متعدد الحدود ، ولهذا السبب اقترحنا بعض الأساليب حتى نتمكن من التعامل مع هذا النوع من المشاكل.

Résumé

Dans ce travail nous nous sommes intéressés au problèmes difficiles dans la théorie des graphes. En premier lieu nous avons défini la notion de complexité et les classes des problèmes, notamment la classe des problèmes dites difficiles. Ensuite nous nous sommes intéressées à la résolution de ces problèmes. Mais, malheureusement on sait pas construire des algorithmes qui puisse les résoudre de façon exacte et en temps polynomial, c'est pour cela nous avons proposé certaines approches afin de pouvoir traiter ce genre de problèmes.

Abstract

In this work we are interested in difficult problems in graph theory. First, we defined the notion of complexity and the classes of problems, in particular the class of so-called difficult problems. Then we focused on solving these problems. But, unfortunately, we don't know how to build algorithms that can solve them exactly and in polynomial time, that's why we have suggested some approaches in order to be able to deal with this kind of problem.