République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITÉ ABDERAHMANE MIRA BEJAIA
Faculté des Sciences Exactes
Département de Recherche Opérationnelle

Mémoire De Fin De Cycle
en vue de l'obtention du diplôme de
MASTER EN RECHERCHE OPERATIONNELLE
Option : Modélisation Mathématique et Evaluation de Performance des Réseaux

Thème

CALCUL D'INVARIANTS DANS LES GRAPHES PARFAITS

Présenté par :
M^r BELKHIRI Akram
M^r KHEMICI Mohamed

Dirigé par :
M$_r$ TALEM Djamel

Jury Composé de :

Président :	D^r	**KABYL Kamal**
Examinateurs :	M^r	**TAOUINET Smail**
Examinateurs :	M^r	**BOUTEBEL Abdedjalil**

Année Universitaire 2017-2018

Remerciements

Nous tenons à remercier :

Le bon Dieu de nous avoir donné la patience et la volonté pour accomplir ce travail,

Nos remerciements s'adressent également à :

Notre promoteur Mr TALEM pour ses conseils, ses orientations pour nous avoir transmis les renseignements nécessaires à la réalisation de ce travail, et son aide durant l'encadrement.

Nous remercions également :

Les membres de jury, pour l'honneur qu'ils nous font en acceptant de juger de lire et d'évaluer ce mémoire.

Nous tenons également à remercier

Tous les enseignants de notre département qui nous ont accompagnés au cours de notre formation et à tout le personnel de la bibliothèque de l'université.

Enfin, nous remercions toute personne ayant contribué de près ou de loin à la réalisation de ce travail.

Je dédie modeste travail à :

Mon trés cher père ;

Ma trés cher mère ;

Mes neveux : Massa, Taeb, Minou et Maram ;

Toute ma famille ;

Tout mes amis ;

Akram

Je dédie modeste travail à :

Mon trés cher père ;

Ma trés cher mère ;

Mes soeurs ;

Mon trés cher frère : Aissa ;

Toute ma famille ;

Tout mes amis ;

Mohamed

TABLE DES MATIÈRES

Introduction Générale **3**

1 DEFINITIONS ET NOTATION DE BASE **5**

1.1 Définitions et notations 5

1.2 Quelques graphes remarquables 8

1.3 Exploration des graphes 9

1.3.1 Parcours 10

1.3.2 Structure de données 10

1.3.3 Parcours en profondeur DFS (Depth-First Search) 11

1.3.4 Parcours en largeur BFS (Breadth-First Search) 13

2 complexité **16**

2.1 La notion de la complexité algorithmique 16

2.1.1 Introduction 16

2.1.2 Problèmes de décisions et Problèmes d'optimisations 17

2.1.3 Complexité d'un algorithme 20

2.1.4 Classes des problèmes P et NP 22

2.1.5 Classe NP-complet 23

3 Les graphes parfaits **27**

3.1 Graphe triangulé 29

3.2 Graphe d'intervalle 31

3.3 Graphe de comparabilité 32

4 Quelques problèmes d'invariants **34**

4.1 Problème de couplage dans un graphe biparti 34

4.2 Coloration d'un graphe triangulé 40

5 Applications **46**

5.1 Planification d'une formation 46

5.2 Problème d'affectation 49

Conclusion Générale **54**

TABLE DES FIGURES

1.1 Graphe non orienté 6
1.2 Corde 7
1.3 Stable et clique 7
1.4 (a) $K_{3,2}$, (b) K_4 8
1.5 Graphe non connexe 9
1.6 Graphe adjoint 9
1.7 Graphe quelconque DFS 12
1.8 Graphe quelconque BFS 14

2.1 26

3.1 Le nombre chromatique 28
3.2 Graphe triangulé 29
3.3 Exécution de Lex-BFS 31
3.4 La représentation d'un graphe d'intervalles 32
3.5 Graphe de comparabilité 33

4.1 Couplage parfait et transversal minimum 35
4.2 Algorithme *COLOR* 40
4.3 Graphe G 42
4.4 Graphe G coloré 43
4.5 Une coronne à 8 sommets 43
4.6 Graphe triangulé G 44
4.7 Graphe colorée 45

4.8 Graphe $\bar{P}_6$ 45

5.1 Modélisation de la formation 47
5.2 L'ordre des sommets de G 48
5.3 Graphe triangulé 48
5.4 Graphe biparti 51
5.5 Couplage minimum 53

Remerciements

Nous tenons à remercier :

Le bon Dieu de nous avoir donné la patience et la volonté pour accomplir ce travail,

Nos remerciements s'adressent également à :

Notre promoteur Mr TALEM pour ses conseils, ses orientations pour nous avoir transmis les renseignements nécessaires à la réalisation de ce travail,

et son aide durant l'encadrement.

Nous remercions également :

Les membres de jury, pour l'honneur qu'ils nous font en acceptant de juger de lire et d'évaluer ce mémoire.

Nous tenons également à remercier

Tous les enseignants de notre département qui nous ont accompagnés au cours de notre formation et à tout le personnel de la bibliothèque de l'université.

Enfin, nous remercions toute personne ayant contribué de près ou de loin à la réalisation de ce travail.

Je dédie modeste travail à :

Mon trés cher père ;

Ma trés cher mère ;

Mes neveux : Massa, Taeb, Minou et Maram ;

Toute ma famille ;

Tout mes amis ;

Akram

Je dédie modeste travail à :

Mon trés cher père ;

Ma trés cher mère ;

Mes soeurs ;

Mon trés cher frère : Aissa ;

Toute ma famille ;

Tout mes amis ;

Mohamed

INTRODUCTION GÉNÉRALE

Sans que l'on en soit toujours conscient, la Théorie des Graphes est aujourd'hui très présente dans notre société moderne. Cette branche des mathématiques, dont on fait remonter l'origine à Euler, a connu un essor spectaculaire au cours des cinquante dernières années, notamment grâce aux travaux de Claude Berge qui a grandement participé à sa diffusion. Parce qu'elle permet de modéliser aussi bien des problématiques de réseaux informatiques que de réseaux routiers, de transport de marchandises que d'emplois du temps, d'électronique que de physique du solide, la théorie des graphes a bénéficié et bénéficie encore d'un engouement considérable non seulement de la part des mathématiciens, mais également de la communauté scientifique toute entière : on observe ainsi depuis quelques années un grand nombre de publications ayant traité à des problèmes en biochimie, en génétique ou encore en sociologie, en lien direct avec la théorie des graphes.

Le plus souvent, ces problèmes se modélisent en la recherche d'une structure optimale particulière dans un graphe, structure qui doit satisfaire un certain nombre de contraintes. Par exemple, il peut s'agir d'identifier un couplage maximum (i.e. un ensemble d'arêtes deux à deux non adjacentes de taille maximum), un stable maximum (i.e. un ensembles de sommets deux à deux non adjacents de taille maximum), ou encore une coloration des sommets(i.e. une partition de taille minimum des sommets en stables).

Parfois, on a la chance de disposer d'algorithmes $\ll$ efficaces $\gg$ (c'est-àdire qui exécutent un nombre d'instructions majoré par un polynôme en la taille du problème, avant de se terminer) pour résoudre un problème donné. Par exemple, le problème du couplage maximum peut être résolu en temps polynomial grâce à l'algorithme d'Edmonds.

Mais très souvent, et c'est le cas pour les problèmes du stable maximum et de la coloration minimum, la difficulté vient justement de ce qu'un tel algorithme n'existe pas (ou du moins, la communauté s'accorde désormais à penser qu'il ne peut exister). On parle de problèmes NP-difficiles, et on se tourne alors vers d'autres solutions, comme les algorithmes d'approximation ou les métaheuristiques.

Le memoire contient cinque chapitres organisés comme suit :

Dans le premier chapitre, nous appelons quelque définitions et notions de base de la théorie des graphes, d'abord nous présentons quelques types des graphe (graphe connexe, graphe adjoint,...., etc.) et les exploration des graphes, nous donnons également un bref aperçu Précisément nous allons introduire le vocabulaire et les notations que nous utiliserons par la suite.

Dans le second chapitre nous rappelons la notion de la complexité algorithmique et quelques types des problèmes(problèmes de décisions, Problèmes d'optimisations) en suit on a classé les problèmes(P et NP et NP complet)

Dans le troisième chapitre nous donnons l'histoire et la définition d'un graphe parfait et quelques types du graphe parfait (graphe triangule) en suit la relation entre le graphe parfait. Les graphes d'intervalles inclus dans les graphes triangulés.

Dans le quatrième chapitre nous présentons quelques problèmes d'invariants ainsi la définition de problème de couplage dans un graphe biparti, définition Coloration d'un graphe triangulé.

En fin comme dernière chapitre on va présenter une application sur diférents types du problèmes sur l'invariant avec des généralisations bien précises.

CHAPITRE 1

DEFINITIONS ET NOTATION DE BASE

Dans ce chapitre, nous présentons les notations de base et quelques notions de la théorie des graphes qui nous seront utiles dans la suite. Notons qu'il y a deux concepts de graphes : graphe orienté et graphe non orienté. Dans ce manuscrit nous parlons que des graphes non orientés.

1.1 Définitions et notations

Un graphe non orienté G est un coupled' ensembles (V, E), où $V = \{v_1, v_2 \dots v_n\}$ est l'ensemble des sommets de G et $E = \{e_1, e_2 \dots e_m\}$ est l'ensemble de ses arêtes. On parle d'une arête $e \in E$ dès qu'il y a une courbe reliant deux sommets u et v et on écrit $e = \{u, v\}$ ou $e = uv$. On dit que le graphe G est d'ordre n s'il est défini par n sommets. Nous ne considérerons par la suite que des graphes finis, c'est-à-dire des graphes pour lesquels les ensembles V et E sont finis.

Etant donné un graphe $G = (V, E)$, on dit que deux sommets $u \in V$ et $v \in V$ sont adjacents, ou encore voisins, si l'arête uv appartient à E ; dans ce cas, on dit que l'arête uv est incidente aux sommets u et v, on dit aussi que les sommets u et v sont les extrémités de e. Pour un sommet $v \in V$, on désigne par $N_G(v) = \{u \in V : uv \in E\}$ le voisinage (ouvert) de v dans G. Le voisinage fermé est désigné par $N_G[v] = N_G(v) \cup \{v\}$.
Le degré d'un sommet v est défini par $d_G(v) = |N_G(v)|$. Un sommet est dit isolé s'il a pour degré 0, c'est-à-dire s'il n'a aucun voisin. On définit le degré minimum et le degré maximum de G par $\delta(G) = min\{d_G(v) : v \in V\}$ et par $\triangle(G) = max\{d_G(v) : v \in V\}$.

Dans la figure 1.1, on a un graphe d'ordre 5 avec $V = \{1, 2, 3, 4, 5\}$ et $E = \{12, 15, 25, 45, 23, 34, 24\}$;

$N(5) = \{1, 2, 4\}$; $\Delta(G) = d(2) = 4$ et $\delta(G) = d(1) = d(3) = 2$

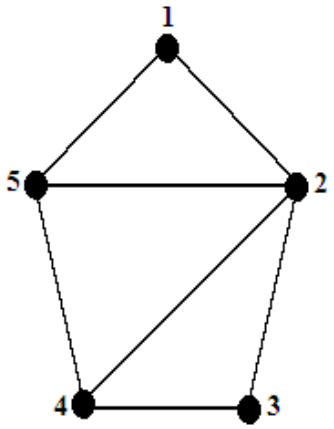

FIGURE 1.1 – Graphe non orienté

Le graphe complémentaire d'un graphe $G = (V, E)$, noté G^c, est défini par $\overline{G} = (V, E^c)$, avec $E^c = \{uv : uv \notin E\}$. Pour tout ensemble $A \subseteq V$, $G[A] = (A, E_A)$ dénote le sous graphe de G induit par A, c'est-à-dire le graphe dont A est l'ensemble des sommets et E_A est l'ensemble des arêtes de G dont les extrémités sont dans A. Dans la figure 1.1, pour $A = \{1, 2, 5\}$, $G[A] = (A, E_A)$ tel que $E_A = \{12, 25, 15\}$.

Une sequence de sommets $\mu = (v_1, v_2, \ldots, v_k)$ forment une chaine (on dit aussi chemin dans le cas d'un graphe non orienté) dans G si, pour $i = 2 \ldots k$, $v_{i-1}v_i$ est une arête dans G. La longueur de cette chaine est noté $l(\mu)$, elle est donnée par le nombre de ses aretes, c'est-à-dire $l(\mu) = k - 1$. La chaine $\mu = (v_1, v_2 \ldots v_k)$ forme un cycle dans G si v_1v_k est une arête dans G. La longueur de ce cycle est k.

Une arête v_iv_j est une corde dans la chaine $\mu = (v_1, \ldots, v_k)$ si elle relie deux sommets non consécutifs dans μ, c'est-à-dire $|i - j| > 1$. v_1v_k est une corde dans la chaine $(v_1, v_2 \ldots, v_k)$, mais ce n'est pas une corde dans le cycle $(v_1, v_2 \ldots, v_k)$. Un trou est un cycle de longueur supérieure ou égale à 4 sans corde. C_k dénote un cycle sans corde de longueur k et P_k dénote un chemin sans corde de longueur $k - 1$. Une boucle est une arête dont les extrémités sont confondues.

Dans la figure 1.2, (a) est un P_4, (b) est un C_6 et (c) est un cycle de longueur 6 avec deux cordes : $c_1 = 14$ et $c_2 = 35$.

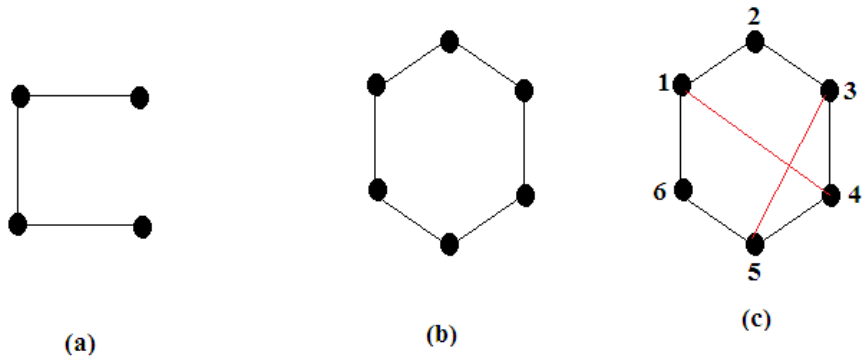

FIGURE 1.2 – Corde

Une clique dans G est un sous-ensemble de sommets de V deux à deux adjacents ; Un stable (on dit aussi ensemble indépendant) dans G est un sous-ensemble $S \subseteq V$ dont les sommets sont deux à deux non adjacents. Un stable (resp. clique) ayant le maximum possible d'éléments est dit maximum ; un stable maximal (resp. clique maximale) est un stable (resp. clique) qui n'est pas inclus(e) dans un autre stable (resp. clique). Notons que un stable (resp. clique) maximum est nécessairement maximal(e), mais pas l'inverse (voir figure 1.3). La taille d'un stable maximum est noté par $\alpha(G)$ et la taille d'une clique maximum par $\omega(G)$.
La proposition ci-dessous donne une relation entre les invariants $\alpha(G)$ et $\omega(G)$.

Proposition 1.1. *Pour tout graphe G on a $\alpha(G) = \omega(G^c)$.*

Démonstration. Il suffit de remarquer qu'un stable dans G est une clique dans G^c, et vice versa. □

Dans le graphe de la figure 1.3, $C = \{a, b, e\}$ est une clique maximum d'ordre 3, donc $\omega(G) = 3$; $C' = \{a, d\}$ est une clique maximale. $I = \{c, e, d\}$ est un stable maximum d'ordre 3, donc $\alpha(G) = 3$; $I' = \{a, f\}$ est un stable maximal.

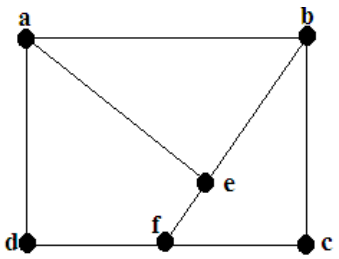

FIGURE 1.3 – Stable et clique

Un graphe orienté G est une paire (V, U) où V est un ensemble de sommets et U est un ensemble d'arcs. Pour un arc $a = (u, v) \in A$, u est l'extrémité initiale de l'arc orienté et v est extrémité terminale ; on dit que l'arc va de u vers v. Dans la suite, lorsque nous utiliserons la notion d'arc orienté, la notation (u, v) désignera un arc. Lorsque nous considérerons des arêtes, la notation uv désignera une arête.

1.2 Quelques graphes remarquables

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

Un graphe G est dit simple s'il est sans boucle et entre deux sommets il y a au plus une arête. Tous les graphes considérés dans ce mémoire sont des graphes simples.

Un graphe G est complet si ses éléments sont deux à deux adjacents . K_n désigne un graphe complet d'ordre n. Un graphe $G = (V, E)$ est dit biparti si l'ensemble de ses sommets peut être partitionné en deux sous ensembles $V = X \cup Y$ tel que les éléments de X (resp. Y) sont deux à deux non adjacents ; dans ce cas, on écrit $G = (X, Y, E)$. Un graphe biparti G est dit complet si tout élément de X est adjacent à tous les éléments de Y ; la notation $K_{n,m}$ désigne un graphe biparti complet, avec $|X| = n, |Y| = m$.

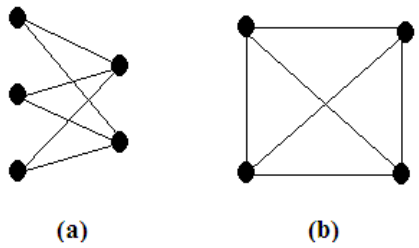

FIGURE 1.4 – (a) $K_{3,2}$, (b) K_4

Un graphe $G = (V, E)$ est connexe si pour toute paire u et v, il existe une chaine dans G passant par u et v. Une composante connexe dans un graphe G non connexe, est un sous-graphe induit qui est connexe et maximal pour cette propriété. Un graphe connexe sans cycle est appelé un arbre. Un trou est un cycle sans corde de longueur supérieure ou égale à quatre. Un anti-trou est le complémentaire d'un trou.

La figure 1.5 montre un graphe avec trois composantes connexes, $C_1 = G[\{a, b, c, d\}]$, $C_2 = G[\{e, f, g\}]$, et $C_3 = G[\{h\}]$.

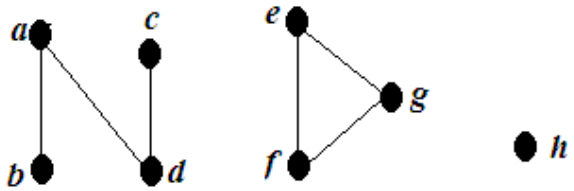

FIGURE 1.5 – Graphe non connexe

Nous dirons également que deux arêtes sont adjacentes si elles possèdent au moins une extrémité commune. On peut alors définir le ≪ graphe adjacence des arêtes ≫ de G, ou graphe adjoint de $G = (V, E)$ de cette manière : Le graphe adjoint (line graph) d'un graphe $G = (V, E)$ est le graphe noté $L(G)$ tel que :

- $V_{L(G)} = EG$,

- Deux sommets de $L(G)$ sont adjacents si et seulement si les arêtes correspondantes dans G sont adjacentes.

La figure 1.4 illustre cette construction.

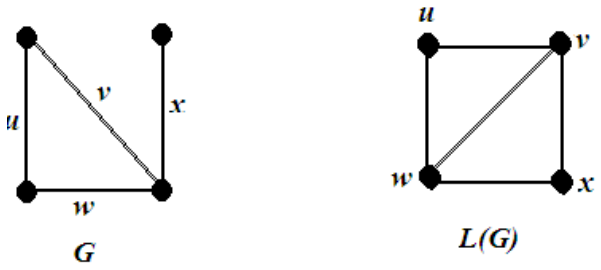

FIGURE 1.6 – Graphe adjoint

Par définition, tout graphe possède un graphe adjoint ; mais un graphe n'est pas forcément le graphe adjoint d'un autre graphe. Considérons par exemple le graphe biparti complet $K_{1,3}$ appelé griffe (claw), on peut vérifier facilement que ce graphe ne peut être le graphe adjoint d'aucun graphe et donc pareille pour tout graphe qui contient une griffe comme un sous graphe induit [15].

1.3 Exploration des graphes

Les parcours servent comme outils pour étudier une propriété globale d'un graphe, comme le graphe est-il biparti ? la connexité ou la forte connexité ...

Il existe de nombreux algorithmes de parcours. Les plus couramment décrits sont le parcours en profondeur et le parcours en largeur. L'algorithme de Dijkstra et l'algorithme de Prim font également partie de cette catégorie.

1.3.1 Parcours

En théorie des graphes, un parcours de graphe est un algorithme consistant à explorer les sommets d'un graphe de proche en proche à partir d'un sommet initial. Un cas particulier important est le parcours d'arbre.
Le mot parcours est également utilisé dans un sens différent, comme synonyme de chemin (un parcours fermé étant un circuit).

1.3.2 Structure de données

File

Une File est une structure de données basée sur le principe du "Premier entré, premier sorti FIFO, ce qui veut dire que les premiers éléments ajoutés à la file seront les premiers à être récupérés.

Pile

Une pile est une structure de données basée sur le principe du "dernier arrivé, premier sorti LIFO , ce qui veut dire que les derniers éléments ajoutés à la pile seront les premiers à être récupérés.

• La Recherche en Profondeur correspond à prendre une structure de PILE, c'est à dire une liste LIFO Last In/First Out (dernier entré/premier sortie), il est possible d'obtenir une complexité $O(n + m)$ pour un algorithme effectuant un parcours en profondeur.

Remarque : La notion d'exploration / parcours peut être utilisée dans les graphes orientés comme non-orientés. Dans la suite, nous supposerons que le graphe est non-orienté. L'adaptation au cas des graphes orientés s'efectue sans aucune diffculté.

1.3.3 Parcours en profondeur DFS (Depth-First Search)

1.3.3.1 Principe

C'est un algorithme de recherche qui progresse à partir d'un sommet V en s'appelant récursivement pour chaque sommet voisin de V.
Le nom d'algorithme en profondeur est dû au fait que, contrairement à l'algorithme de parcours en largeur, il explore en fait $\ll$ à fond $\gg$ les chemins un par un : pour chaque sommet, il marque le sommet actuel, et il prend le premier sommet voisin jusqu'à ce qu'un sommet n'ait plus de voisins (ou que tous ses voisins soient marqués), et revient alors au sommet père.
Si G n'était pas un arbre, l'algorithme pourrait tourner indéfiniment, c'est pour cela que l'on doit en outre marquer chaque sommet déjà parcouru, et ne parcourir que les sommets non encore marqués.
Dans le cas d'un arbre, le parcours en profondeur est utilisé pour caractériser l'arbre.
Enfin, on notera qu'il est tout à fait possible de l'implémenter itérativement à l'aide d'une pile $LIFO$ contenant les sommets à explorer : on désempile un sommet et on empile ses voisins non encore explorés.

1.3.3.2 Implémentation récursive

Durant l'exploration, on marque les sommets afin d'éviter de re-parcourir des sommets parcourus. Initialement, aucun sommet n'est marqué.

algorithme :
explorer(graphe G, sommet s)
marquer le sommet s
afficher(s)
pour tout sommet t voisin du sommet s
si t n'est pas marqué **alors**
explorer(G, t) ;

La complexité en temps d'un parcours en profondeur est $O(n + m)$
• L'algorithme de parcours en profondeur peut être modifié pour résoudre d'autres problèmes sur les graphes :

- trouver et retourner un chemin entre deux sommets.
- trouver un cycle dans un graphe.

1.3.3.3 Exemple d'application

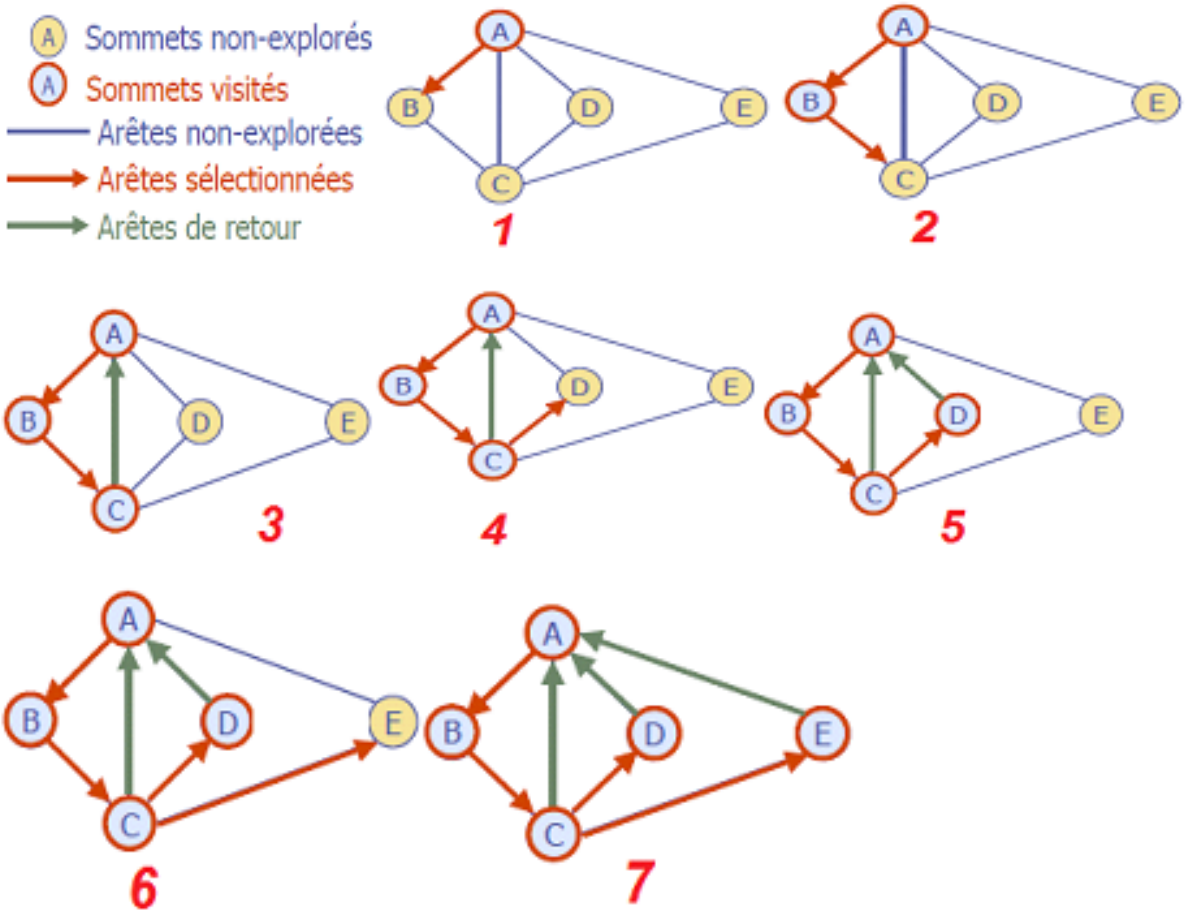

FIGURE 1.7 – Graphe quelconque DFS

1.3.4 Parcours en largeur BFS (Breadth-First Search)

1.3.4.1 Principe

L'algorithme de parcours en largeur (ou BFS, pour Breadth First Search en anglais) permet le parcours d'un graphe ou d'un arbre de la manière suivante : on commence par explorer un noeud source, puis ses successeurs, puis les successeurs non explorés des successeurs, etc. L'algorithme de parcours en largeur permet de calculer les distances de tous les noeuds depuis un noeud source dans un graphe non pondéré (orienté ou non orienté). Il peut aussi servir à déterminer si un graphe non orienté est connexe.

1.3.4.2 Pseudo code

L'algorithme s'implémente à l'aide d'une file.

Algorithme : ParcoursLargeur(Graphe G, Sommet s) :
f = CreerFile() ;
f.enfiler(s) ;
marquer(s) ;
tant que la file est non vide
s = f.defiler() ;
afficher(s) ;
pour tout voisin t de s dans G
si t non marqué f.enfiler(t) ;
marquer(t) ;

La complexité en temps d'un parcours en largeur est $(n + m)$

1.3.4.3 Exemple d'application

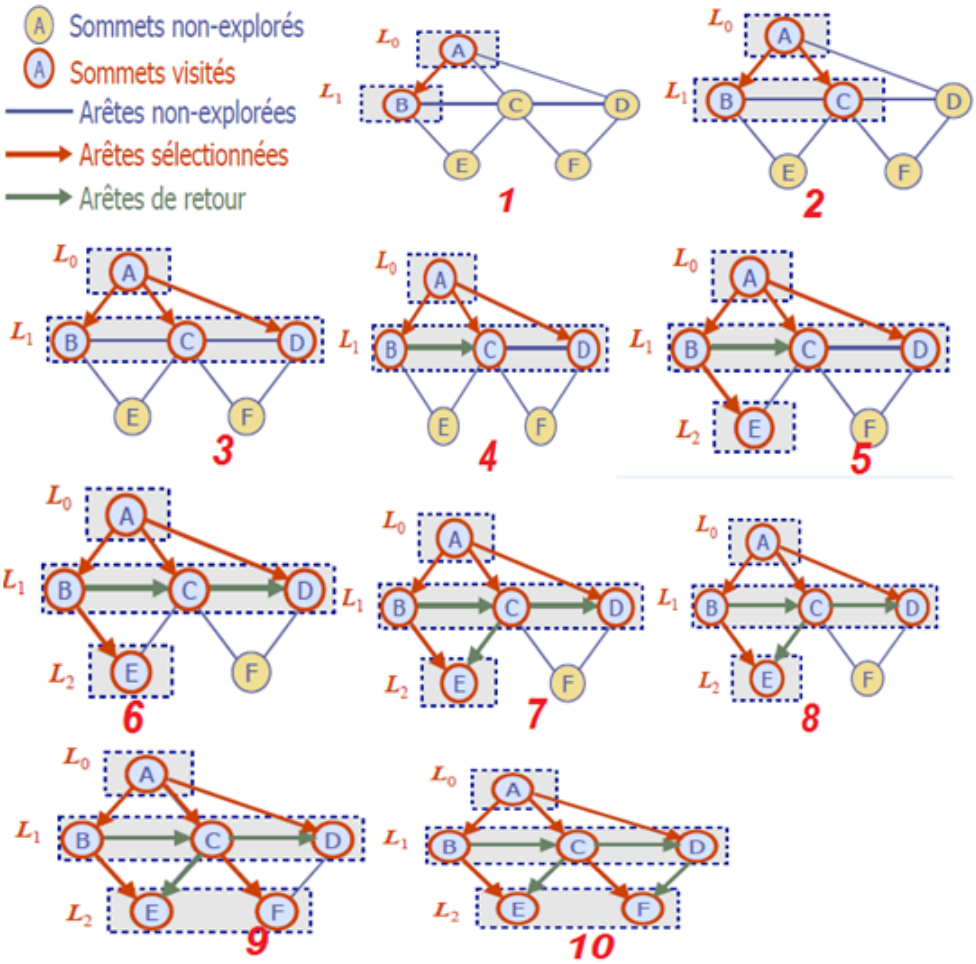

FIGURE 1.8 – Graphe quelconque BFS

1.3.4.4 Méthode hongroise

L'algorithme hongrois ou méthode hongroise (parfois appelé aussi algorithme de kuhn) est un algorithme d'optimisation combinatoire, que résout le problème d'affectation en temps polynômial. Il a été proposé en 1955 par le mathématicien américain harold kuhn, que la baptisé " méthode hongroise " parce qu'il s'appuyait sur des travaux antérieurs de deux mathématiciens : Dénes Konig et Jeno Egervary.

Étape 1 :

Réduction des lignes : trouver l'élément minimum dans chaque linge de la matrice $n \times n$. le soustraire de chaque ligne le coût minimum dans sa ligne .

Étape 2 :

Réduction des colonnes : pour cette nouvelle matrice, trouver le coût minimum dans chaque colonne. Construire une nouvelle matrice en soustrayant dans chaque colonne sont minimum.

Étape 3 :

Déterminer le nombre minimal de lignes nécessaires sur les lignes et les colonnes pour couvrir tout les zéros.

- Si ce nombre est égal au nombre de ligne (ou colonnes), la matrice est réduite, aller à l'étape 5 ;
- Si ce nombre est inférieur au nombre de ligne ;aller à l'étape 4 ;

Étape 4 :

Trouver la cellule de valeur minimum non-couverte par une ligne.

- Soustraire cette valeur de toutes les cellules non couverte ;
- Ajouter cette valeur aux cellules situées à l'intersection de deux ligne.retourner à l'étape 3.

Étape 5 :

Déterminer la solution optimal.

CHAPITRE 2

COMPLEXITÉ

2.1 La notion de la complexité algorithmique

2.1.1 Introduction

La théorie de la complexité est une théorie jeune, beaucoup plus que la théorie des graphes puisque ses origines remontent a la fin des années 60, bien qu'on trouve déjà des notions d'algorithmes efficaces chez les Grecs anciens, ou encore chez Lame en 1844 et qu'elle repose sur les travaux de Turing en 1937. Fortnow et Homer font coïncider l'origine de la théorie de la complexité avec la publication en 1965 d'un article de Hartmanis et Stearns intitulé "On the Computational Complexity of Algorithms" dans lequel on trouve les premières définitions de complexité temporelle et spatiales (exprimées à l'aide de machines de Turing 50). En 1965 aussi, Edmonds a explicité un algorithme en temps polynomial pour le problème du couplage maximum (cf. section 4.1), et affirme qu'il s'agit d'une bonne formalisation de la notion de calcul "efficace". Ceci est annonciateur de la fameuse question $P = NP$?, sans doute le plus célèbre problème en informatique théorique ; ses implications sont telles qu'il figure sur la liste des sept problèmes du millénaires du "Clay Mathematics Institute", chacun récompensé d'un prix d'un million de dollars pour sa résolution ! Une étape cruciale dans l'histoire de la théorie de la complexité eut lieu en 1971 quand Cook [2] aux Etats-Unis, et indépendamment Levin [3] en $U.R.S.S$, apportèrent la preuve de l'existence de problèmes dits NP-complets, un résultat connu aujourd'hui sous le nom de théorème de Cook-Levin. Comme nous le verrons dans la section suivante, les problèmes NP-complets sont d'une certaine manière les problèmes les plus difficiles que nous rencontrerons par la suite. Le premier

problème à avoir été démontré NP-complet est SAT ou problème de satisfaisabilité. Les implications du théorème de Cook-Levin sont extrêmement importantes. Tout d'abord, il signifie que tous les problèmes d'une classe appelée NP (qui contient les problèmes NP-complets) peuvent se ramener à SAT au moyen d'une transformation appelée réduction polynomiale. Grâce à cette transformation, Karp [4] à pu prouver la NP-complétude de 21 problèmes réputés difficiles de combinatoire et théorie des graphes, connus aujourd'hui comme les 21 problèmes de Karp. Depuis, on a recense plusieurs milliers de problèmes NP-complets ; les plus célèbres et importants d'entre eux sont recensés dans l'ouvrage de référence de Garey et Johnson [29]. Les problèmes NP-complets sont donc plus ou moins tous équivalents entre eux. une conséquence fondamentale du théorème de Cook-Levin est que si on arrive à mettre au point un algorithme $\ll$ efficace $\gg$ pour résoudre l'un d'entre eux, tous les autres pourront aussi être résolus efficacement.

2.1.2 Problèmes de décisions et Problèmes d'optimisations

Un problème est une question générique, c'est-à-dire qui s'applique à un ensemble d'éléments. Une instance (une donnée) du problème est une question posée pour un élément particulier de cet ensemble. Par exemple déterminer si un entier naturel est premier ou calculer une clique de cardinalité maximal sont deux problèmes. Dès qu'on fixe un entier, on aura une instance du premier problème (connu sous le non du probleme de primalité) : 19 est-il un nombre premier ? est une instance ayant une reponse oui. De même, la donnée d'un graphe G va constituer l'instance " calculer la clique maximum pour le graphe G".
Un problème est donc composé de deux éléments : une entrée (ou instance) et une question ou une tâche à réaliser. Les deux exemples précédents se reformulent ainsi :

1. Primalité :
   - entrée : un entier N ;
   - question : N est-il premier ?
2. clique de taille maximale :
   - entrée : Un graphe $G = (V, E)$;
   - tâche : Trouver une clique de G avec un maximum possible de sommets.

On distingue ainsi deux types de problèmes : ceux qui consistent à répondre par oui ou par non à une question donnée (dans l'exemple précédent, déterminer si un entier est premier), qu'on appelle problèmes de décision ; et ceux qui consistent à maximiser (ou minimiser) une certaine fonction sur un ensemble fini (dans l'exemple précédent, trouver $max(f)$ sur X, avec X est l'ensemble de

toutes les cliques de G et f est une application sur X, qui à toute clique $C \in X$, $f(C)$ est le nombre de sommet de la clique C), qu'on appelle problèmes d'optimisation. Ainsi, on obtient la définition suivante :

Définition 2.1.
- Un problème d'optimisation combinatoire consiste à chercher le minimum (resp. le maximum) x^* d'une certaine application f sur un ensemble fini X à valeur le plus souvent entières ou réelles : $f(x^*) = min_{x \in X} f(x)$ (resp ; $f(x^*) = max_{x \in X} f(x)$). La fonction f est une fonction-objectif ou économique, l'ensemble X est appelé ensemble des solutions réalisables.
- Un problème de décision (on dit aussi langage) consiste à chercher dans un ensemble fini X s'il y a un élément x vérifiant une certaine propriété P. Ainsi un problème de décision est une application sur X à valeur dans $\{0, 1\}$ telle que : $f(x) = 1$ si x vérifie P et $f(x) = 0$ sinon.

Pour resoudre un problème P, il nous faut un algorithme A (ou un programme). D'une manière informelle, un algorithme est un ensemble fini d'opérations de calcul élémentaires qui prend en entrée toute donnée d du problème P et produit en sortie un resultat $A(d)$. Ainsi, un algorithme qui résout le problème de Tri quand il s'applique sur l'instance $(3, 0, 12, 30, 5)$, une suite de nombres désordonnée, produit le resultat $(0, 3, 5, 12, 30)$ qui est une suite triée par un ordre croissant.

Si un problème doit être résolu par un algorithme, il faut que ses instances soient représentées d'une façon accessible à cet algorithme. Donc on a besoin de coder des entiers, des listes, des arbres, des graphes ... Le par des chaines de caractère qui soient comprehensibles par l'ordinateur. Le plus souvent, ces instances sont représentées par des chaines en 0 et 1. Pour fixer les idées, les définitions ci-dessous nous sont utiles.

Codage des mots

Un alphabet Σ est un ensemble fini non vide dont les éléments sont appelés des symboles ou des lettres. Un mot x (ou une chaîne) sur l'alphabet Σ est une suite finie $x_1 x_2 \dots x_n$ de lettres de Σ. Par exemple, si $\Sigma = \{a; b\}$, *abba*, *bab* et *bbbb* sont des mots sur Σ.
On note $taille(w) = n$(ou $|w| = n$) la longueur du mot $w = w_1 w_2 \dots w_n$. Ainsi $|abba| = 4$, $|bab| = 3$ et $|bbbb| = 4$. Σ^n est l'ensemble des mots de longueur n, et $\Sigma^* = \bigcup_{n \in Z_+} \Sigma^n$ l'ensemble de tous les mots sur Σ. Le mot vide ξ est le seul mot de longueur 0, c'est-à-dire $\xi \in \Sigma^*$ et $|\xi| = 0$. Le

mot vide est utilisé comme un blanc pour séparer les mots non vides. Un langage sur Σ est un ensemble $L \subseteq \Sigma^*$ de mots sur Σ . On définit une opération de concaténation sur les mots de Σ^* de la manière suivante : la concaténation du mot $u = u_1u_2 \dots u_n$ et du mot $v = v_1v_2 \dots v_m$ est le mot noté $u.v = u_1u_2 \dots u_nv_1v_2 \dots v_m$, c'est-à-dire le mot dont les lettres sont obtenues en juxtaposant les lettres de v à la fin de celles de u. En particulier, w^n est la concaténation de n copies du mot w, et on pose $u^0 = \xi$. Par exemple, $(aba)^3 = abaabaaba$. L'opération de concaténation notée . est associative, mais non-commutative. Le mot vide est un élément neutre à droite et à gauche de cette opération. On utilise la convention que A_0 contient exactement un élément, la chaîne vide.

On utilise souvent l'alphabet $\Sigma = \{0, 1\}$. Ainsi $\{0, 1\}^*$ est l'ensemble de toutes les chaînes en $0 - 1$ (ou chaînes binaires). Les composantes d'une chaîne en $0 - 1$ sont parfois appelées bits. Il y a donc exactement une seule chaîne en $0 - 1$ de longueur zéro, la chaîne vide. Un langage sur $\{0, 1\}$ est un sous-ensemble de $\{0, 1\}^*$.

Codage, Décodage Nous appellerons fonction d'encodage (encoding function) la fonction qui, à partir d'une instance d'un problème, donne son encodage sous la forme d'une chaîne de caractères, c'est-a-dire toute instance de ce probleme est représentée par une chaine de caractères definis sur un alphabet fini. Ainsi, toutes les données (graphes, nombres, matrices, chaînes de caractères,...) utilisées en informatique sont clairement représentables par des chaînes de caractères.

Exemple. ***Codage des entiers***

On peut coder les entiers

- *En base* 2 *:* $< x >$ *serait l'écriture en binaire de* x *sur l'alphabet* $\Sigma = \{0, 1\}$. *Par exemple* $< 12 >= 1100$, $< 27 >= 11011$.
- *En base* 10 *:* $< x >$ *serait l'écriture usuelle de l'entier* x *sur l'alphabet* $\Sigma = \{0, 1, \dots, 9\}$. *Par exemple,* $< 12 >= 12$.
- *En unaire :* $< x >$ *serait le mot* $11 \dots 11$ x *fois, c'est-à-dire le mot de longueur* x *sur l'alphabet* $\Sigma = \{1\}$. *Pour* $x = 12$, $< 12 >= 111111111111$.

Soit maintenant un problème de dècision dont les instances sont encodées par des mots Σ^* définis sur un alphabet Σ. D'apres ce qui précède, l'ensemble Σ^* est partitionné en deux sous-ensembles :

• les mots représentant des instances du problème pour lesquelles la réponse est oui, ce sont les instances positives (positive instances) ;

• les mots représentant des instances du problème pour lesquelles la réponse est non, ce sont les

instances négatives (négative instances).
On peut donc dire qu'un problème est caractérisé par le langage des encodages de ses instances positives, c'est-a-dire résoudre un problème revient à reconnaître les instances positives de ses encodages.

2.1.3 Complexité d'un algorithme

A tout algorithme est associee deux fonctions de complexité, une fonction de complexité spaciale qui donne l' évaluation des ressources nécessaires à son exécution, essentiellement la quantité de mémoire requise, et la fonction de complexité temporelle qui indique le temps de calcul à prévoir pour obtenir la solution. Pour une instance (une donnée) x d'un problème donné p, le temps d'exécution de l'algorithme pour résoudre le probléme p sur l'entrée x ne s'exprime pas en seconde, mas il est donné par le nombre fini d'opérations élémentaires nécessaires jusqu'à l'affichage de la solution. Parmi les oprrations elementaires, on a les opérations arithmétiques ou logiques, les affectations, les comparaisons.... Toute tâche qui se réalise en un temps constant par les calculateurs usuels indépendamment de leurs puissances est opération élémentaire. Notons que la fonction de compexite spaciale est toujours majoree par la fonction de complexite temporelle. En effet, l'utilisation de chaque unité élémentaire de mémoire nécessite au moins l'exécution d'une instruction. Par la suite, nous ne parlerons pas de la complexité en espace mémoire.
En générale, le temps d'exécution d'un algorithme sur l'entrée x dépend de la taille de x mais aussi de sa nature, et il peut varier sur des instances de même taille. En effet, rechercher une valeur dans un tableau demande plus de temps dans un tableau dont les éléments sont désordonnés que dans le même tableau trié . Pour cette raison, on définit la complexité d'un algorithme en considérant la pire instance possible parmi toutes les instances de taille n, c'est-à-dire celle demandant le plus de ressources.

Définition 2.2. Soient p un probléme 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 (ou tout court la complexité) de l'algorithme A est une application c_A définie sur l'ensemble des entiers naturels par :

$$c_A(n) = max_{I_{(P,n)}}\varphi(n).$$

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

Remarque 2.1. *1. Il existe aussi d'autres types de complexités, 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 à calculer. Pour ces raisons, on se contente généralement de la complexité asymptotique dans le pire des cas.

2. *Un algorithme de complexité polynômiale est quelquefois appelé bon ou efficace, et le problème qu'ilresout est dit facile.*

La complexité en temps d'un algorithme est habituellement exprimée à l'aide de la notation O. Sa définition est la suivante.

Définition 2.3. (Notation O) Soient f et g deux fonctions $f; g : N \to R_+$. On dit que $f \in O(g)$ (on dit aussi f est un grand O de g) lorsqu'il existe un entier n_0 et une constante reelle c tel que pour tout $n \geq n_0$, $f(n) \leq cg(n)$.

Intuitivement, cela signifie que g devient plus grande que f à partir d'un certain entier n_0, à une constante multiplicative près. $O(g)$ est l'ensembles des fonctions d'ordre supérieur à g pour n assez grand. Par abus de langage, on écrira $f = O(g)$ là où on devrait èrcire $f \in O(g)$.

Exemple. *:Soit* $f(n) = 6n^4 - 2n^3 + 5$. *Choisissons* $n_0 = 1$. *Alors pour tout* $n \geq n_0$, *on a* $6n^4 - 2n^3 + 5 \leq 6n^4 + 2n^4 + 5n^4 = 13n^4$ *Ainsi, en prenant* $c = 13$, *on a* $f = O(n^4)$. *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* $O(n^k)$.

Définition 2.4. (Notations Ω, Θ)
Soient f et g deux fonctions $f; g : \mathbb{N} \to \mathbb{R}_+$.

- On note $f = \Omega(g)$ lorsque il existe un entier n_0 et une constante reelle $c^{'}$ tel que pour tout $n \geq n_0$, $f(n) \geq c^{'}g(n)$.
- On note $f = \Theta(g)$ lorsque $f(n) = O(g)$ et $f(n) = \Omega(g(n))$, c'est-à-dire lorsque il existe un entier n_0 et deux constantes reelles c et $c^{'}$ tel que $cg(n) \leq f(n) \leq c^{'}g(n)$.

$\Omega(g)$ (respectivement $\Theta(g)$) est l'ensemble des fonctions d'ordre inférieur (respectivement équivalentes) à g pour n assez grand.

Exemple. *Soit* $f(n) = n^3 sinn$, *on a* $\forall n$, $-n^3 \leq f(n) \leq n^3$. *Ainsi* $f(n) = \Theta(n^3)$.

Définition 2.5. Un algorithme A est dit polynômial si sa complexité est majorée par un polynôme en la taille des données, c'est-à-dire il existe un entier k tel que $c_A \in O(n^k)$. Dans le cas contraire, il est dit exponentiel.

2.1.4 Classes des problèmes P et NP

Nous allons maintenant nous intéresser à l'étude de la difficulté intrinsèque des problèmes de décision, ce que l'on appelle complexité des problèmes (non pas des algorithmes), et on va les classer selon la complexité des algorithmes les résolvant. Un grand nombre d'entre eux sont des problémes faciles car on connait des algorithmes polynômiaux pour les résoudre. Cependant, il existe aussi un grand nombre de problèmes pour lesquels on ne connait pas d'algorithmes polynômiaux. On ne peut pas prouver qu'il n'en existe pas, mais on peut cependant montrer que l'existence d'un algorithme polynomial pour l'un d'entre eux impliquerait l'existence d'un algorithme polynomial pour presque tous les problèmes.

Définition 2.6. La complexité d'un probléme est la complexité du meilleur algorithme qui permet de le résoudre. Si cet algorithme est polynômial, le problème est dit facile, autrement le probléme est difficile.

Définition 2.7. La classe P est l'ensemble de tous les problèmes de décision pour lesquels il existe un algorithme polynômial.

Pour prouver qu'un problème est dans P, on décrit un algorithme polynômial résolvant ce problème.

Exemple. *Soit le problème du plus court chemin entre deux sommets dans un graphe orienté et valué par des coûts positifs.*

- *Entrée : un graphe valué $G = (V, E, u)$, deux sommets s et t, et une constante k positive ;*
- *Sortie : exite-t-il un chemin allant de s à t de longueur au plus k.*

Ce problème est dans la classe P car l'algorithme de Dijkstra, qui est un algorithme polynômial, peut résoudre ce problème.

Cependant, on ne sait pas si les problèmes de décision associés au problème du cycle hamiltonien au problème du stable maximum appartiennent à P ou pas. Nous allons maintenant introduire une autre classe, notée NP, qui contient ces deux problèmes. Pour les problèmes de la classe NP, nous n'exigeons pas un algorithme polynômial, en revanche nous demandons qu'il y ait, pour chaque instance "oui", un certificat (une solution devinée) qui puisse être vérifié en temps polynômial. Par exemple, pour le problème du cycle Hamiltonien, la solution est un cycle Hamiltonien, donc un cycle hamiltonien constitue un certificat. Comme il est facile de vérifier si un ensemble d'arêtes donné forme un cycle hamiltonien, alors ce problème est dans NP.
La même chose pour le problème du stable de cardinalité au plus égale à k. En effet, ici la solution

est un stable, et si l'on dispose d'un stable S (S est un certificat), on peut vérifier en un temps polynômial que S est un stable et aussi si $|S| \geq k$, ce qui implique que ce problème est dans NP.

Définition 2.8. La classe NP est l'ensemble de tous les problèmes de décision pour lesquels toute solution proposée est vérifiable par un algorithme polynômial.

Remarque 2.2.
- *Les problèmes de la classe NP sont ceux que l'on peut rèsoudre par énumération complète de toutes les solutions possibles (méthode "brutale") et en les testant à l'aide d'un algorithme polynômial.*
- *On a clairement $P \subseteq NP$. En effet, si on peut résoudre un problème par un algorithme polynômial, alors on peut aussi vérifier en temps polynômial que la solution fournie est bien une solution du même problème.*
- *La question de savoir si $P = NP$ est un problème ouvert, le plus important, de la théorie de la complexité. Cela revient à savoir si le fait de chercher une solution est aussi simple que de vérifier une solution. De nombreuses personnes pensent que $P \neq NP$.*

2.1.5 Classe NP-complet

Elle est constituée par les problèmes les plus difficiles de NP. Les problèmes $NP - complet$ sont tous équivalents en termes de difficultés. Pour affirmer que certains problèmes sont les plus difficiles, il faut pouvoir comparer les problèmes entre eux. On définit pour cela la notion de réduction polynômiale sur la classe NP.

Définition 2.9. Soient p et p' deux problèmes de décision. On dit que p se transforme (ou se réduit) polynômialement en p' s'il existe un algorithme polynômial transformant toute instance x de p en une instance x' de p' admettant la même réponse que x. On écrit alors $p \prec p'$. Autrement dit, les instances "oui" sont transformées en instances "oui", et les instances "non" sont transformées en instances "non".

L'importance du concept de réduction polynômiale est principalement justifiée par la proposition suivante :

Proposition 2.1. ***proposition****[Optimisation combinatoire] Si p se réduit polynômialement à p' et s'il existe un algorithme polynômial pour p', alors il existe un algorithme polynômial pour p.*

Remarque 2.3.
- *La relation $\prec$ est transitive et $p \prec p'$ signifie que p n'est pas plus difficile que p'.*

– *Ainsi, "p est polynômialement réductible à p′" signifie que si l'on connait un algorithme polynômial résolvant p′, on en déduit que p ∈ P. En effet, on traduit les instances de p en instances de p′, puis on résout p′ et enfin on retraduit les solutions de p′ en solutions de p, tout cela se fait en temps polynômial.*

Définition 2.10. Un problème de decision q est dit $NP - complet$ si :

1. $q \in NP$
2. $\forall p \in NP,\ p \prec q$

Il est facile de voir que la restriction de la notion de "réduction polynô-miale" sur les problèmes $NP - complet$ est une relation d'équivalence, donc s'il existe un algorithme polynômial pour un seul élément de la classe $NP - complet$, on pourait en déduire un algorithme polynômial pour n'importe quel autre élément dans la même classe, et d'après la proposition précédente, on aurait aussi $P = NP$!

Il a été demontré que de nombreux problèmes naturels réputés difficiles sont $NP - complets$. Cook [2] et, indépendamment, Levin [3] ont donné de tels problémes au début des années 1970, complétés ensuite par Karp [4] notamment. Ici nous allons enoncé le théorème du problème SAT, la satisfaisabilité de formules booléennes.

Définition 2.11. Soient $X = \{x_1, \ldots, x_k\}$ un ensemble de variables boolêennes et $\overline{X} = \{\overline{x} : x \in X\}$, où $\overline{x}$ est la négation de la variable x. Un assignement pour X est une application $T : X \cup \overline{X} \longrightarrow \{vrai, faux\}$ vérifiant : $T(x) = vrai$ si et seulement si $T(\overline{x}) = faux$. Les éléments de $X \cup \overline{X}$ sont appelés les littéraux sur X.
Une clause sur X est un ensemble de littéraux sur X. Une clause peut être considérée aussi comme la disjonction de ses littéraux et est satisfaite par un assignement si et seulement si au moins un de ses membres est vrai. Une famille Z de clauses sur X est satisfaisable si et seulement s'il existe un assignement satisfaisant simultanément toutes ses clauses. Puisque nous considérons la conjonction de disjonctions de littéraux, nous parlons aussi de formules booléennes sous forme normale conjonctive.

Exemple. *Pour $X = \{x_1, x_2, x_3\}$, on a $\overline{X} = \{\overline{x}_1, \overline{x}_2, \overline{x}_3\}$. La famille $Z = \{\{x_1, \overline{x}_2\}, \{\overline{x}_2, \overline{x}_3\}, \{x_1, x_2, \overline{x}_3\}, \{\overline{x}$ qui correspond à la formule booléenne $(x_1 \vee \overline{x}_2) \wedge (\overline{x}_2 \vee \overline{x}_3) \wedge (x_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_3)$ est satisfaite. En effet, l'assignement défini par $T(x_1) = vrai$, $T(x_2) = faux$ et $T(x_3) = vrai$ satisfait toutes les clauses.*

Le problème de satisfaisabilité (ou simplement SAT) s'annonce de la manière suivante :

- Instance : Un ensemble X de variables et une famille Z de clauses sur X.
- Question : Existe-t-il une fonction d'assignation T permettant de satisfaire simultanément toutes les clauses de Z ?

Théorème 2.2. *(Cook [1971]) Le problème de la satisfaisabilité est $NP-complet$.*

Ainsi, à partir de SAT, on montre progressivement la $NP-compltude$ d'autres problèmes qui, à leur tour, peuvent être utilisés pour démontrer la $NP-compltude$ de nouveaux problèmes, et ainsi de suite. Par exemple, à partir de SAT, on a montré que $3-SAT$ est $NP-complet$ ($Karp$(1972)[4]) (le Problème $3-SAT$ s'énonce comme SAT avec la condition que toutes les clauses sont de cardinalité 3). Maintenant , on va utiliser $3-SAT$ pour démontrer la $NP-compltude$ du problème du stable maximum.

Mais citons d'abord la démarche à suivre pour montrer la $NP-compltude$ d'un problème de décision p quelconque. qui est la suivante :

1. Montrer que $p \in NP$ en dennant un certificat, c'est-à-dire en devinant une solution qui doit être vérifiée en un temps polynômial ;
2. Choisir un problème q connu pour être $NP-complet$;
3. Montrer la relation $q \prec p$.

Exemple. *Le problème du stable maximum (MIS) est $NP-complet$ ($Karp$(1972)[4]).*
On va suivre la démarche précédente pour le montrer :

1. *Vérifier qu'un ensemble de sommet est un stable contenant au moins k élément se fait en temps polynômial, donc le problème $MIS \in NP$;*
2. *Le problème $3-SAT$ est $NP-complet$ (Karp [4]) ;*
3. *Montrons maintenant que $3-SAT \prec MIS$.*

On associe à chaque littéral de la formule booléenne un sommet d'un graphe (en le dupliquant éventuellement autant de fois qu'il apparaît dans ladite formule) et on relie ensuite entre deux sommets s'ils correspondent respectivement à un littéral et à sa négation, ou s'ils apparaissent dans la même clause. Par exemple, la figure suivante donne le graphe associé à la formule $(x_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (x_1 \vee x_2 \vee x_4)$ *:*

Le graphe ainsi construit contient un ensemble stable de taille au moins k (k étant le nombre de clauses) si et seulement si la formule booléenne est satisfaisable. En effet :

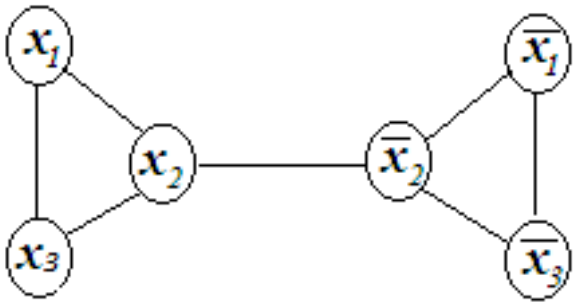

FIGURE 2.1 –

$\Longrightarrow$) *Si l'on connait une fonction d'assignation T telle que la formule est vraie, on choisit dans chaque clause un littéral rendu vrai par T (et on peut le faire, puisque la formule est vraie).*

L'ensemble constitué des sommets correspondants dans le graphe est bien stable (puisqu'il ne contient qu'un sommet de chaque clause et qu'aucune fonction d'assignation ne rend vrais à la fois un littéral et sa négation) et est de taille k car la formule contient k clauses.

$\Longleftarrow$) *Réciproquement, supposons que nous ayons un stable S de taille k ou plus. Pour les raisons que l'on vient d'éévoquer, il ne peut contenir deux sommets correspondant à des littéraux opposés ou appartenant à une même clause, et comme la formule contient k clauses, chaque clause doit contenir un littéral correspondant à un sommet de S. Par conséquent, en donnant la valeur vrai aux littéraux correspondant aux sommets de S, la formule est satisfaite.*□

Il est facile de voir que cette réduction peut être effectuée en temps polynômial. Par conséquent, le problème du stable maximum est $NP-complet$.

CHAPITRE 3

LES GRAPHES PARFAITS

Les graphes parfaits ont été introduits par Claude Berge au début des années 1960 à la suite d'un cheminement assez complexe qu'il raconte lui même dans un article agréable [16]. Pour une synthèse complète des recherches sur les graphes parfaits nous renvoyons aux références : un ouvrage collectif, publié sous la direction de L. Ramírez Alfonsín et B. Reed [17] et un article de M. Chudnovsky, N. Robertson, P. Seymour et R. Thomas [6]. L'introduction des graphes parfaits est motivée par la théorie de la complexité. Le problème qui nous occupe est celui de la coloration de graphes, formulé ci-dessous comme un problème de décision :

Etant donné un graph $G = (V, E)$. On définit une k-coloration propre de G comme une application $c : V \longrightarrow \{1, 2, ..., k\}$ qui associe à chaque sommet de G une couleur tel que pour toute arête $uv \in E$, on a $c(u) \neq c(v)$, c'est-à-dire deux sommets adjacents ne peuvent pas être colorés par une même couleur. Pour $i = 1, 2, ..., k$, $c^{-1}(i)$ est un sous-ensemble de sommets ayant une même couleur, donc qui sont deux à deux non adjacent ; ainsi, une k-coloration partitionne l'ensemble des sommets en k stables $V_1, V_2, ..., V_k$. Les ensembles V_i sont appelés les classes de couleur de la coloration.
On appelle nombre chromatique le plus petit k pour lequel G est k- colorable, noté $\chi(G)$. Il vient que un graphe 1-colorable est un graphe sans arête, donc un graphe à un seul sommet ; un graphe 2-colorable est un graphe biparti ; tout graphe complet K_n est n-colorable. Une coloration optimale est celle qui utilise le moins de couleur possible :
PROBLÈME DE LA COLORATION DES SOMMETS
Entrée : Un graphe G et un entier k.

Question : Existe-t-il une k-coloration de G ?

Ce problème est connu pour être NP-complet (Karp, [4]).

Il est clair que si un graphe G contient une clique de taille k, il faut au moins k couleurs pour le colorier. Donc, pour tout graphe on a : $\chi(G) \geqslant \omega(G)$. Il est donc naturel de se demander s'il y a toujours égalité entre le nombre chromatique et la taille d'une plus grande clique. La figure 3.1 montre que ce n'est pas le cas. Le trou C_5 présenté dans cette figure vérifie $\omega = 2 < \chi = 3 = \Delta(C_5) + 1$.

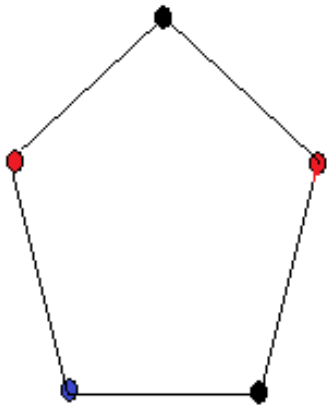

FIGURE 3.1 – Le nombre chromatique

Plus généralement, n'importe quel trou impair C_{2p+1} vérifie $\chi(C_{2p+1}) = 3$ et $\omega(C_{2p+1}) = 2$. Alain Ghouila-Houri [28] a également remarqué que n'importe quel anti-trou impair C^c_{2p+1} vérifie $\chi(C^c_{2p+1} = p + 1$ et $\omega(C^c_{2p+1}) = p$. Donc, l'égalité $\chi = \omega$ est fausse en général, c'est à partir de là que Claude Berge s'est intéressé aux graphes vérifiant cette égalité.

Définition 3.1. Un graphe parfait (perfect graph) G est un graphe verifiant $\chi(H) = \omega(H)$ pour tout sous-graphe induit H de G.

Il est aujourd'hui bien connu que le complémentaire d'un graphe parfait est parfait (*conjecture faible* des graphes parfaits formulée par *C. Berge* et démontrée par *Lovász*). Dans son étude des graphes parfaits, Berge a été amené à définir une classe de graphes qui porte désormais son nom :

Définition 3.2. Un graphe de Berge (Berge graph) est un graphe sans trou impair ni anti-trou impair.

Berge avait conjecturé que la classe des graphes parfaits coïncide avec la classe des graphes de Berge : c'était la fameuse conjecture forte des graphes parfaits , démontrée en 2002 :

Théorème 3.1. *(Chudnovsky, Robertson, Seymour et Thomas (2002) [6]) Un graphe est parfait si et seulement s'il est de Berge.*

A titre d'exemple, nous avons le résultat suivant :

Proposition 3.2 (18)**.** *Les graphes bipartis sont parfaits.*

A la fin de l'année 2002, la principale question ouverte sur les graphes parfaits restait celle de leur reconnaissance en temps polynomial. Ce problème a été finalement résolu par Maria Chudnovsky, Gerard Cornuejols, Xinming Liu, Paul Seymour et Kristina Vu.kovi. [19, 20, 21].

3.1 Graphe triangulé

Définition 3.3. Un graphe est cordal ou triangulé (chordal graph) s'il ne contient pas de trou.

Autrement dit, dans un graphe triangulé tout cycle de longueur au moins 4 contient une corde (une arête joignant deux sommets non consécutifs du cycle), i.e. les seuls cycles induits sont des triangles.

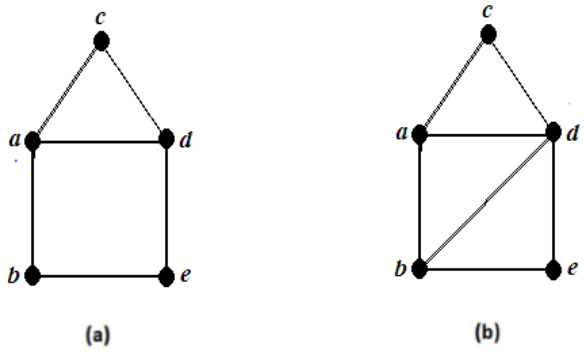

FIGURE 3.2 – Graphe triangulé

La figure 3.2 présente deux graphes : dans le graphe de (a), le cycle (a, b, e, d) est un trou donc ce graphe n'est pas triangulé. Par contre les cycles élimentaire du graphe de (c) sont tous des triangles, donc ce graphe est triangulé.

Définition 3.4. Un propriété P dans un graphe G est dite héréditaire si elle est conservée par tous les sous graphe induit de G.

Remarque 3.1. *Il est évident que si tous les cycles d'un graphe G sont des triangles, alors les cycles de tout sous graphe induit de G sont aussi des triangles. Ainsi la propriété d'être un graphe triangulé est une propriété héréditaire, c'est-à-dire tout sous graphe induit d'un graphe triangulé est triangulé.*

Proposition 3.3. *Les graphes triangulés sont parfaits.*

Démonstration. Un anti-trou d'ordre 5 est un trou d'ordre 5 ; et tout anti-trou d'ordre 6 ou plus contient un trou d'ordre 4, donc si G est triangulé, il ne contient pas de " grand" anti-trou. Donc il est de Berge, et donc parfait. □

Définition 3.5. Un sommet v est dit simplicial si son voisinage $N(v)$ est une clique.

Un célèbre théorème de Dirac [22] affirme qu'on peut toujours trouver un sommet simplicial dans un graphe cordal (et même deux, si le graphe n'est pas une clique).

Théorème 3.4. *(Dirac [22]) Tout graphe triangulé G autre qu'une clique contient au moins deux sommets simpliciaux non adjacents.*

Définition 3.6. Un ordre d'élimination simplicial (perfect elimination ordering) d'un graphe est un ordre $v_1 \dots v_n$ sur les sommets, tel que v_i est un sommet simplicial dans le graphe induit par les sommets $\{v_i, \dots, v_n\}$.

Fulkerson et Gross ont donné la caractérisation suivante des graphes cordaux :

Théorème 3.5. *(Fulkerson et Gross (1965) [23]) Un graphe est cordal si et seulement s'il admet un ordre d'élimination simplicial.*

Rose, Tarjan et Lueker ont introduit dans [9] un algorithme appelé parcours en largeur lexicographique (ou Lex-BFS pour Lexicographic Breadth-First Search) qui permet de trouver efficacement un ordre d'élimination simplicial d'un graphe cordal, et qui permet par conséquent de déterminer rapidement si un graphe est cordal ou non.

Théorème 3.6. *Un graphe est triangulé si et seulement si il admet uns schéma d'élimination simplicial.*

Algorithme. *Algorithme lex-BFS :*
donnée : *Un graphe $G = (V, E)$.*
Résultat : *Un ordre σ des sommets de G.*

1. *Pour chaque sommet $x \in V$ faire :*
   marque $(x) \leftarrow \varnothing$
2. *Pour $i = n$ à 1 faire :*
   - *Choisir un sommet x non numéroté de marque maximum dans l'ordre lexicographique ;*
     $\sigma(i) \leftarrow x$
   - *Pour chaque voisin non numéroté y de x faire :*
     $marque(y) \leftarrow marque(y) \cup i$

Lex-BFS calcul un ordre d'élimination simplicial pour un graphe triangulé avec une complexité temporelle $O(n + m)$.

Exemple. *L'ordre obtenu dans la figure 3.3 par l'algorithme Lex-BFS est* $\sigma = (1, 2, 3, 4, 5, 6)$. *Nous pouvons vérifier facilement que σ est un ordre d'élimination simplicial pour le graphe G et donc le graphe G est triangulé.*

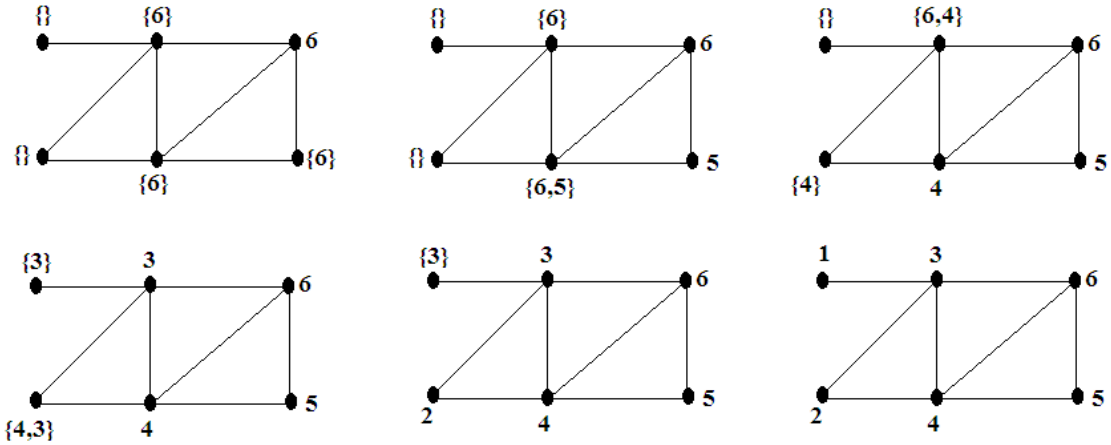

FIGURE 3.3 – Exécution de Lex-BFS

3.2 Graphe d'intervalle

Définition 3.7. Un graphe $G = (V, E)$ est d'**intervalles** (interval graph) s'il existe une famille de n intervalles $I = \{I_1, I_2, ..., I_n\}$ dans $\mathbb{R}$ et une bijection de V dans I qui, à tout $v \in V$, fait associer un intervalle $I_v \in I$ telles que $uv \in E$ si et seulement si $I_u \cap I_v \neq \emptyset$. Autrement dit, G est le graphe d'intersection d'un ensemble d'intervalles de la droite réelle, c'est-à-dire chaque sommet représente un intervalle et une arête relie deux sommets lorsque les intervalles correspondants s'intersectent. La figure 3.4 montre un graphe d'intervalle.

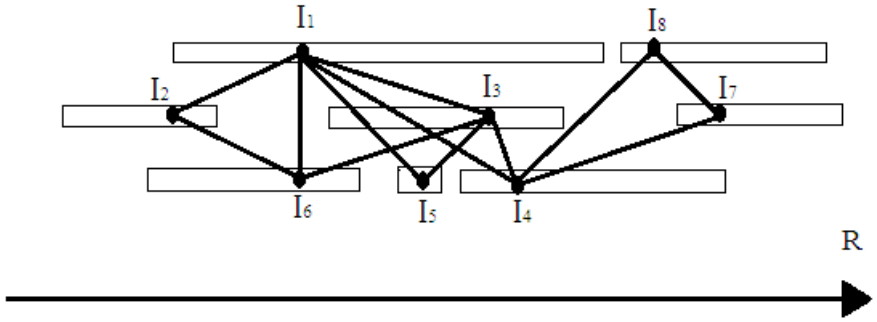

FIGURE 3.4 – La représentation d'un graphe d'intervalles

A partir de la définition ci-dessus, on peut établir la caractérisation suivante :

Proposition 3.7. *Un graphe est d'intervalles si et seulement s'il est possible d'ordonner toutes ses cliques maximales $C_1, C_2, \ldots, C_k$ de sorte que pour tout sommet $v \in C_i \cap C_k$, $v \in C_j, \forall i \leq j \leq k$ $i \neq k$.*

Plusieurs algorithmes efficaces ont été proposés pour reconnaître les graphes d'intervalles [24], [25]. Enfin, notons que :

Proposition 3.8. *Les graphes d'intervalles sont cordaux, et donc parfaits.*

3.3 Graphe de comparabilité

Définition 3.8. Un graphe $G = (V, E)$ est un graphe de **comparabilité** s'il existe une orientation O transitive et anti-symétrique des arêtes de G, c'est-à-dire pour toute arête $uv \in E$, soit $(u, v) \in O$, soit $(v, u) \in O$; et si $(u, v) \in O$ et $(v, w) \in O$ alors $(u, w) \in O$.

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

Il est facile de voir que le graphe obtenu après l'orientation d'un graphe de comparabilité est un graphe sans circuit dont la réduction transitive est un diagramme de Hass d'une certaine relation d'ordre définie sur V.

Proposition 3.9. *Un graphe d'intervalles est le graphe complémentaire d'un graphe de comparabilité.*

FIGURE 3.5 – Graphe de comparabilité

Démonstration. Soient $I = \{I_1, \ldots, I_n\}$ l'ensemble des intervalles associés à un graphe d'intervalles $G = (V, E)$ et x, y deux sommets non adjacents dans G (et donc x et y sont adjacents dans G^c). Donc $I_x \cap I_y = \emptyset$, orientons l'arête xy de x vers y si et seulement si I_x est à gauche de I_y. Ceci définit une orientation transitive de G. □

Corollaire 3.10. *Les graphes de comparabilité sont parfaits.*

Démonstration. D'après le Théorème 3.1, un graphe est parfait si et seulement si sont complémentaire est parfait. Puisque un graphe de comparabilité est un complémentaire d'un graphe parfait, alors il est parfait. □

L'algorithme ci-dessous donne une reconnaissance des graphes de comparabilité en temps polynomiale.

Algorithme. Algorithme permettant de déterminer si $G = (V, E)$ est un graphe de comparabilité

1. $F = E$
2. *Tant* $F - E \neq \varnothing$, *faire :*
   Choisir une arête e dans $E - F$, *donner une orientation à e et propager cette orientation pour assurer une orientation transitive de* G.
   Si une arête doit être orientée dans les deux sens : STOP, G *n'est pas de comparabilité. Sinon, rajouter à* F *toutes les arêtes nouvellement orientées. Si* $F = E$ *alors STOP :* G *est de comparabilité.*

CHAPITRE 4

QUELQUES PROBLÈMES D'INVARIANTS

On dit pour deux graphes $G = (V(G), E(G))$ et $H = (V(H), E(H))$ qu'ils sont isomorphes s'il existe une bijection f définie de $V(G)$ dans $V(H)$ telle que pour tout $u, v \in V(G)$, $uv \in E(G)$ si et seulement si $f(u)f(v) \in E(H)$. On appelle invariant d'un graphe G tout paramètre associé à G qui n'est pas modifié par un isomorphisme de graphes. Par exemple, le nombre chromatique $\chi(G)$, la taille d'un couplage maximum, $\omega(G)$ et $\alpha(G)\ldots$ sont des invariants de graphe. La détermination des invariants $\omega(G)$, $\chi(G)$ et $\alpha(G)$ d'un graphe G est un problème NP-complet. Cela signifie que tout problème NP peut se réduire de manière polynomiale au calcul d'un des paramètres ci-dessus. Donc il est peu probable (excepté si $P = NP$) que des algorithmes polynomiaux existent pour calculer $\omega(G), \chi(G)$ et $\alpha(G)$. Or tous ces paramètres peuvent être calculés en temps polynomial pour les graphes parfaits, ce qui rend cette classe extrêmement intéressante du point de vue algorithmique.

4.1 Problème de couplage dans un graphe biparti

Définition 4.1. Un **couplage** (*matching*) dans un graphe $G_(V, E)$ est un ensemble d'arêtes M deux à deux non adjacentes. Un sommet v qui est incident à une arête de M est dit *saturé* par le couplage ; sinon, il est dit **insaturé**. Un couplage qui sature tous les sommets de G est un couplage **parfait (perfect matching)**.

Notons qu'un couplage parfait n'existe pas toujours, car un graphe d'ordre impair ne peut avoir de couplage parfait, puisqu'un couplage couvre clairement un nombre pair de sommets ; il est maximum et il peut ne pas être unique (voir la figure 4.1). Le problème consiste à trouver un

couplage maximum :

PROBLEME DU COUPLAGE MAXIMUM (*Maximum Matching*)

Instance : Un graphe $G = (V, E)$

Objectif : Déterminer un couplage maximum de G.

On note $\nu(G)$ la taille d'un couplage maximum de G.

Il existe une différence fondamentale en théorie des graphes entre les termes maximal et maximum, très bien illustré sur les couplages :

- Un couplage est dit maximal s'il est impossible d'ajouter une arête au couplage, c'est-à-dire on ne peut pas trouver un autre couplage le contenant.
- Un couplage est dit maximum s'il est de cardinalité maximale, c'est-à-dire s'il est impossible de trouver un couplage de taille plus importante dans le graphe (contenant plus d'éléments).

Dans la figure 4.1, (1) $M_1 = \{ah, bj, ef, cd\}$ et $M_2 = \{bf, aj, ch, de\}$ sont deux couplages parfaits, donc, ils sont maximums. Dans la figure 4.1(2) $M = \{ac, be\}$ est un couplage maximal.

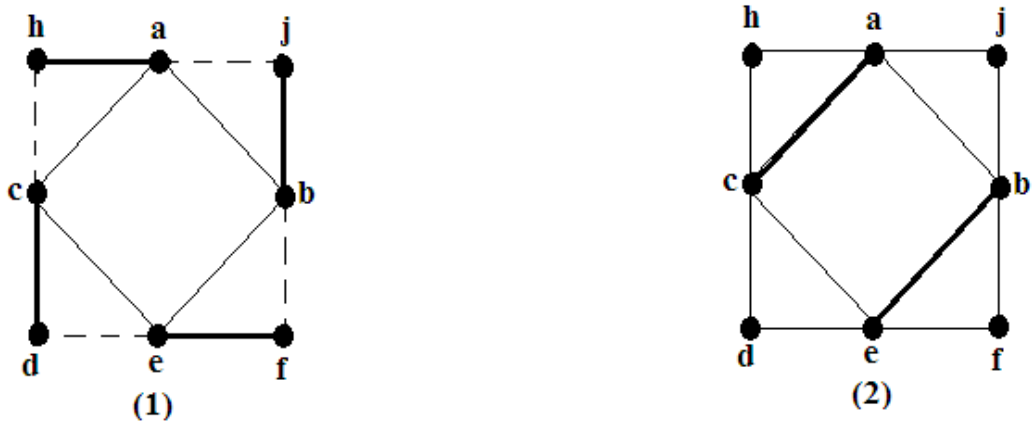

FIGURE 4.1 – Couplage parfait et transversal minimum

Par définition du graphe adjoint $L(G)$, un couplage dans G est un stable dans $L(G)$, donc $\alpha(G) = \nu(L(G))$. Il y a beaucoup de questions pratiques qui, une fois traduites dans le langage de théorie des graphes, reviennent à trouver un couplage maximum dans un graphe. Une d'entre elles est la suivante :

Exemple. *(tiré de* Claude Berge*)[5]*

PROBLEME D'AFFECTATION DU PERSONNEL.

Dans une organisation utilisant p ouvriers $x_1, x_2, ..., x_p$ et q postes de travail $y_1, y_2, ..., y_t$. Chaque ouvrier est qualifié pour un ou plusieurs de ces postes. Est-il possible d'affecter chacun à un poste

pour lequel il est qualifié ? Si l'on désigne par $E = \{xy$ tel que l'ouvrier x est califié pour le poste $y\}$), le problème revient à considérer un graphe biparti $G = (X, Y, E)$ et à chercher si le couplage maximum sature tous les sommets de X.

Définition 4.2. On dit qu'une chaîne μ est M-alternée si ses arêtes appartiennent alternativement à M et $E \setminus M$.

Le théorème suivant, dû à *Berge (1957)*, souligne l'importance des chaînes augmentant pour l'étude des couplages maximums.

Théorème 4.1. *(Claude BERGE [16]) Un couplage M est maximum si et seulement s' il n'existe pas de chaîne alternée dont les deux extrémités soient insaturés par M.*

Dans la figure 4.1(1) précédente, $M = \{ac, be\}$ est un couplage maximal qui ,'daprès le théorème de *BERGE*, n'est pas maximum, car $\mu = \{a, c, d, h\}$ est une chaîne alternée dont les extrémités sont insaturées par M.
Dans beaucoup d'applications, on désire trouver un couplage dans un graphe biparti $G[V_1, V_2]$ qui sature tous les sommets de V_1. Une conditions nécessaire et suffisante pour l'existence d'un tel couplage a été donnée par *Hall (1935)* :

Théorème 4.2. *(*Hall (1935)*)[26] Un graphe biparti $G = (V_1, V_2, E)$, où $|V_1| = |V_2|$, a un couplage parfait si et seulement si $|X| \leq |N(X)|$ pour tout $X \subset V_1$.*

Un **transversal** (*vertex cover*) est un ensemble de sommets T tel que toute arête de G ait au moins une de ses extrémités dans T. Le problème consiste à trouver un transversal minimum :
Instance : Un graphe $G = (V, E)$
Objectif : Déterminer un transversal minimum de G.
Le parametre $\tau(G)$ désigne le nombre d'elements dans un transversal minimum de G.

Remarquons que $T \subset V$ est un transversal si et seulement si $V \setminus T$ est un stable. En effet, si $u, v \in V \setminus T$, alors $uv \notin E$ sinon, on auraient forcement $u \in T$ ou $v \in T$, car T est un transversal. D'où $V \setminus T$ est stable. Réciproquement, si $V \setminus T$ est un stable, alors $\forall uv \in E$, $u \notin V \setminus T$ ou $v \notin V \setminus T$, ce qui implique que $u \in T$ ou $v \in T$, c'est-à-dire T est un transversal. De plus, le théorème ci-dessous montre que le problème du stable maximum et celui du transversal minimum sont équivalents en terme de difficulté :

Théorème 4.3. *(*Gallai (1959*) Pour tout graphe G d'ordre n, $\tau(G) + \alpha(G) = n$.*

Remarquons que deux arêtes d'un couplage ne peuvent pas êtres incidentes à un même sommet, donc un transversal dans G doit avoir au moins $\nu(G)$ éléments, d'où l'inégalité $\tau(G) \geqslant \nu(G)$. Cette inégalité peut être stricte (il suffit de considérer le triangle K3). Cependant, *König* [27] a demontré que dans un graphe biparti, les deux invariants sont toujours égaux :

Théorème 4.4. *(*König (1931), Egervary (1931)*) Pour tout graphe biparti, la cardinalité d'un couplage maximum est égale à la cardinalité d'un transversal minimum :* $\tau(G) = \nu(G)$.

Du théorème 4.1 : On déduit un algorithme de façon assez naturelle : on part d'un couplage M réduit à une arête, et tant qu'on arrive à trouver une chaîne alternée augmentante (relativement à M), on augmente la taille du couplage en effectuant un transfert. L'algorithme prend fin quand il n'existe plus de chaîne alternée augmentante. Il peut cependant être assez technique de trouver une chaîne alternée augmentante dans un graphe quelconque. Dans ce mémoire, nous nous intéresserons uniquement au cas où le graphe est biparti. Comme vu précédemment, il suffit de savoir détecter si une chaîne alternée augmentante (relativement à un couplage) (notée CAA par la suite) existe pour pouvoir soit augmenter la cardinalité du couplage (si cette chaîne existe), soit prouver que le couplage est maximum. L'algorithme qui suit permet de résoudre un tel problème.

Dans cet algorithme, le principe est de partir d'une extrémité potentielle d'une CAA. Pour cela, tous les sommets non saturés d'une partie du graphe biparti (soit X soit Y, le choix est arbitraire) sont stockés dans . Tant qu'aucune CAA n'a été trouvée avec un sommet de comme sommet initial, on itère sur P. Pour trouver une CAA à partir d'un sommet p , on parcourt les voisins non marqués de p. Si un de ces voisins n'est pas saturé, on a une CAA (réduite à une arête). Sinon, pour tout voisin y saturé et non marqué, on cherchera une CAA passant par les arêtes py et yz où yz est une arête du couplage. On stocke donc z dans une file d'attente F et on essaiera de trouver une CAA à partir de ce sommet.

Exemple.

Algorithme 16 *Algorithme d'une chaîne alternée augmentante*

Donnée : *Un graphe biparti* $G = (X \cup Y, E)$ *et un couplage* C

$P \longleftarrow$ *tous les sommets non saturés de* X

//perd : ***tableau des prédécesseurs pour la chaîne alternée augmentante***

Initialiser $Pred[i]$ *à 0 pour tout sommet* i

tant que $P \neq \varnothing$ ***faire***

Prendre un sommet p *de* P *et l'enlever de* P

//F ***est une file d'attente***

$F \longleftarrow \{p\}$

Marquer p

tant que $F \neq \varnothing$ ***faire***

Prendre le premier sommet x *de* F *et l'enlever de* F

pour *tout voisin* y *non marqué de* x ***faire***

si y *est saturé par une arête* $[y, z]$ *de* C ***alors***

Marquer y *et* z

$F \longleftarrow F \cup z$

$Perd[y] = x$

$Perd[z] = y$

// Chaîne alternée augmentante trouvée

sinon

$Perd[y] = x$

Fair un transfert sur la chaîne allant de p *à* y

retourner $Couplageaugment$

retourner $Couplagenonaugment$

4.2 Coloration d'un graphe triangulé

L'algorithme *COLOR*

L'algorithme *COLOR* est l'algorithme le plus naturel permettant de colorier les sommets d'un graphe. Etant donné un graphe G, cet algorithme parcourt les sommet selon cet ordre et donne à chaque sommet la plus petite coleur non attribuée à ses voisins. Voici une description formelle de l'algorithme :

ALGORITHME COLOR

Entrée : un graphe G avec n sommets et un ordre σ sur ses sommets.

Sortie : une coloration des sommets de G.

CALCUL :

Pour $i = 1, \ldots, n$

- Choisir le sommet non colorier x qui est minimum pour σ ;
- Colorier x par la plus petite coleur non présente dans son voisinage.

COMPLEXITE : $O(n + m)$.

Si l'on applique cet algorithme sur un P_4 ordonné comme la figure 3.2(a), on obtient une 2-coloration, ce qui correspond à une solution optimale puisqu ?un P_4 est un graphe biparti. En revanche, si l ?on exécute le même algorithme sur un P_4 ordonné comme dans la figure 3.2(b), la solution obtenue est une 3-coloration, et n ?est donc pas optimale : Evidemment, tous les graphes

FIGURE 4.2 – Algorithme *COLOR*

admettent un ordre tel que l'algorithme glouton donne une coloration optimale (puisque cette coloration existe). Chvátal [87] a alors posé la question suivante : quels sont les graphes pour lesquels il existe un ordre parfait sur les sommets, c'est-à-dire un ordre tel que l'algorithme *COLOR* donne une coloration optimale non seulement pour le graphe, mais également pour chacun de ses sous-graphes induits, et a prouvé que la seule obstruction à un ordre parfait est le P_4 décrit ci-dessus. Autrement dit :

Proposition 4.5. *(Chvátal (1984) [9]) Un graphe G est parfaitement ordonnable (perfectly orderable graph) (i.e. il admet un ordre parfait) si et seulement s'il existe un ordre $<$ sur les sommets tel que G ne contienne pas de P_4 $a-b-c-d$ avec $a<b$ et $d<c$.*

Algorithme de Welsh et Powell

Il s'agit maintenant d'utiliser l'algorithme de coloration séquentiel avec un ordre judicieux, en vue d'obtenir une coloration propre la plus "acceptable" possible. L'algorithme de Welsh et Powell consiste ainsi à colorer séquentiellement le graphe en visitant les sommets par ordre de degré décroissant. L'idée est que les sommets ayant beaucoup de voisins seront plus difficiles à colorer et donc il faut les colorer en premier. Cet algorithme couramment utilisé permet d'obtenir une assez bonne coloration d'un graphe,c'est-à-dire une coloration n'utilisant pas un trop grand nombre de couleurs, mais il n'assure pas que le nombre de couleurs utilisé soit minimum (et donc égal au nombre chromatique). On parle d'heuristique dans le sens où l'algorithme fourni une réponse approchée au problème de coloration.
Nous allons décrire ci-après l'algorithme de coloration de Welsh et Powell.

1. On classe d'abord les sommets du graphe dans l'ordre décroissant de leur degré On obtient ainsi une liste $x_1, x_2 \dots x_n$ de sommets telle que $deg(x_1) > deg(x_2) \dots > deg(x_n)$
2. On choisit une couleur c1 pour le sommet x_1, et :
   - en parcourant la liste dans l'ordre,on attribue la couleur c_1 au premier sommet non colorié et non adjacent à x_1
   - en continuant à parcourir la liste dans l'ordre,on attribue la couleur c_1 aux autres sommets non coloriés et non adjacents aux sommets déjà coloriés avec c_1 et, ce,jusqu'à la fin de la liste.
3. S'il reste des sommets non coloriés,on attribue une nouvelle couleur au premier sommet non colorié et on reprend la démarche.
4. On s'arrête dès que tous les sommets ont été coloriés.

Algorithme de Welsh et Powell
Entrée- Un graphe triangulé G=(V,E)
Sortie-Une coloration propre c : V $\to N$
1 : $n := |V|$;

2 : $Deg := tableau(Taille : n)(Defaut : 0)$;

3 : $sommet.(i) := Tableau(Taill : n)(Dfaut : 0)$;

4 : **pour** i de 0 à $n-1$ **faire**

5 : $Deg.(i) \leftarrow d(i)$;

6 : sommet.$(i) \leftarrow i$;

7 : **Fin pour** ;

8 : Tri (Tableau :sommet) (Relation d'ordre : $a \leq b$ ssi $deg.(a) \geq deg.(b)$) ;

9 : Coloration séquentielle(Graphe :G)(Numérotation : Sommet)

Si on utilise par exemple un tri par dénombrement (en $O(n)$, car les degrés de deux sommets ne peuvent différer de plus de n), on obtient un algorithme de complexité $O(m+n)$. Si on utilise plutôt un tri par comparaison,l'algorithme est alors de complexité $O(n\log(n)+n)$.

Exemple

Considérons le graphe G dessiné ci-dessous. Appliquons l'algorithme décrit ci-dessus : **2 ; 3 ; 4 ; 1 ; 0 ; 5** est une liste des sommets classés dans l'ordre décroissant de leurs degrés.

D'après l'algorithme :

- à la première étape on attribue une couleur**c1** aux sommets **2** et **5**
- à la deuxième étape,on attribue une couleur **c2** aux sommets **3** et **0**
- à la troisième étape,on attribue une couleur **c3** aux sommets **4** et **1**

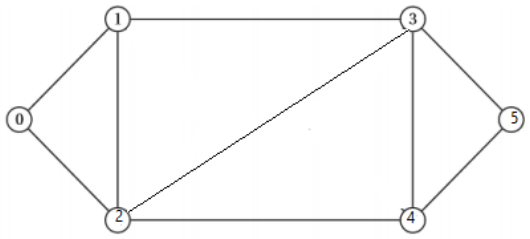

FIGURE 4.3 – Graphe G

$sommets(degr)$	2(4)	3(4)	4(3)	1(3)	0(2)	5(2)
$tape1$	$c1$					$c1$
$tape2$		$c2$			$c2$	
$tape3$			$c3$	$c3$		

On obtient un coloriage de G avec 3 couleurs, mais en fait le nombre chromatique de G est 3.

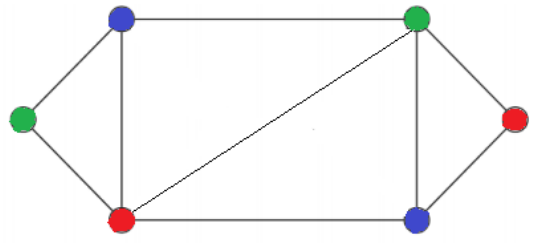

FIGURE 4.4 – Graphe G coloré

Remarque : Cette méthode peut aboutir à la pire des colorations possibles, par exemple si le graphe G a la structure de couronne à n sommets (voir la figure 3.5 qui présente une coronne à 8 sommets), son nombre chromatique est 2 tandis que Welsh-Powell donne dans certains cas (selon l'ordre dans lequel sont rangés les sommets) une coloration utilisant $n/2$ couleurs !

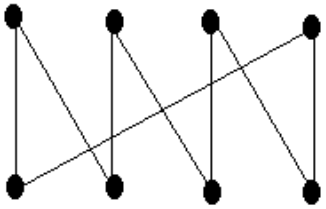

FIGURE 4.5 – Une coronne à 8 sommets

Définition 4.3. On appelle algorithme LexBFS-COLOR l'algorithme-*COLOR* (donné plus haut) appliqué à l'ordre inverse de celui calculé par l'algorithme lexBFS.

Théorème 4.6. *La coloration obtenue par l'algorithme lexBFS-COLOR sur les graphes triangulés est optimale.*

Exemple. *Considerons le graphe de la figure 4.3. Le tableau suivant donne l'ordre des sommets de G calculé par lex-BFS :*

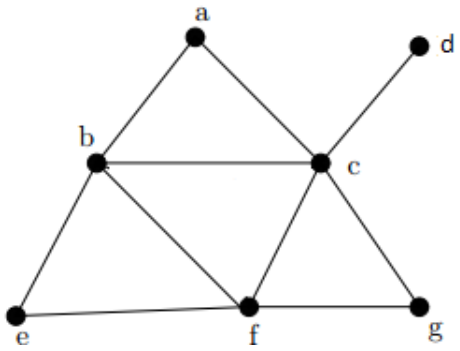

FIGURE 4.6 – Graphe triangulé G

σ	7	6	5	4	3	2	1
a	$\varnothing$		6		•6.4		
b	$\varnothing$	•7					
c	$\varnothing$		6	•6.5			
d	$\varnothing$				4	4	•4
e	•$\varnothing$						
f	$\varnothing$	7	•7.6				
g	$\varnothing$			5	5	•5.4	

On vérifie facilement que l'ordre $\sigma = (d, g, a, c, f, b, e$ des sommets de G est un ordre d'élimination simplicial, ce qui implique que G est triangulé.

Maintenant, on applique l'algorithme de coloration LexBFS-COLOR à l'ordre inverse de σ, c'est-à-dire à l'ordre $\sigma' = (e, b, f, c, a, g, d)$.

- *A la première étape on attribue la couleur **rouge** aux sommets **e** et **c**.*
- *A la deuxième étape,on attribue la couleur **jaune** aux sommets **b**, **g** et **d***
- *A la troisième étape,on attribue la couleur **bleu** aux sommets **f** et **a***

sommets	e	b	f	c	a	g	d
*tape*1	*rouge*					*rouge*	
*tape*2		*jaune*				*jaune*	*jaune*
*tape*3			*bleu*		*bleu*		

Tableau de coloration d'un graphe triangulé

Puisque G est triangulé, alors $\chi(G)$=3.

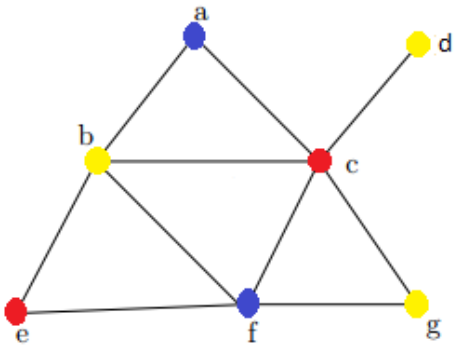

FIGURE 4.7 – Graphe colorée

L'algorithme LexBFS-COLOR ne donne pas une coloration optimale pour un graphe quelconque. En effet, le graphe $\bar{P}_6$ (le complementaire de P_6) présenté dans la figure ?est l'un des graphes pour lesquels la coloration obtenue par LexBFS-COLOR n'est pas optimale.

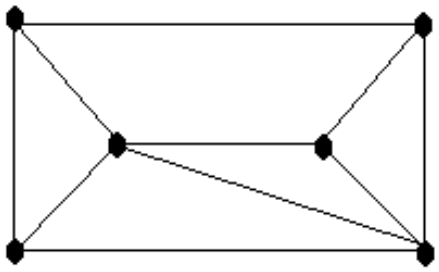

FIGURE 4.8 – Graphe $\bar{P}_6$

CHAPITRE 5

APPLICATIONS

5.1 Planification d'une formation

Une entreprise composée de neuf salaries : Ali, Lila, Amel, Omar, Sara, Samir, Adam, Karim et Amina. Le chef de cette entreprise décide de faire subir à ses salaries des formations dans des différentes spécialités : comptabilité, informatique, Droit , théorie des graphes, Marketing et Anglais. Le plan de formation est donné par le tableau suivant :

Salaries/Cours	Comptabilité	informatique	Droit	T.Graphes	Marketing	Anglait
Ali	x	x		x	x	
Lila		x		x	x	x
Amel	x		x	x	x	
Omar	x		x	x	x	
Sara		x		x	x	
Samir	x			x	x	
Adam				x	x	x
Karim			x	x	x	
Amina				x	x	

On suppose que chaque cours dure une heure et que les salaries n'ont pas les mêmes connaissances préalables. Le symbole "x" dans le tableau indique les formation dont le salarie a besoin par exemple Ali est concerné par les cours : comptabilité, informatique, théorie des graphe et marketing mais pas les autres cours. L'objectif est d'organiser une session de formation qui dure le moins longtemps possible de façons à ce que tout salarie suit les formations indiquées dans le plan.

La solution la plus évidente pour planifier tout ça est de programmer les cours l'un après l'autre. Ainsi la session de formation durera six heures puisque il y a six cours d'une heure. La question maintenant est de savoir si on peut faire mieux en tenant compte du fait que les formations dont un même candidat est concernées ne peuvent pas être programmées à la même heure.
Pour résoudre ce problème, associons le graphe d'incompatibilité $G = (V, E)$ tel que les sommet représentent les cours et deux sommet sont relies par une arête si et seulement si les deux cours ne peuvent pas avoir lieu au même temps. Le graphe de la figure 5.1 donne une modélisation de ce plan formation :

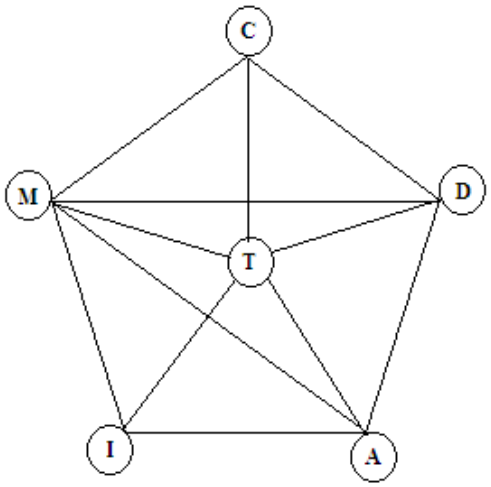

FIGURE 5.1 – Modélisation de la formation

D'après ce graphe on voit bien que le cours d'informatique et celui du droit peuvent être programmés au même temps car ils sont pas adjacents dans G. Ainsi il est naturel de penser qu'une partition minimale en stable donne un planning optimale pour le déroulement de cette formation, c'est-à-dire ce planning sera donné par le nombre chromatique de G.
Commençons par l'application de l'algorithme Lex-BFS sur G. La figure 5.2 montre l'ordre sur les sommets donné par cet algorithme.

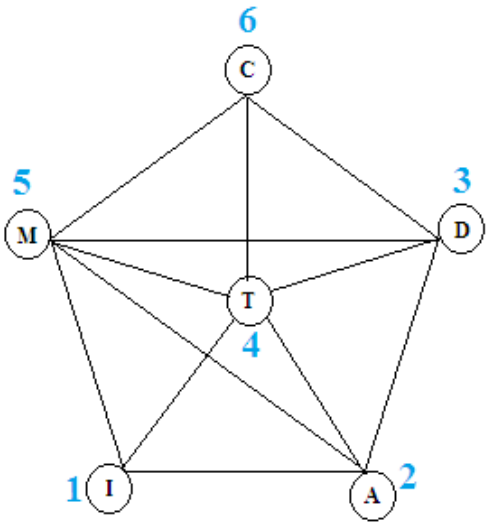

FIGURE 5.2 – L'ordre des sommets de G

On peut vérifier facilement que l'ordre des sommets de G est un ordre d'élimination parfait (simplicial), ce qui implique que le graphe G est triangulé. Par conséquent l'algorithme LexBFS-COLOR donne une coloration optimale pour les sommets de G comme est montré sur la figure ?

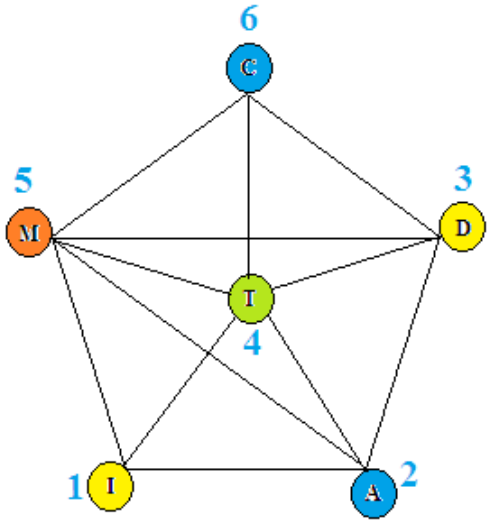

FIGURE 5.3 – Graphe triangulé

On voit bien que le nombre de couleurs minimum est quatre, c'est-à-dire $\chi(G) = 4$. A partir de la figure 5.3, on déduit un planning optimal de la manière suivante :
A la première heure on programme le cours de la Comptabilité et le cours d'Anglais, à la deuxième

heure nous pouvons mettre le cours de Marketing qui Intéresse tout le monde, à l'heure suivant correspond le cours de la Théorie des Graphe qui intéresse aussi tout le monde et enfin nous avons le cours d'informatique et celui du droit.

5.2 Problème d'affectation

Le problème d'affectation est une généralisation du problème de couplage maximum dans un graphe biparti, puisqu'il s'agit de trouver, parmi les couplages de cardinal maximal, celui qui a le poids minimum (ou maximum). Ce problème s'appelle problème d'affectation, car l'exemple le plus classique est l'affectation de tâches n à m machines. On suppose qu'on connaît le coût c_{ij} de production d'une tâche i sur une machine j (ce coût est infini s'il n'est pas possible d'effectuer cette tâche sur cette machine). Le but est d'affecter toutes les tâches de sorte à minimiser le coût total de production (défini comme la somme des coûts). On peut modéliser ce problème par un graphe biparti pondéré $G = (X \cup Y, E, M)$ dans lequel X est l'ensemble des tâches, Y l'ensemble des machines, et les arêtes $ij \in E$ sont pondérées par le coût de production de la tâche i sur la machine j. Notons que le problème de couplage maximum dans un graphe biparti est un cas particulier du problème d'affectation, dans lequel toutes les arêtes existantes ont un coût unitaire. Dans ce qui suit, on supposera que :

1. Il y a autant de tâches que de machines, c'est-à-dire $m = n$. Si ce n'est pas le cas, il est toujours possible d'ajouter une machine (ou une tâche) fictive avec des coûts d'exploitation nuls.
2. Tous les poids sont positifs ou nuls. Si ce n'est pas le cas, on peut augmenter tous les poids d'une même valeur, cela ne changera pas la structure de la solution optimale.
3. Il est aussi possible de résoudre le problème d'affectation en maximisation (i.e. trouver, parmi les couplages de cardinal maximal, celui qui a le poids maximum). Il suffit de remplacer le poids de chaque arête par son opposé, d'ajouter le plus grand poids à toutes les arêtes, et de résoudre le problème en minimisation.

Pour résoudre ce probleme, nous utilisons l'algorithme hongrois ou méthode hongroise (parfois appelé aussi algorithme de kuhn). Cet algorithme calcule une affectation optimale avec une complexité temporelle qui est polynômiale. Il a été proposé en 1955 par le mathématicien américain *HAROLD Kuhn.*Le nom "*méthode hongroise*" parce qu'il s'appuyait sur des travaux antérieurs de deux mathématiciens : *D. Konig et J. Egervary.*

1. Réduction des lignes : trouver l'élément minimum dans chaque linge de la matrice $n \times n$, le soustraire de chaque ligne le coût minimum dans sa ligne.
2. Réduction des colonnes : pour cette nouvelle matrice, trouver le coût minimum dans chaque colonne. Construire une nouvelle matrice en soustrayant dans chaque colonne sont minimum.
3. Déterminer le nombre minimal de lignes nécessaires sur les lignes et les colonnes pour couvrir tout les zéros :
   - Si ce nombre est égal au nombre de ligne (ou colonnes), la matrice est réduite, aller à l'étape 5 ;
   - Si ce nombre est inférieur au nombre de ligne ;aller à l'étape 4.
4. Trouver la cellule de valeur minimum non-couverte par une ligne.
   - Soustraire cette valeur de toutes les cellules non couverte ;
   - Ajouter cette valeur aux cellules situées à l'intersection de deux ligne.retourner à l'étape 3.
5. Déterminer la solution optimale.

Soit c la matrice des coûts du problème d'affectation donnée, on réduit c de la manière suivant (pour avoir (x, y) réalisable du dual).

$$x_i = \min_{j=\overline{1.n}} C_{i,j}, \forall i = \overline{1.n}$$

$$y_j = \min_{i=\overline{1.n}} C_{i,j} - x_i, \forall j = \overline{1.n}$$

On obtient une matrice $\overline{c}$

On cherche un recouvrement minimum pour les zéros de $\overline{c}$

Soit R l'ensemble de ligne ou de colonne de recouvrement.

- Si $|R| = n$,la solution primal correspondant à $(x; y)$ est réalisable, donc ces deux solutions sont optimales (on arrête).

- Si $|R| < n$, on améliore la solution dual couvrant.
  Soit :

$$\sigma^0 = \min_{(i,j)\notin R} \overline{c}_{ij}$$

$$\overline{c'}(i,j) = \begin{cases} (\overline{c}_{ij}) - \sigma^0 \ non\ recouvert; \\ (\overline{c}_{ij}) + \sigma^0 \ couvert \quad par\ une\ ligne\ et\ une\ colonne; \\ (\overline{c}_{ij}) \qquad\qquad sinon. \end{cases}$$

Et on refait la même procédure pour $\overline{c'}_{ij}$.

Exemple. *Quatre produits P_1, P_2, P_3 et P_4 sont à affecter sur 4 machines dont les coûts d'exploitation dépendent du produit, comme indiqué sur le tableau. Sur quelle machine chaque produit doit-il être fabriqué pour minimiser le coût total ?*

produits/machines	t_1	t_2	t_3	t_4
p_1	3	2	2	3
p_2	1	1	4	2
p_3	4	2	5	3
p_4	3	2	4	4

Cela revient à chercher un couplage de taille 4 dans le graphe biparti $G = (P \cup T, E, C)$, représenté dans la figure 5.1 avec $P = p_1, p_2, p_3, p_4$ et $T = t_1, t_2, t_3, t_4$ de telle sorte à minimiser la somme des coûts des arêtes du couplage.

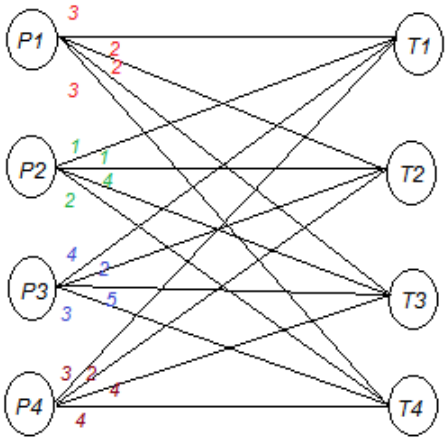

FIGURE 5.4 – Graphe biparti

Soit A la matrice d'affectation associée à ce problème :

$$A = \begin{pmatrix} 3 & 2 & 2 & 3 \\ 1 & 1 & 4 & 2 \\ 4 & 2 & 5 & 3 \\ 3 & 2 & 4 & 4 \end{pmatrix}$$

(1) *On fait apparaître sur chaque ligne et chaque colonne un zéro.*

$$A = \begin{pmatrix} 3 & 2 & 2 & 3 \\ 1 & 1 & 4 & 2 \\ 4 & 2 & 5 & 3 \\ 3 & 2 & 4 & 4 \end{pmatrix} \begin{matrix} -2 \\ -1 \\ -2 \\ -2 \end{matrix} \rightarrow \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 3 & 1 \\ 2 & 0 & 3 & 1 \\ 1 & 0 & 2 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 2 & 0 & 3 & 0 \\ 1 & 0 & 2 & 1 \end{pmatrix}$$

0 0 0 -1

$$\begin{pmatrix} 1 & \emptyset & [0] & \emptyset \\ [0] & \emptyset & 3 & \emptyset \\ 2 & \emptyset & 3 & [0] \\ 1 & [0] & 2 & 1 \end{pmatrix}$$

On obtient 4 zéros encadrés, donc 4 zéros indépendant.

Le nombre de zéros indépendants est égale à n=4. L'affectation est donc optimal

Le meilleur choix d'affectation est :

$p_1 \longrightarrow I_3$ *de de valeur : 2.*

$p_2 \longrightarrow I_1$ *de de valeur : 1.*

$p_3 \longrightarrow I_4$ *de de valeur : 3.*

$p_4 \longrightarrow I_2$ *de de valeur : 2.*

La valeur de l'affectation minimale est égale à la somme des a_{ij} *de la matrice de départ, Val(aff)=* $2 + 1 + 3 + 2 = 8$

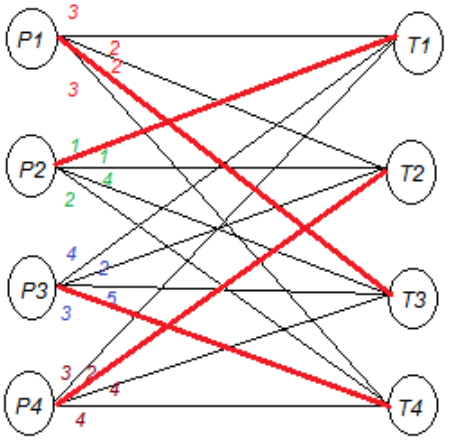

FIGURE 5.5 – Couplage minimum

CONCLUSION GÉNÉRALE

Ce travail présente à ses lecteurs une vision globale sur la théorie des graphes qui englobe unsujet très intéressant qui est l'invariant d'un graphe parfait.

On a pu constater que les graphes constituent une méthode qui permet de modéliser une grande variété de problèmes concrets en se ramenant à l'étude de sommets et des arcs. Donc c'est l'un des instruments le plus courants et les plus efficaces pour résoudre des problèmes discrets posés en recherche Opérationnelle(RO).

L'importance de la théorie des graphes vient aussi du fait qu'elle fournit un cadre conceptuel adéquat pour l'analyse et la résolution de nombreux problèmes. En effet elle s'est développé ausein de disciplines diverses, Planification d'une formation, , problèmes d'affectation,...,etc.

Notre travail consiste à donner un certain nombre d'outil (algorithme) de la théorie des graphes directement utilisables pour résoudre des problèmes qui peuvent être rencontrer.

L'exactitude de notre objectif consiste à résoudre des problèmes en utilisant les graphes parfaits, pour cela on a traité les deux problèmes suivants :

le premier consiste à résoudre un problème de Planification d'une formation en prenant compte la minimisation des heures en utilisant l'algorithme Lex-BFS(on cherche le nombre de couleurs minimum)

le deuxieme est le problème d'affectation des machines aux produits, afin de minimiser le coût total on utilise l'algorithme Hongrois pour la résolution.

BIBLIOGRAPHIE

[1] J. Orlin, Containment in Graph Theory : covering graphs with cliques, Nederl. Akad. Wetensch. lndag. Math. 39, (1977) 21l-218.

[2] S.A. Cook : The complexity of theorem-proving procedures. In Proceedings of the third annual ACM symposium on Theory of computing, pages 151-158. ACM, 1971. (Cité page 182.)

[3] L.A. Levin : Universal'nye perebornye zadachi (Universal search problems) (en russe). Problemy Peredachi Informatsii (Problems of In- formation Transmission), 9(3) :265-266, 1973. (Cité page 182.)

[4] R.Karp, Reducibility among combinatorial problems, complexity of computer computation 85-103 plenum pres 1972.

[5] C.Berge, Les problèmes de coloration en theorie des graphes, Publication du l'institut de statistique de l'université de Paris (1960) 123-160.

[6] M.Chudnovsky, N.Robertson, P.Seymour, R.Thomas, The strong perfect graph theorem, Annals of Mathematics 164, (2006) 51-266.

[7] M.Grostchel, L.Lovasz, A.Chrijver, The ellipsoid method and its consequence in combinatorial optimisation, Combinatorica 1(1981) 169-197.

[8] L.Lovasz, A characterisation of perfect graphs, Journal of combinatorial theory B13 (1972) 95-98.

[9] D.J.Rose, R.E.Tarjan, G.S.Lueker, Algorithmics aspects of vertex elimination of graphs, SIAM Journal on computing (1976) 266-283.

[10] D. Jungnickel, Graphs, Networks and Algorithms, Springer, Berlin Heidelberg New York Dordrecht London (2013).

[11] M.C. Golumbic Algorithmic Graph Theory and Perfect Graphs, Elsevier, Second Edition (2004).

[12] H. J. Bandelt and H.M. Mulder, Distance-hereditary graphs, J. Combin. Theory Ser. B 41 (1986)182-208.

[13] A. Brandstädt, V.B. Le., and J. Spinrad. Graph classes : a survey. SIAM, Philadelphia, (1999).

[14] H. Fleischner, E.Mujunib, D. Paulusmac, S. Szeider, Covering graphs with few complete bipartite subgraphs, Theoretical Computer Science 410 (2009) 2045-2053.

[15] L.W. Beineke : Characterizations of derived graphs. Journal of Com- binatorial Theory, 9(2) :129-135, 1970.

[16] C. Berge and J. L. Ramírez Alfonsín. Origins and Genesis. In Ramírez Alfonsín and Reed, pages 1-12 (1994).

[17] J. L. Ramírez Alfonsín and B. A. Reed, editors. Perfect graphs. Series in Discrete Mathematics and Optimization. Wiley-Interscience, 2001.

[18] G. Morel, Stabilité et coloration des graphes sans P_5, thèse de doctorat, l'Université de GRENOBLE(2011).

[19] M. Chudnovsky, G. Cornuéjols, X. Liu, P. Seymour, and K. Vu ?kovi ?. Cleaning for Bergeness. Manuscript, 2003.

[20] M. Chudnovsky and P. Seymour. Recognizing Berge graphs. Manuscript, 2002.

[21] G. Cornuéjols, X. Liu, and K. Vu ?kovi ?. A polynomial algorithm for recognizing perfect graphs. Manuscript, 2002.

[22] G.A. Dirac : On rigid circuit graphs. In Abhandlungen aus dem Mathematischen Seminar der Universit "at Hamburg, volume 25, pages 71-76. Springer, 1961.

[23] D.R. Fulkerson et O.A. Gross : Incidence matrices and interval graphs. Pacific J. Math., 15 :835-855, 1965.

[24] M. Habib, R. McConnell, C. Paul et L. Viennot : Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. Theoretical Computer Science, 234(1-2) :59-84, 2000.

[25] K.S. Booth et G.S. Lueker : Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. Journal of Computer and System Sciences, 13(3) :335-379, 1976.

[26] P. Hall : On representatives of subsets. Journal of the London Mathe- matical Society, 10 :26-30, 1935.

[27] D. König : Graphok és matrixok (Graphes et matrices). Matematikai és Fizikai Lapok, 38 :116-119, 1931.

[28] A.Ghouila-Houri : Sur la généralisation de la notion de commande d'un Systèm guidable,1,n4(1967),p.7-32.

[29] Garey et Johnson : Algorithme exacte et exponentiels pour les problème difficiale :domination,variantes et généralisation :07/12/2007.

Resumé

Nous avons vu que la classe des graphes parfaits était une classe de graphes importantes, en particulier parce qu'elle contient de nombreuses autre classes usuelles, dont les graphes d'intervalles, les graphes de comparabilité ou les graphes triangulés. On peut penser que la preuve des deux conjectures posées par Berge au début des années 1960 clôt la discussion au sujet des graphes parfaits, car cette classe est désormais trés bien connue. P. Seymour mentionne dans un premier axe de recherche, qui consisterait à améliorer la preuve du théorème fort des graphes parfaits. Un autre axe, qui semble a priori plus intéressant, serait d'obtenir un algorithme de construction explicite des graphes de Berge. Dans le même axe, on sait qu'il existe un algorithme de coloriage polynômial, à base de polyèdres et de programmation linéaire : mais existe-t-il un algorithme combinatoire, plus proche de la structure des graphes parfaits .

Mots-clés : parfaits ,comparabilité, triangulés...

Abstract

We have seen that the class of perfect graphs is a class of important graphs, especially because it contains many other usual classes, including interval graphs, comparability graphs, or triangulated graphs. One can think that the proof of the two conjectures posed by Berge at the beginning of the 1960s ends the discussion about the perfect graphs, because this class is from now on very well known. P. Seymour mentions in a first line of research, which would consist in improving the proof of the strong theorem of the graphs perfect. Another axis, which seems a priori more interesting, would be to obtain an explicit algorithm for constructing Berge graphs. In the same axis, we know that there is a polynomial coloring algorithm, based on polyhedra and linear programming : but is there a combinatorial algorithm, closer to the structure of perfect graphs.

Keywords : perfect, comparability, triangulated ...