كلية العلوم و التقنيات فاس
ⵜⴰⵖⵙⵍⵓⵜ ⵏ ⵜⴻⵛⵀⵓⵍⵓⵊⵢⴰⵜ ⴰ ⵜⵙⵙⵙⵜⵙ
Faculté des Sciences et Techniques de Fès

جامعة سيدي محمد بن عبد الله
ⵜⴰⵙⵙⵓⵍⵉⵜ ⵙⵙⴰⵙ ⵇⵙⴰⵇⵇⴰ ⵙⵏ ⵃⵙⴱⵏⵙⴼ
Université Sidi Mohamed Ben Abdellah

DEPARTEMENT DES MATHEMATIQUES

Master Mathématique et Application au Calcul Scientifique (MACS)

MEMOIRE DE FIN D'ETUDES

Pour l'obtention du Diplôme de Master Sciences et Techniques (MST)

Coloration des graphes et applications

Réalisé par : Abdelouahed ALLA HAMOU

Encadré par : Pr. Abdelmajid HILALI

Soutenu le 27 février 2018

Devant le jury composé de :

- Pr. Fatima EZZAKI	Faculté des Sciences et Technique de Fès
- Pr. Ahmed ELHILALI ALAOUI	Faculté des Sciences et Technique de Fès
- Pr. Abdelmajid HILALI	Faculté des Sciences et Technique de Fès
- Pr. Anisse OUADGHIRI	Faculté des Sciences et Technique de Fès

Année Universitaire 2017 / 2018

À mon père, ma mère, mes sœurs...

À ma grande famille et mes amis...

Je vous aime beaucoup.

Remerciements

Au terme de ce travail, je voudrais exprimer mes remerciements et ma profonde reconnaissance à tout ceux qui ont contribué de prêt ou de loin à sa réalisation :

Je voudrais, en premier lieu, exprimer ma profonde reconnaissance à mon encadrant, Professeur **Abdelmajid HILALI**, qui, grâce à ses orientations précieuses, ses encouragements valorisants, sa direction compétente de ce mémoire, son soutien dans les moments d'incertitude et sa disponibilité inconditionnelle, j'ai pu réaliser ce travail et s'initier à la recherche.

Je tiens également à exprimer toute ma gratitude et tout mon sentiment de reconnaissance aux professeurs **Fatima EZZAKI**, **Ahmed ELHILALI ALAOUI** et **Anisse OUADGHIRI** pour l'honneur qu'ils me font d'accepter d'être membres de jury de ce mémoire.

Je voudrais adresser une deuxième fois mes chaleureux remerciements au Professeur **Ahmed ELHILALI ALAOUI** coordonnateur du cycle Master Mathématiques et Applications aux calculs scientifiques. J'adresse mes sincères remerciements et ma grande reconnaissance à tous mes professeurs du cycle Master Mathématiques et Applications aux calculs scientifiques.

J'exprime ma gratitude à tous **mes collègues** du cycle Master Mathématiques et Applications aux calculs scientifiques pour leur soutien amical durant ces deux années d'étude.

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Introduction

L'histoire de la théorie des graphes débute peut-être avec les travaux d'Euler au 18ème siècle et trouve son origine dans l'étude de certains problèmes, tels que celui des ponts de Königsberg (les habitants de Königsberg se demandaient s'il était possible, en partant d'un quartier quelconque de la ville, de traverser tous les ponts sans passer deux fois par le même et de revenir à leur point de départ), la marche du cavalier sur l'échiquier ou le problème du coloriage de cartes et du plus court trajet entre deux points.

La théorie des graphes s'est alors développée dans diverses disciplines telles que la chimie, la biologie, les sciences sociales (réseaux de transports), gestion de projets, informatique (topologie des réseaux, complexité algorithmique, protocoles de transferts), la physique quantique, etc.

Depuis le début du 20ème siècle, elle constitue une branche à part entière des mathématiques, grâce aux travaux de König, Menger, Cayley puis de Berge et d'Erdös. Cette branche des mathématiques a connu un grand regain d'intérêt suite à l'émergence des réseaux sociaux Internet dont les connexions entre "amis" et "suiveurs" constituent des graphes dont l'analyse topologique et statistique peut nous apprendre de nombreuses choses.

De manière générale, un graphe permet de représenter la structure, les connexions d'un ensemble complexe en exprimant les relations entre ses éléments : réseau de communication, réseaux routiers, interaction de diverses espèces animales, circuits électriques.

Les graphes constituent donc une méthode de pensée qui permet de modéliser une grande variété de problèmes en les ramenant à l'étude de sommets et d'arcs.

Les derniers travaux en théorie des graphes sont souvent effectués par des informaticiens, du fait de l'importance que revêt l'aspect algorithmique dans leur domaine.

Effectivement, il s'agit essentiellement de modéliser des problèmes. Nous exprimons le problème en termes de graphes de sorte qu'il relève d'un problème de la théorie des graphes que nous savons le plus souvent résoudre car rentrant dans une catégorie de problèmes connus.

Le problème de coloration de graphes est l'un des origines de la théorie des graphes, ce problème remonte au 19 ème siècle lorsque Guthrie a remarqué que quatre couleurs étaient suffisantes pour colorer la carte d'Angleterre, sans donner la même couleur à deux régions ayant une frontière commune. Il pose alors la question de savoir si quatre couleurs suffisent toujours pour colorier n'importe quelle carte géographique de sorte que deux régions voisins n'aient pas la même couleur. Malgré un énoncé simple, cette conjecture est restée non résolue pendant plus d'un siècle. Il fallut attendre 1978 pour qu'Appel et Haken parviennent à démontrer ce résultat à l'aide d'un ordinateur. Même si le problème des cartographes est résolu, celui des mathématiciens ne l'est pas, car ce théorème traite seulement du cas particulier des graphes planaires.

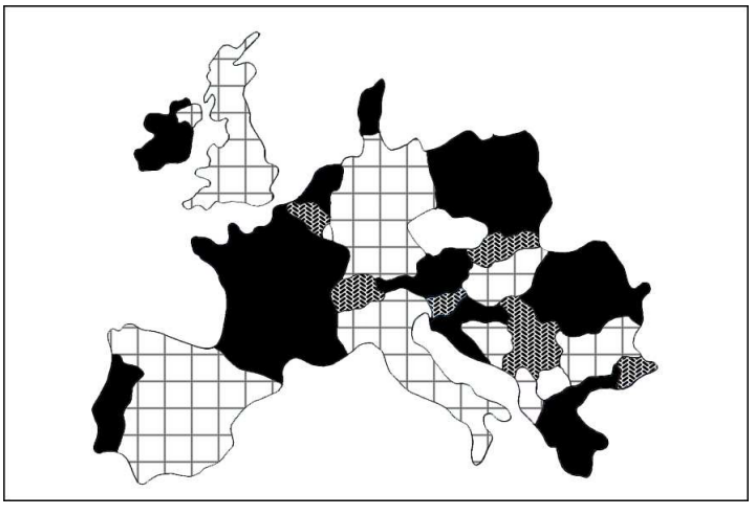


FIGURE 1 – Carte de l'Europe coloriée avec quatre couleurs .

Un exemple plus récent d'application du problème de coloration de graphes est celui de l'affectation de fréquence radio. Un opérateur de téléphonie dispose d'un certain nombre d'antennes sur territoire. Certaines de ces antennes sont trop proches pour pouvoir émettre sur la même fréquence. L'opérateur doit donc attribuer des fréquence aux différentes antennes qui interfèrent entre elles, tout en cherchant à minimiser le nombre total de fréquence utilisées. Il s'agit dans ce cas d'une coloration propre de sommet du graphe dont les sommets correspondent bijectivement aux antennes, avec une arête entre deux sommets si, et seulement si, les antennes correspondantes peuvent interférer.

De nombreuses variantes et extensions de colorations de graphes ont été considérées dans la littérature. Les colorations impropres (également connues sous le nom de colorations généralisées ou relaxées) sont une généralisation des colorations propres où la contrainte consistant à colorier de façon différente deux sommets adjacents est supprimée.

Voici un résumé des différents chapitres qui composent ce travail.

Le premier chapitre est consacré aux notions de base, nous donnons des définitions fondamentaux et rappels sur les notions de base en théorie des graphes, que nous utiliserons aux cours de ce mémoire. Nous donnons également un bref aperçu de la théorie de la complexité des algorithmes en insistant sur les notions fondamentales de classes de problèmes P et NP .

Le deuxième chapitre porte sur la coloration de graphes. Nous parlerons de graphe avec ses différentes manières ainsi que la caractérisation et la coloration des différentes classes de graphes à savoir les graphes planaires et les graphes parfaits.

Dans le troisième chapitre, nous présentons quelques algorithmes de reconnaissance de quelques graphes particulière ainsi les algorithmes permettant de colorier les graphes, ils seront utilisés par la suite dans le chapitre des applications. Chaque algorithme est illustré par un exemple

Dans le dernier chapitre nous présentons quelques applications de la coloration des graphes dans le monde réel, ainsi qu'une étude comparative des résultats expérimentaux obtenus avec les divers algorithmes appliqués à ces applications réelles.

Nous terminons notre travail par une conclusion générale, dans laquelle nous résumons tout ce que nous avons fait dans ce mémoire.

CHAPITRE 1

Notions de base

Dans ce premier chapitre, nous définissons les notions qui serviront dans la suite de ce document. Nous commençons par des notions et notations de théorie des graphes, ensuite nous introduisons quelques notions de théorie de la complexité.

1.1 Notions sur les graphes

1.1.1 Concept de base

Intuitivement un graphe simple G, est un ensemble de points ou sommets (que nous supposerons toujours non vide et fini dans la suite) dont certaines paires sont reliées, formant ainsi les extrémités d'une arête. Plus formellement :

Définition 1.1

Un graphe simple fini G est un couple (V_G, E_G) où :
V_G est un ensemble fini et $E_G \subseteq [V_G]^2$ où $[V_G]^2$ désigne l'ensemble des parties de V_G qui ont exactement 2 éléments.
Les éléments de V_G sont appelés **les sommets de** G, et les éléments de E_G sont appelés **les arêtes de** G .

Il est courant d'attribuer un nom ou un numéro aux sommets d'un graphe. Un tel procédé est appelé étiquetage. Ainsi, nous parlerons souvent du sommet u, ou du sommet v d'un graphe. Afin d'alléger les notations, la paire contenant u et v est notée uv, au lieu de $\{u, v\}$, vu représentant la même arête. .
Le nombre de sommets d'un graphe G est appelé **ordre de** G, égal donc à $|V_G|$, est plus généralement noté **n**, le nombre d'arêtes de G est appelé **taille de** G, est noté **m**.
Dans la suite tous les graphes que nous considérerons seront simples.

Voisinage et degré

Deux sommets u et v d'un graphe simple G formant les extrémités d'une même arête uv (ou vu), sont dit **adjacents** ou **voisins** , ce que l'on note $u \sim v$, et dans le cas contraire non-adjacents ou non-voisins ($u \nsim v$).
On définit alors le voisinage d'un sommet :

Définition 1.2

- Le **voisinage** d'un sommet v de G, noté $N_G(v)$, est l'ensemble des sommets qui lui sont adjacents.

Un sommet n'ayant aucun voisin est qualifié d'**isolé**.
On définit également le voisinage d'un ensemble de sommets :

Définition 1.3

Le **voisinage** (ouvert) d'un ensemble de sommets $S \subset V_G$, noté $N_G(S)$, est l'ensemble

$$\{u \in V_G \setminus S | \exists v \in S, uv \in E_G\}$$

Définition 1.4

Le **degré** d'un sommet v, noté $d_G(v)$, est égal au nombre $|N_G(v)|$.

Soit G un graphe simple, le plus petit degré d'un graphe G est noté $\delta(G)$ et le plus grand degré est noté $\Delta(G)$, plus formellement :

$$\delta(G) = \min_{v \in V_G} d_G(v)$$

et

$$\Delta(G) = \max_{v \in V_G} d_G(v)$$

Il est facile de démontrer le résultat suivant, que l'on trouve déjà dans l'article d'Euler sur les sept ponts de Königsberg :

Lemme 1.1

Soit $G = (V_G, E_G)$ un graphe simple de taille m, alors

$$\sum_{v \in V_G} d_G(v) = 2m.$$

Démonstration: Il suffit de constater que chaque arête compte pour deux degrés (un pour chacune de ses extrémités).

□

Définition 1.5

Soit k un entier, un graphe G est **k-régulier** si tout sommet est de degré k.
Un graphe 3-régulier est dit **cubique**.

Complémentaire

Quand on recherche les propriétés d'un graphe, il est parfois plus simple d'étudier son complémentaire :

Définition 1.6

Le **complémentaire** d'un graphe $G = (V_G, E_G)$ est le graphe noté $\overline{G}$ défini par :
- $V_{\overline{G}} = V_G$.
- l'arête uv $(u \neq v) \in E_{\overline{G}}$ ssi $uv \notin E_G$.

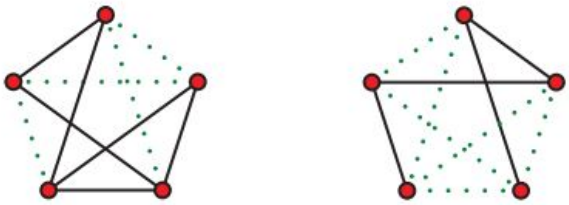


FIGURE 1.1 – Un graphe et son complémentaire.

Graphe adjoint d'un graphe

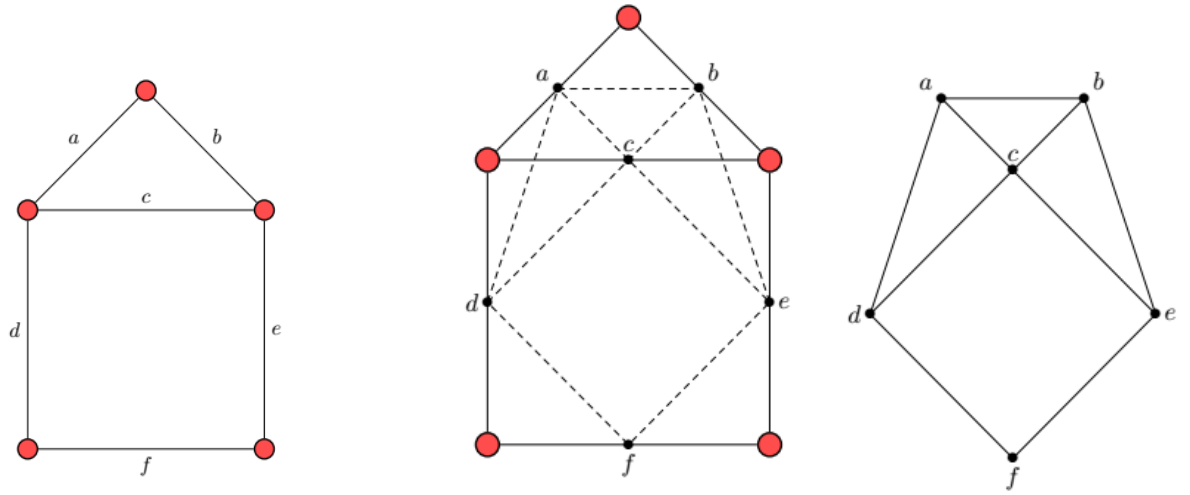
Nous dirons également que deux arêtes distinctes sont adjacentes si elles possèdent une extrémité commune. On peut alors définir le « graphe d'adjacence des arêtes » de G, ou graphe adjoint [1]de G (il est également appelé line graph).

Définition 1.7

Le graphe **adjoint** (line graph) d'un graphe $G = (V_G, E_G)$ est le graphe noté $L(G)$ tel que :
- $V_{L(G)} = E_G$.
- Deux sommets de $L(G)$ sont adjacents si et seulement si sont adjacentes dans G (arêtes adjacentes).

La figure 1.2 illustre cette construction.

1. En français comme en anglais, il existe de nombreux termes (derivative, edge-to-vertex dual, interchange graph, conjugate, derived graph ou même encore theta-obrazom. . .) pour désigner un graphe adjoint. Bien que dans les deux langues line graph ait tendance à s'imposer, nous préférons employer dans la suite l'expression française.



(a) Graphe G.

(b) On relie les sommets correspondant à des arêtes adjacentes.

(c) Graphe $L(G)$.

FIGURE 1.2 – Un graphe et la construction de son graphe adjoint.

Remarque 1.1. *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 de la figure 1.3, appelé griffe. Appelons e le sommet central, e_1, e_2 et e_3 les trois autres sommets, et essayons de reconstruire un graphe G dont la griffe serait le graphe adjoint. Dans G, e est une arête, dont les extrémités peuvent être notées u et v. Comme e est adjacent à e_1, on peut supposer que ces arêtes ont pour extrémité commune u dans G, de même e et e_2 ont une extrémité commune dans G, mais ce ne peut être u puisque e_1 et e_2 n'ont pas d'extrémité commune, c'est donc v. On voit alors qu'il n'est pas possible que l'arête e_3 soit adjacente à l'arête e sans être adjacente ni à l'arête e_1 ni à l'arête e_2. La griffe ne peut donc pas être le graphe adjoint d'un graphe.*

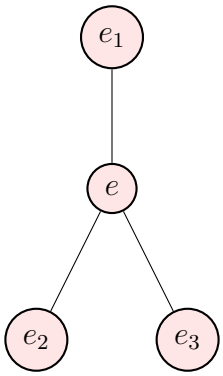


FIGURE 1.3 – La griffe.

1.1.2 Graphes particuliers

Certains graphes, comme la griffe, interviennent si souvent que les chercheurs leur ont attribué un nom ou une notation spéciale. Ce sont notamment les graphes complets, les graphes bipartis, les cycles et les chemins.

Définition 1.8

Le graphe **complet** d'ordre n, noté K_n, est le graphe où chaque sommet est adjacent aux $n-1$ autres sommets.

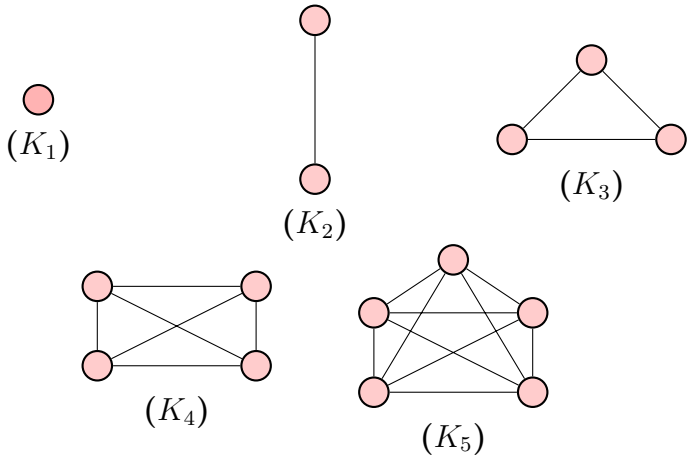


FIGURE 1.4 – Graphes complets d'ordres 1 à 5

Proposition 1.1

Le nombre d'arêtes de K_n est :

$$\frac{n(n-1)}{2}.$$

Démonstration: Dans un graphe complet ayant n sommets, chaque sommet étant relié aux $n-1$ autres sommets, le degré de chaque sommet est $n-1$. Le nombre d'arêtes d'un graphe est égal à la moitié de la somme des degrés de tous ses sommets. Par conséquent, un graphe simple complet ayant n sommets aura $\frac{n(n-1)}{2}$ arêtes. □

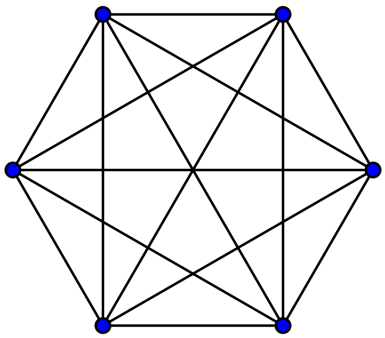


FIGURE 1.5 – Le nombre d'arêtes de K_6 est 15.

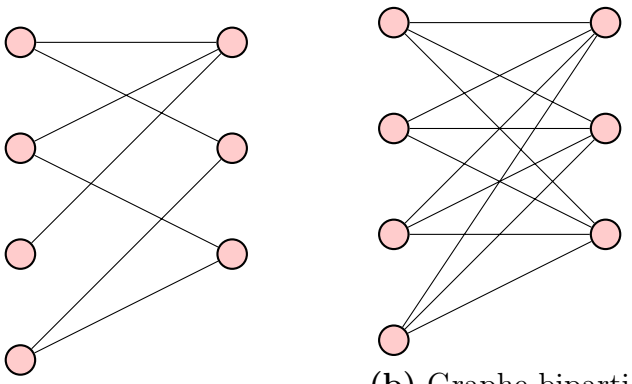
Définition 1.9

Un graphe **biparti** $G = (V_G, E_G)$ est un graphe dont on peut partitionner l'ensemble des sommets V_G en deux ensembles A et B non vides tels que toute arête de G a une extrémité dans A et l'autre dans B. On note alors $G = (A, B; E_G)$.

Définition 1.10

Un graphe **biparti complet** est un graphe biparti $G = (A, B; E)$ où tout sommet de A est adjacent à tout sommet de B. On le note $K_{i,j}$, dans le cas où $i = |A|$ et $j = |B|$.

La figure 1.6 illustre quelques-uns de ces graphes particuliers.



(a) Graphe biparti G.

(b) Graphe biparti complet $K_{3,4}$

FIGURE 1.6 – Un graphe complet et un graphe biparti complet.

Définition 1.11

Un chemin C dans un graphe $G = (V_G, E_G)$ reliant x à y est définie par une suite finie de sommets de type

$$C = [x_0, x_1, x_2, ..., x_{k-1}, x_k]$$

où $x_0 = x, x_k = y$ et $x_i x_{i+1} \in E_G$ pour tout $i \in \{0, 1, ..., k-1\}$.
Le nombre k est la longueur du chemin C. Le chemin C est un **cycle** si $x = y$.
complémentaire d'un trou.
Un chemin simple (resp. **cycle simple**) est un chemin (resp. cycle) ne passant pas deux fois par une même arête, c'est-à-dire dont toutes les arêtes sont distinctes.
Un **chemin élémentaire** (resp. **cycle élémentaire**) d'ordre n, noté P_n (resp. C_n), est un chemin (resp. cycle) ne passant pas deux fois par un même sommet, c'est-à-dire dont tous les sommets sont distinctes.
Un graphe sans cycle élémentaire est appelé **acyclique**.
Une **corde** dans un cycle élémentaire est une arête reliant deux sommets non consécutifs dans ce cycle.
Un **trou** est un cycle élémentaire avec au moins quatre sommets et sans cordes. Un **antitrou** est le complémentaire d'un trou.

Par abus de langage, on parlera plus simplement de **cycle (im)pair** pour désigner un cycle de longueur (im)paire.

Example 1.1. *Considérons le graphe suivant*

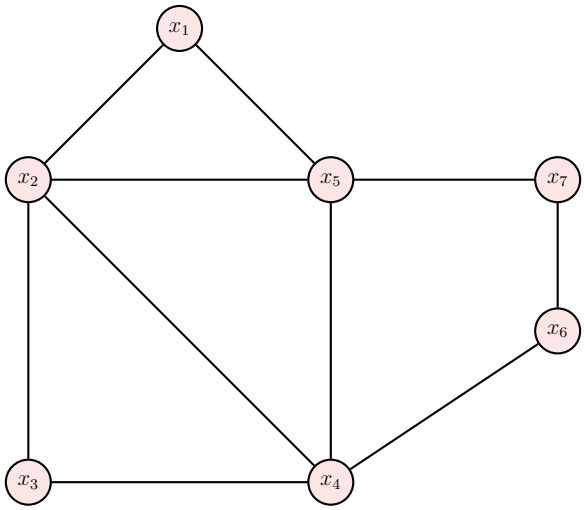


FIGURE 1.7 – Exemple de graphe simple G.

Dans ce graphe on a :

- $[x_1, x_2, x_4, x_5, x_2, x_3]$ *est un chemin simple non élémentaire.*
- $[x_1, x_2, x_3, x_4, x_6]$ *est un chemin élémentaire.*
- $[x_1, x_2, x_3, x_4, x_2, x_5, x_1]$ *est un cycle simple non élémentaire.*
- $C = [x_1, x_2, x_3, x_4, x_5, x_1]$ *est un cycle élémentaire.*
- x_2x_4 *et* x_2x_5 *sont des cordes dans* C*.*
- $[x_5, x_4, x_6, x_7, x_5]$ *est un trou.*

1.1.3 Connexité

La connexité d'un graphe est une mesure importante de sa robustesse quand on le considère comme un réseau (réseau de transport, réseau informatique. . .).

Définition 1.12

Un graphe est dit **connexe** si pour toute paire de sommets x et y de G, il existe un chemin entre x et y, on dit alors que les sommets x et y sont **connectés**.

La connexité définit une relation d'équivalence sur l'ensemble des sommets, et chacune des classes d'équivalence est appelée **composante connexe** (connected component) du graphe. Autrement dit, un graphe est connexe si et seulement s'il ne contient qu'une seule composante connexe. Intuitivement, le nombre de composantes connexes correspond au nombre de « morceaux » du graphe quand on le dessine. Un résultat classique énonce que :

Lemme 1.2

Pour tout graphe G, G est connexe ou $\overline{G}$ est connexe.

Démonstration: Supposons G non connexe et montrons que son complémentaire l'est en montrant que pour tout $v_1, v_2 \in V_G = V_{\overline{G}}$, il existe un chemin dans $\overline{G}$ allant de v_1 à v_2.

- Si v_1 et v_2 ne sont pas dans la même composante connexe, alors $v_1v_2 \notin E_G$, donc $v_1v_2 \in E_{\overline{G}}$, ainsi v_1 et v_2 sont connectés dans $\overline{G}$.
- Si v_1 et v_2 appartiennent a la même composante connexe, comme G n'est pas connexe il existe un sommet v_3 de G qui n'est pas dans la même composante connexe que v_1 et v_2. On en déduit alors en appliquant deux fois le raisonnement précédent que v_1v_3 et v_3v_2 sont des arêtes de $\overline{G}$ qui forment un chemin de v_1 à v_2 ($[v_1, v_3, v_2]$).

□

Définition 1.13

Soit G un graphe connexe.
Un **point** (resp. ensemble) **d'articulation** dans G est un sommet (resp. ensemble de sommets) de G tel que sa suppression rend le graphe G non connexe.

Définition 1.14

On dit qu'un graphe simple $G = (V_G, E_G)$ d'ordre n est k-connexe ($k \geq 2$) si $n > k$ et pour tout ensemble de sommets $S \subset V_G$ avec $|S| \leq k-1$, $G - S$ est connexe.

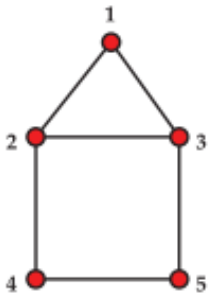
1.1.4 Représentations des graphes en machine

Matrice d'adjacence

Définition 1.15

Soit $V_G = \{1, 2, ..., n\}$.
La matrice d'adjacence d'un graphe $G = (V_G, E_G)$ est la matrice carrée $A = (a_{ij})_{1 \leq i,j \leq n}$ de taille n telle que $a_{ij} = 1$ s'il existe une arête entre les sommets i et j, et $a_{ij} = 0$ sinon.

Notons que la matrice d'adjacence d'un graphe dépend de la numérotation des sommets du graphe qu'elle représente.



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

FIGURE 1.8 – Un graphe (la maison) et sa matrice d'adjacence.

Matrice d'incidence

Définition 1.16

Soit $V_G = \{1, 2, ..., n\}$.
La matrice d'incidence d'un graphe $G = (V_G, E_G)$ de taille m est la matrice $B = (b_{ij})_{\substack{1 \leq i \leq n \\ 1 \leq j \leq m}}$ de taille $n \times m$, telle que $b_{ij} = 1$ si le sommet i est une extrémité de l'arête j, 0 sinon.

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{cccccc} \multicolumn{6}{c}{\text{Les arêtes}} \\ 1-2 & 1-3 & 2-3 & 2-4 & 3-5 & 4-5 \end{array} \\
\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}
\end{array}
$$

FIGURE 1.9 – Représentation par listes d'adjacence du graphe de la figure 1.8.

1.1.5 Sous-graphes

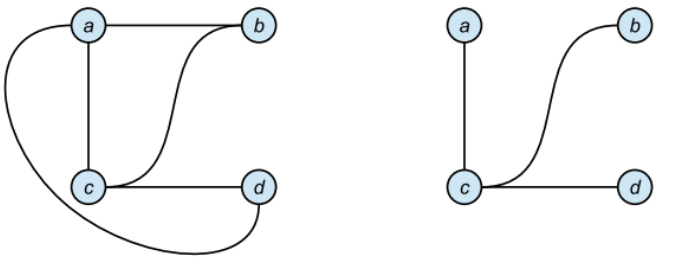
En théorie des graphes, un sous-graphe est un graphe contenu dans un autre graphe. Formellement,

Définition 1.17

Un graphe $H = (V_H, E_H)$ est un **sous-graphe** de $G = (V_G, E_G)$ si $V_H \subseteq V_G$ et $E_H \subseteq \{a \in E_G | \textit{ les extrémités de l'arête } a \textit{ sont dans } V_H\}$; G est alors un **sur-graphe** de H.

Définition 1.18

Un **sous-graphe partiel** (ou parfois graphe couvrant) de $G = (V, E)$ est un sous-graphe $H = (V, E_H)$ (c'est-à-dire qu'on peut l'obtenir à partir de G en supprimant uniquement des arêtes).

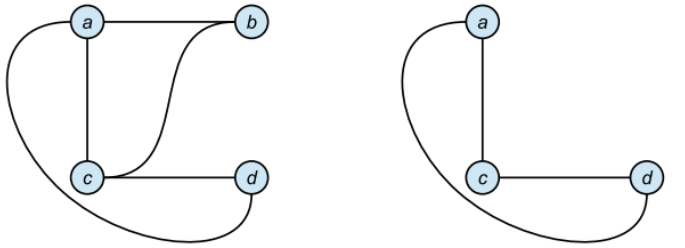


(a) Graphe G (b) Graphe partiel G''

FIGURE 1.10 – Un graphe et un graphe partiel.

Définition 1.19

Le sous-graphe de $G = (V_G, E_G)$ **induit** par $W \subseteq V_G$, noté $G[W]$, est le sous-graphe ayant pour ensemble de sommets W et pour ensemble d'arêtes toutes les arêtes de G ayant leurs extrémités dans W.



(a) Graphe G (b) Sous-graphe induit G'''

FIGURE 1.11 – Un graphe et un sous-graphe induit.

Example 1.2. *On représente la carte des routes nationales et autoroutes de Maroc par un graphe : les villes sont les sommets, et les routes sont les arêtes. Si l'on ne considère que les autoroutes, on est en présence d'un sous-graphe de la carte. Si l'on se restreint aux villes de la région Fès-Meknès, on a le sous-graphe induit par ces villes.*

1.1.6 Stables et cliques

Définition 1.20

Soit G un graphe simple d'ordre au moins deux.
- Un **stable** (ou independant) dans G est un ensemble de sommets deux à deux non adjacents.
- Une **clique** dans G est un ensemble de sommets deux à deux adjacents.

Ainsi, une clique dans un graphe est un stable dans son complémentaire et inversement. De plus, tout graphe induit par une clique étant un graphe complet, les deux termes sont le plus souvent confondus, de même, un graphe sans arête est régulièrement appelé stable [2]

Définition 1.21

- Un ensemble (de sommets, d'arêtes. . .) est **minimal** (resp. **maximal**) pour une propriété P s'il ne contient (resp. n'est contenu dans) aucun ensemble vérifiant la propriété P.
- Un ensemble (de sommets, d'arêtes. . .) est **minimum** (resp. **maximum**) pour une propriété P s'il est de cardinalité minimale (resp. maximale) pour la propriété P.

Il est facile de voir qu'un ensemble maximum est aussi maximal ; mais un ensemble maximal n'est pas nécessairement maximum. Par exemple, le stable formé par les sommets rouges sur la figure 1.12(a) est maximal (il est en effet impossible d'ajouter à ce stable un autre sommet pour former un stable de cardinalité supérieure), mais il n'est pas maximum (puisqu'il est possible de trouver un autre stable, de cardinalité supérieure (fig. 1.12(b))).

(a) Un stable maximal (b) Un stable maximum

FIGURE 1.12 – La différence entre un stable maximal et un stable maximum.

Remarque 1.2. *Dans un graphe simple il est possible de trouver plusieurs stables maximals.*

La taille d'un stable maximum de G est notée $\alpha(G)$, et celle d'une clique maximum est notée $\omega(G)$. Naturellement, on a

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

Déterminer un stable maximum ou une clique maximum dans un graphe arbitraire n'est pas du tout trivial.

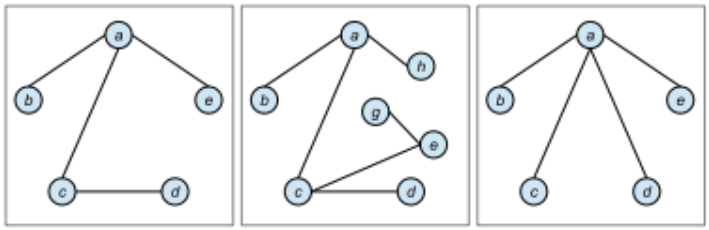
2. On trouve également dans la littérature les termes graphe nul ou graphe vide pour désigner un graphe sans arête, mais ces expressions désignent chez d'autres auteurs le graphe sans aucun sommet (et donc sans arête).

1.1.7 Forêts et arbres

Définition 1.22

- Une **forêt** est un graphe acyclique, i.e. un graphe sans cycle élémentaire.
- Un **arbre** est une forêt connexe, i.e. un graphe acyclique connexe.
  Les sommets de degré 1 dans un arbre sont appelés ses **feuilles**

La Figure1.13 montre des exemples d'arbres.



(a) Un premier arbre (b) Une deuxième arbre (c) Un troisième arbre

FIGURE 1.13 – Quelques exemples des arbres.

Définition 1.23

Soit $G = (V_G, E_G)$ un graphe simple.
Un **arbre de recouvrement**(arbre couvrant) de G est un sous-graphe partiel de G qui est un arbre.

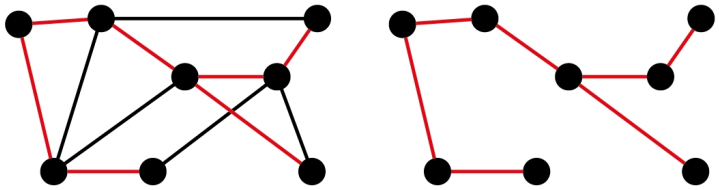


FIGURE 1.14 – Exemple d'arbre couvrant

Proposition 1.2

Un graphe $G = (V_G, E_G)$ admet un arbre arbre de recouvrement si et seulement s'il est connexe.

1.1.8 Isomorphisme de graphes

Définition 1.24

Deux graphes $G = (V_G, E_G)$ et $H = (V_H, E_H)$ sont isomorphes s'il existe une bijection $f : V_G \to V_H$ telle que $\forall u, v \in V_G$, u et v sont adjacents dans G si et seulement si

$f(u)$ et $f(v)$ sont adjacents dans H. On note alors $G \backsimeq H$.

Intuitivement, deux graphes sont isomorphes s'ils ont la même « structure », i.e. s'il est possible de déplacer les sommets de l'un pour qu'il soit la copie conforme de l'autre (au nom des sommets près). La figure 1.15 illustre cette définition.

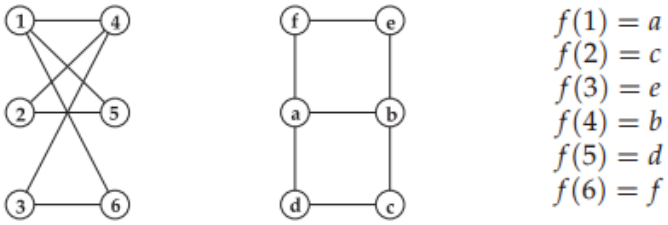


FIGURE 1.15 – Deux graphes isomorphes et la bijection explicitée.

Remarque 1.3. *1. Lorsqu'un graphe est isomorphe à son complémentaire, on dit qu'il est autocomplémentaire. C'est le cas par exemple de P_4 ou de C_5.*

2. La relation "être isomorphe à" est une relation d'équivalence.

1.2 Notions sur la complexité

La théorie de la complexité est une branche des mathématiques et de l'informatique ayant pour cadre l'étude de la difficulté intrinsèque des problèmes algorithmiques, et qui vise à classer ces problèmes en fonction de cette difficulté. Ici, les mots « complexité » et « difficulté » ne se rapportent pas à la mise au point d'un algorithme de résolution, ou aux concepts avancés auxquels il peut faire appel (comme une structure de données élaborée), mais plutôt à la quantité de ressources à utiliser pour résoudre le problème. En ce qui nous concerne, les ressources consistent en le temps que met un algorithme à résoudre le problème (c'est sa complexité temporelle) et l'espace mémoire qu'il utilise au cours de son exécution (sa complexité spatiale) [3], mais il existe d'autres mesures de complexité, comme le nombre de portes logiques à utiliser pour la réalisation d'un circuit, ou encore la quantité d'information à transmettre dans le cadre de la théorie de la complexité de la communication. Le principal atout de la théorie de la complexité est que ces grandeurs sont exprimées indépendamment de tout dispositif physique concret, au contraire, elle est basée sur un modèle de calcul abstrait, généralement une machine de Turing, ce qui permet de comparer facilement l'efficacité de deux algorithmes en s'affranchissant de considérations telles que la vitesse du processeur. On est quasiment sûrs aujourd'hui que certains problèmes nécessitent, pour leur résolution, des algorithmes dont le temps de calcul est bien supérieur à celui d'autres problèmes, et c'est en ce sens que l'on dira qu'ils sont « plus difficiles ».

Définition 1.25 (Premières définitions)

Un problème est une question générale possédant des paramètres dont la valeur n'est pas connue.
Une **instance** d'un problème est obtenue en affectant une valeur à chacun de ses

3. Il faut distinguer la théorie de la complexité de l'analyse des algorithmes : celle-ci a pour objet l'analyse de la quantité de ressources utilisées par un algorithme particulier

paramètres.
La **taille** d'une instance désigne généralement la quantité de cases mémoires nécessaires pour décrire les paramètres.

Example 1.3. *Le problème du voyageur de commerce (ou TSP pour Traveling salesman problem) consiste, étant donné un ensemble de villes séparées par des distances connues, à trouver le plus court chemin qui relie toutes les villes, en ne passant qu'une seule fois par chaque ville. Une instance du TSP est donc un ensemble de n points (représentant les villes) définis chacun par un couple de coordonnées et la taille de cette instance est* $2n+1$ *(il faut une case mémoire pour chaque coordonnée des n points et une autre pour stocker l'entier n).*

Nous donnons maintenant des définitions plus formelles des notions abordées dans le paragraphe précédent. On commence par définir la notion de complexité et la notation « grand O », puis les classes de complexité $\mathcal{P}$ et $\mathcal{NP}$qui en découlent. Enfin nous présentons rapidement le concept d'algorithme d'approximation.

Problèmes de décision et d'optimisation

Définition 1.26

Un **problème de décision** est un problème auquel la réponse est oui ou non.

Définition 1.27

Un **problème d'optimisation** consiste à déterminer la meilleure solution parmi toutes les solutions réalisables.

Complexité en temps et en espace

Définition 1.28

La complexité **temporelle** d'un algorithme correspond au nombre d'instructions élémentaires (opérations arithmétiques, comparaison, affectation. . .) effectuées au cours de son exécution.

Définition 1.29

La complexité **spatiale** d'un algorithme (dans le cas où l'algorithme se termine) correspond au nombre de cases mémoires occupées par les données manipulées par l'algorithme au cours de son exécution.

Notation « grand O »

La notation grand O, dite aussi symbole de Landau, décrit le comportement asymptotique d'une fonction, exprimé à l'aide d'une autre fonction généralement plus simple. Plus formellement, nous dirons que :

Définition 1.30

$f(n) = O(g(n))$ ($f(n)$ est en grand O de $g(n)$) quand $n \to +\infty$ si et seulement si $\exists M > 0, n_0 \in \mathbb{N}$ tels que $\forall n \geq n_0, |f(n)| \leq M|g(n)|$

Intuitivement, ceci signifie qu'à partir de n_0 et à un facteur constant près, f ne croît pas plus rapidement que g.

Example 1.4. *Soit $f(n) = 6n^4 - 2n^3 + 5$. Choisissons $n_0 = 1$. Alors pour tout $n \geq n_0$, on a*

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

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

Il est facile de voir qu'un polynôme $P(n)$ de degré k est toujours en $O(n^k)$.
En fonction de l'ordre de grandeur, nous avons différents types de complexité :
– $O(1)$: complexité constante, indépendante de la taille des données,
– $O(log(n))$: complexité logarithmique,
– $O(n)$: complexité linéaire,
– $O(nlog(n))$: complexité quasi-linéaire,
– $O(n^2)$: complexité quadratique,
– $O(n^3)$: complexité cubique,
– $O(n^k)$: complexité polynomiale,
– $O(2^n)$: complexité exponentielle,
– $O(n!)$: complexité factorielle.

Classes de complexité $\mathcal{P}$ et $\mathcal{NP}$

Définition 1.31

Un problème de décision est dans la **classe** P si, pour chacune de ses instances, dont la taille est notée n, il existe un réel positif k tel qu'il peut être résolu par un algorithme de complexité temporelle $O(n^k)$, c'est-à-dire qu'il peut être décidé en temps polynomial.

Les problèmes de la classe P sont dits faciles. Ce sont ceux que l'on sait résoudre efficacement.

Définition 1.32

Un problème de décision est dans la **classe** $\mathcal{NP}$ [4]si l'on peut vérifier en temps polynomial qu'une solution pour une instance donnée est valide (ce que l'on appelle un certificat du oui).

Intuitivement, les problèmes de la classe $\mathcal{NP}$ sont ceux que l'on peut résoudre en énumérant l'ensemble des solutions possibles (méthode « brutale ») et en les testant à l'aide d'un algorithme polynomial. Naturellement, si on peut résoudre un problème avec un algorithme polynomial, on peut aussi vérifier en temps polynomial que la solution fournie est bien une solution, par conséquent $\mathcal{P} \subseteq \mathcal{NP}$.

Problèmes $\mathcal{NP}$-complets

Une réduction est une transformation d'un problème en un autre, ceci permet de capturer la notion informelle de « problème au moins aussi difficile qu'un autre problème ». Plus précisément, si un problème $\mathcal{X}$ peut être résolu en utilisant un algorithme permettant de résoudre un problème $\mathcal{Y}$, c'est que $\mathcal{X}$ n'est pas plus difficile que $\mathcal{Y}$; on dit alors que se réduit à $\mathcal{Y}$. Par exemple, le problème consistant à élever un nombre au carré se réduit au problème plus général de multiplication de deux nombres (ici, aucune transformation n'est nécessaire). Une réduction est polynomiale lorsque le processus de transformation peut se faire en temps polynomial.

Définition 1.33

Un problème $\mathcal{X}$ est difficile pour une classe de problèmes $\mathcal{C}$, ou $\mathcal{C}$-difficile, si tout problème de $\mathcal{C}$ se réduit à $\mathcal{X}$.

Autrement dit, il n'existe pas de problème de $\mathcal{C}$ plus difficile que $\mathcal{X}$, puisque tout algorithme résolvant $\mathcal{X}$ résout aussi n'importe quel problème de $\mathcal{C}$. En particulier, les problèmes difficiles pour la classe $\mathcal{NP}$ forment la classe de problèmes $\mathcal{NP}$-difficiles. Lorsque, de plus, $\mathcal{X}$ appartient lui aussi à la classe $\mathcal{C}$, on dit qu'il est complet pour $\mathcal{C}$; ceci signifie que $\mathcal{X}$ est l'un des problèmes les plus difficiles de $\mathcal{C}$ (il peut en effet y avoir plusieurs problèmes de même difficulté).

Définition 1.34 (Problème $\mathcal{NP}$-complet)

Un problème de décision est $\mathcal{NP}$-complet s'il vérifie les deux conditions suivantes :
– il appartient à la classe $\mathcal{NP}$,
– tous les problèmes de la classe $\mathcal{NP}$ se ramènent à celui-ci via une réduction polynomiale.

Tout problème d'optimisation combinatoire dont le problème de décision associé est $\mathcal{NP}$-complet est $\mathcal{NP}$-difficile. Le problème du stable maximum est $\mathcal{NP}$-difficile

4. Pour Non-déterministe Polynomial et non « Non-Polynomial ».

CHAPITRE 2

Coloration de graphes

2.1 Introduction

La coloration de graphes est depuis plusieurs décennies, un domaine très actif de la théorie des graphes du fait de ses nombreuses applications. En effet, un grand nombre de problèmes peuvent se traduire comme une partition d'un ensemble d'objets suivant certaines contraintes. Supposons par exemple que nous avons k conteneurs de produits chimiques à répartir dans des entrepôts. Certains produits chimiques ne peuvent être stockés dans le même bâtiment en raison du risque d'explosion en cas de mélange. Nous voulons ainsi connaître le nombre minimum d'entrepôts dont nous avons besoin pour stocker ces conteneurs de façon à minimiser le coût de stockage. Les graphes constituent alors un outil qui permet de modéliser ce genre de problèmes en se ramenant à l'étude de colorations. Nous construisons tout d'abord un graphe non orienté où chaque sommet représente un conteneur de produits chimiques, et nous relions deux sommets si les deux conteneurs correspondants contiennent des produits chimiques qui ne peuvent être stockés dans un même entrepôt. Nous colorions alors les sommets de ce graphe de telle sorte que deux sommets reliés par une arête ont des couleurs différentes. Ainsi, chaque couleur correspond à un entrepôt. Il s'agit dans ce cas d'**une coloration propre** de sommet du graphe. Par conséquent, les conteneurs dont les sommets sont de la même couleur seront stockés dans le même entrepôt. Enfin, en déterminant le nombre minimum de couleurs dont nous avons besoin pour colorier ce graphe, nous connaîtrons le nombre minimum d'entrepôts dont nous avons besoin pour stocker les k conteneurs.
De nombreuses variantes et extensions de colorations de graphes ont été considérées dans la littérature. **Les colorations impropres** (également connues sous le nom de colorations **généralisées** ou **relaxées**) sont une généralisation des colorations propres où la contrainte consistant à colorier de façon différente deux sommets adjacents est supprimée. Dans la suite de ce chapitre tous les graphes sont simples et non-orientés.

2.2 Coloration propre

Définition 2.1

Une coloration propre d'un graphe $G = (V_G, E_G)$ est une application f des sommets de G vers un ensemble de couleurs C telle que deux sommets adjacents reçoivent des couleurs différentes. Si $|C| = k$, alors f est une k-coloration propre de G.
Le nombre chromatique de G, noté $\chi(G)$, est défini comme étant le plus petit entier k tel que G admet une k-coloration propre.

Une coloration propre de G utilisant k couleurs est une partition de V_G en ensembles $S_1, S_2, ..., S_k$ tels que pour tout $1 \leq i \leq k$, S_i est un stable.

Example 2.1. *Considérons les figures suivants*

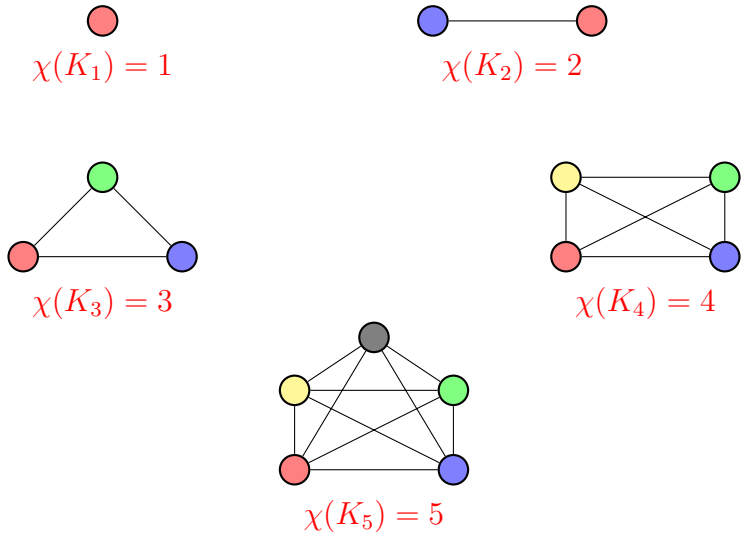


FIGURE 2.1 – Le nombre chromatique des graphes complets d'ordres 1 à 5

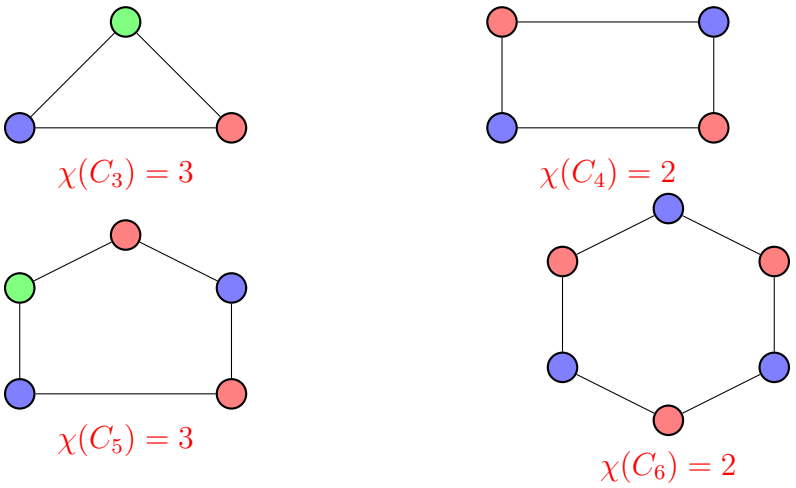
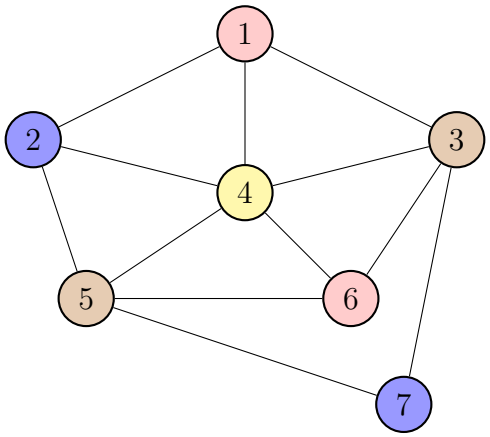


FIGURE 2.2 – Le nombre chromatique des cycles élémentaires d'ordres 3 à 6

On remarque que $\chi(K_n) = n$ et que $\chi(C_n) = 2$ si n est pair et $\chi(C_n) = 3$ si n est impair (pour la preuve voir la proposition 2.4).

Example 2.2. *La figure 2.3 illustre un exemple de graphe 4-coloriable.*



$S_1 = \{2, 7\}, S_2 = \{1, 6\}, S_3 = \{4\}, S_4 = \{3, 5\}.$

FIGURE 2.3 – Graphe admettant une 4-coloration

Ce graphe est 4-coloriable donc $\chi(G) \leq 4$*, il contient un* C_5 *impair donc* $\chi(G) \geq 3$*, de plus le sommet* 4 *est adjacent à tous les sommets de* C_5*, il faut donc colorier ce sommet avec une couleur différente des couleurs déjà affectées aux sommets de* C_5*, on en conclut que* $\chi(G) \geq 4$*, donc le nombre chromatique de* G *est* $\chi(G) = 4$*.*

2.3 Colorations impropres

Définition 2.2

Soient $G = (V_G, E_G)$ un graphe, k et l deux entiers.
Une coloration k-impropre d'un graphe $G = (V_G, E_G)$ est une application c des sommets de G vers un ensemble de couleurs C telle que tout sommet $v \in V_G$, possède au plus k voisins $v_1, v_2..., v_k$ dans G tels que $c(v) = c(v_i)$ pour tout $i \in \{1, 2, ..., k\}$.
Si $|C| = l$, alors c est une **l-coloration k-impropre** de G.

On peut définir aussi une **l-coloration k-impropre** de G comme une partition de l'ensemble des sommets de G en l classes de couleurs $C_1, C_2, ..., C_l$ telles que chaque classe C_i induit un graphe de degré maximum au plus k, c'est-à-dire telle que chaque sommet de G possède au plus k voisins de la même couleur que lui dans G.

Remarque 2.1. *Il ne faut pas confondre les classes des couleurs $C_1, C_2, ..., C_l$ d'une coloration impropre et les ensembles stables $S_1, S_2, ..., S_l$ d'une coloration propre. Lorsqu'une coloration impropre est propre alors chaque C_i est un stable dans G.*

Définition 2.3

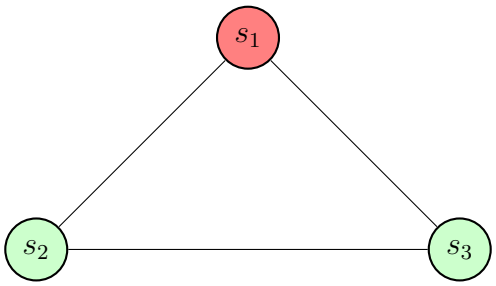
Si c est une coloration impropre de G, **l'impropreté** d'un sommet v sous c, notée $im_G^c(v)$, est le nombre de voisins de v colorés $c(v)$, c'est-à-dire colorés de la même couleur que v. L'impropreté de c, notée $im_G(c)$, est **l'impropreté maximale** d'un sommet de G sous c :

$$im_G(c) := max\{im_G^c(v) : v \in V(G)\}$$

Une coloration d'impropreté au plus k est dite k-impropre. Un graphe est k-improprement l-colorable, si et seulement si, il admet une l-coloration k-impropre.

Définition 2.4

Le nombre minimum de couleurs d'une coloration k-impropre de G noté $\chi_k(G)$, est le **nombre chromatique k-impropre** de G.



$c(s_1) = rouge$, $c(s_2) = c(s_3) = vert$.
$im_G(s_1) = 0, im_G(s_2) = im_G(s_3) = 1$
$im_G(c) = 1$

FIGURE 2.4 – Un exemple de 2-coloration 1-impropre, $\chi_1(G) = 2$.

.

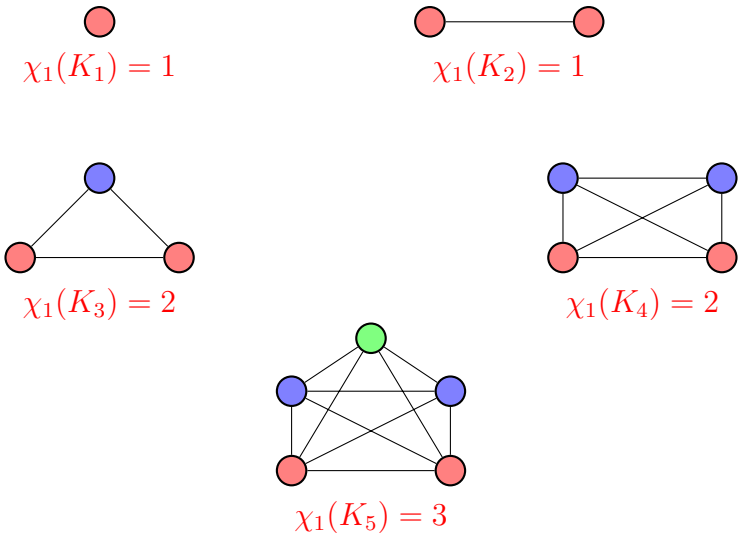


FIGURE 2.5 – Le nombre chromatique 1-impropre des graphes complets d'ordres 1 à 5

Une coloration 0-impropre est une coloration propre (coloration san conflits) , c'est-à-dire telle que deux sommets adjacents soient colorés différemment, on a donc $\chi_0(G) = \chi(G)$, où $\chi(G)$ est le nombre chromatique propre (usuel) de G.
La coloration k-impropre est une généralisation de la coloration propre et a été introduite en premier par Lovász [12] en 1966 qui a démontré les premiers résultats sur le nombre chromatique k-impropre $\chi_k(G)$. Ce n'est que vingt ans plus tard qu'Andrews et Jacobson [2], Harary et Fraughnaugh [20], Cowen et Woodall [26] ont remis cette théorie au goût du jour, de manière indépendante. Andrews et Jacobsen ont dérivé des bornes inférieures sur le nombre chromatique k-impropre d'un graphe, tandis que Haray et Fraughnaugh ont étudié ce paramètre comme un cas particulier d'une généralisation plus large du nombre chromatique. Enfin Cowen et Woodall ont trouvé des bornes supérieures sur ce paramètre en vue de généraliser le fameux théorème des quatre couleurs [3] .
Puisque une coloration k-impropre est également $(k+1)$-impropre, il est clair que $\chi_k(G) \geq \chi_{k+1}(G)$ donc $\chi(G) = \chi_0(G) \geq \chi_k(G)$. De plus, puisqu'une classe de couleur d'une coloration k-impropre induit un sous-graphe dont les sommets ont au plus k voisins, il est toujours possible de séparer cette classe en $k+1$ stables c'est-à-dire,

$$\chi(G) \leq (k+1)\chi_k(G)$$

d'où les inégalités

$$\left\lceil \frac{\chi(G)}{k+1} \right\rceil \leq \chi_k(G) \leq \chi(G)$$

.
Dans la suite de ce chapitre on va s'intéresser à la coloration propre des graphes simples (ou simplement la coloration des graphes simples).

2.4 Encadrement de nombre chromatique

Proposition 2.1

Soit G un graphe simple d'ordre n.

1. $\chi(G) = 1$ ssi G n'a pas d'arêtes.
2. $\chi(G) \leq n$.
3. Si G a au moins une arête alors, G est biparti ssi $\chi(G) = 2$.
4. $\chi(K_n) = n$ pur tout $n \geq 1$.
5. $\chi(C_n) = 2$ si n est pair et $\chi(C_n) = 3$ si n est impair.
6. Pour tout sous graphe H de G on a $\chi(H) \leq \chi(G)$.
7. $\chi(G) \geq \omega(G)$

Démonstration:

1. Facile, car si deux sommets sont adjacents, on aura $\chi(G) \geq 2$.
2. Facile aussi, il suffit de colorier chaque sommet d'une couleur différente.
3. Un graphe biparti est un graphe dont on peut partitionner l'ensemble des sommets en deux stables, d'où le résultat.
4. N'importe quel coloriage de K_n avec moins de n couleurs implique que deux sommets sont de la même couleur, ce qui est une contradiction car tous les sommets sont adjacents entre eux.
5. Par 1), $\chi(Cn) \geq 2$. Supposons que l'ensemble des sommets de C_n sont $s_1, s_2, ..., s_n$ avec $s_i s_{i+1} \in E_G$ pour tout $1 \leq i \leq n-1$ et $s_1 s_n \in E_G$.
   Si n est pair, alors on colorie tous les sommets d'indice pairs avec une couleur et tous les autres avec l'autre, donc $\chi(C_n) \leq 2$.
   Si n est impair, on montre que C_n n'est pas 2-coloriable. Supposons qu'il le soit et qu'on colorie ses sommets en bleu ou rouge. On suppose que s_n est bleu. Alors s_1 et s_{n-1} doivent être rouges, s_2 et s_{n-2} doivent être bleus et de façon générale s_k et s_{n-k} sont de la même couleur pour tout $1 \leq k \leq \frac{n}{2}$. Si $n = 2m + 1$, alors avec $k = m$, on voit que s_m et s_{m+1} sont de la même couleur, c'est une contradiction, donc $\chi(C_n) \geq 3$.
   Il est aussi facile de voir que C_n est 3-coloriable, dans la preuve ci-dessus, il suffit de colorier le sommet d'indice m avec la troisième couleur (donc $\chi(C_n) = 3$).
6. Facile, car tous les sommets adjacents dans H sont adjacents dans G.
7. $\omega(G)$ est le cardinal d'une clique maximum C de G, donc $G[C]$ est complet et $\omega(G) = \chi(G[C]) \leq \chi(G)$.

□

Définition 2.5

Un graphe G est **k-dégénéré** si tout sous-graphe de G contient un sommet de degré au plus k.

Example 2.3. – *Les arbre sont des graphes 1-dégénéré.*

– *Le graphe de la figure 2.7 est 2-dégénéré.*

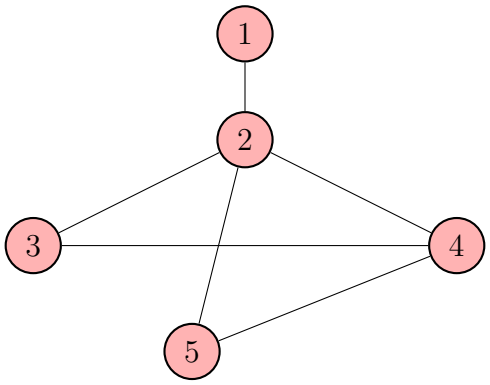


FIGURE 2.6 – Un graphe 2-dégénéré.

Remarque 2.2. *Tout graphe qui contient un sommet de degré k n'est pas nécessairement k-dégénéré.*
En effet, considérons le graphe suivant :

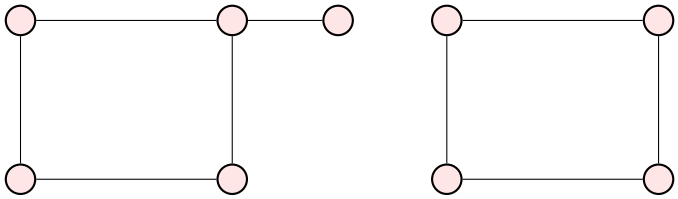


FIGURE 2.7 – Graphe n'est pas 1-dégénéré.

Ce graphe (a) contient un sommet de degré 1, mais sont sous graphe induit (b) ne contient pas un sommet de degré 1, donc n'est pas k-dégénéré.

Proposition 2.2

Soit $G = (V_G, E_G)$ un graphe simple d'ordre n de degré maximum $\Delta(G)$, alors

$$\chi(G) \leq \Delta(G) + 1$$

Démonstration: Raisonnons par récurrence sur le nombre n de sommets du graphe.
Si $n = 1$ c'est trivial.
Supposons l'assertion vraie pour tout graphe ayant au plus $n - 1$ sommets, $n \geq 1$, et soit G un graphe avec n sommets. Si on supprime un sommet x le graphe obtenu G' a un degré maximum qui est au plus égal à $\Delta(G)$ (i.e. $\Delta(G') \leq \Delta(G)$). Par hypothèse de récurrence, on a

$$\chi(G') \leq (\Delta(G) + 1).$$

Une $(\Delta(G) + 1)$-coloration de G est obtenue en coloriant le sommet x avec une couleur différente des couleurs déjà affectées aux sommets adjacents à x (il y en a au plus $\Delta(G)$). □

Cependant, Brooks a amélioré cette borne (théorème 2.1). Il est claire que le résultat de la proposition (2.2) équivalent à $\chi(G) \leq \Delta(G)$ ou $\chi(G) = \Delta(G) + 1$ puisque $\Delta(G)$ est un entier, d'où le théorème 2.3 :

Proposition 2.3 ([1])

Soit $G = (V_G, E_G)$ un graphe simple connexe d'ordre n et de degré maximum $\Delta(G)$. Si G est un cycle élémentaire impair ou un graphe complet, alors

$$\chi(G) = \Delta(G) + 1$$

(Dans le cas contraire nous avons $\chi(G) \leq \Delta(G)$, c'est le théorème 2.1).

Démonstration: **Si** G est un cycle élémentaire impair, alors d'après 5 de la proposition 2.1 on a $\chi(G) = 3$. Dans un cycle élémentaire, tous les sommets sont de degré 2, donc $\Delta(G) = 2$. On en conclut que $\Delta(G) + 1 = 2 + 1 = 3 = \chi(G)$.
Si G est un graphe complet alors tous les sommets sont de degrés $n-1$, donc $\Delta(G) = n-1$, et d'après 4 de la proposition 2.1 $\chi(G) = n$, on en conclut que $\chi(G) = n = \Delta(G) + 1$. □

Proposition 2.4 ([1])

Soit $G = (V_G, E_G)$ un graphe simple d'ordre n, alors

$$\left\lceil \frac{n}{\alpha(G)} \right\rceil \leq \chi(G)$$

et

$$\chi(G) + \alpha(G) \leq n + 1$$

Démonstration: Nous savons qu'une coloration des sommets est une partition $\{S_1, S_2, ..., S_{\chi(G)}\}$ de V_G . De plus pour tout $1 \leq i \leq \chi(G)$, S_i est un stable et nous avons $|S_i| \leq \alpha(G)$. Donc $n = |V_G| = |S_1 \cup S_2 \cup ... \cup S_{\chi(G)}| = \sum_{1 \leq i \leq \chi(G)} |S_i| \leq \chi(G)\alpha(G)$. D'où la première inégalité.

Si on colorie un ensemble stable maximum S_m avec une couleur, on utilisera au plus $|V_G| - \alpha(G)$ couleurs pour colorier $V_G \setminus S_m$. Ainsi on a $\chi(G) \leq (|V_G| - \alpha(G)) + 1 = n - \alpha(G) + 1$. D'où la deuxième inégalité.

□

En combinant la proposition 2.2 et la proposition 2.4 nous obtenons un encadrement du nombre chromatique d'un graphe simple.

Corollaire 2.1 ([1])

Soit $G = (V_G, E_G)$ un graphe simple d'ordre n, alors

$$\left\lceil \frac{n}{\alpha(G)} \right\rceil \leq \chi(G) \leq \Delta(G) + 1$$

Proposition 2.5 ([1])

Soit $G = (V_G, E_G)$ un graphe simple d'ordre n. Alors

$$\chi(G) \leq 1 + \max\{\delta(H), H \text{ sous-graphe induit de } G\} \leq 1 + \Delta(G).$$

Démonstration: Rappelons que $\delta(G)$ est le degré minimum des sommets de G. Posons $k = \max\{\delta(H), H \text{ sous-graphe induit de } G\}$.
Le graphe G contient un sommet x_n tel que $d_G(x_n) = \delta(G) \leq k$, car G est également un sou-graphe induit de G. Si on supprime ce sommet on obtient le graphe $G_{n-1} = G[V \setminus \{x_n\}]$. Soit x_{n-1} un sommet de G_{n-1} tel que $d_{G_{n-1}}(x_{n-1}) = \delta(G_{n-1}) \leq k$. On continue le processus jusqu'à l'obtention du graphe $G_1 = G[V \setminus \{x_n, x_{n-1}, ..., x_2\}]$ qui est constitué du seul sommet x_1. Ainsi nous avons ordonné les sommets $x_1, x_2, ..., x_n$ de telle sorte que chaque x_i, $1 \leq i \leq n$, a au plus k voisins dans G_i, car $\delta(G_i) \leq k$ est le degré de x_i dans G_i. Ainsi on a besoin d'au plus $1 + k$ couleurs pour colorier le graphe G (i.e. $\chi(G) \leq k + 1$).
Pour la deuxième inégalité il suffit de remarquer que pour tout sous-graphe induit H de G on a $\delta(H) \leq \Delta(H) \leq \Delta(G)$. □

Corollaire 2.2

Soit $G = (V_G, E_G)$ un graphe simple connexe qui n'est pas régulier. On a

$$\chi(G) \leq \Delta(G).$$

Démonstration: Supposons que $\chi(G) > \Delta(G)$, alors $\chi(G) \geq \Delta(G) + 1$ en utilisant la proposition 2.5 on a : $\chi(G) = \Delta(G) + 1 = \max\{\delta(H), H \text{ sous-graphe induit de } G\}$, il existe donc un sous-graphe induit H de G tel que $\delta(H) = \Delta(G)$, or $\delta(H) \leq \Delta(H) \leq \Delta(G)$, alors $\delta(H) = \Delta(H)$ Par conséquent H est un sous-graphe induit $\Delta(G)$-régulier et celui-ci ne peut pas être connecté à un sommet extérieur à H, car sinon on aura un sommet de G de degré au moins $\Delta(G) + 1$,G étant connexe, on a $G = H$, mais H est régulier : contradiction. □

Une question intéressante est de savoir quels sont les graphes qui ne vérifient pas l'inégalité du corollaire 2.2. Le théorème de Brooks, qui est le principal résultat de cette section, montre que les exceptions sont des graphes très particuliers.

Théorème de Brooks

Avant de donner la démonstration du théorème de Brooks nous avons besoin du lemme suivant,

Lemme 2.1

Soit $G = (V_G, E_G)$ un graphe simple connexe sans points d'articulation (i.e. 2-connexe), non complet, vérifiant $\Delta(G) \geq 3$ (donc $n \geq 3$). Alors il existe un sommet x ayant deux voisins x_1 et x_2 non adjacents tels que $G[V_G \setminus \{x_1, x_2\}]$ soit connexe.

Théorème 2.1 (Brooks[30][1] [7])

Soit G un graphe simple connexe. Si G n'est ni un cycle d'ordre impair, ni un graphe complet, alors $\chi(G) \leq \Delta(G)$.

Démonstration: Si $\Delta(G) \leq 2$, alors G est soit un chemin, soit un cycle d'ordre pair et c'est fini.
Supposons que $\Delta(G) \geq 3$. Si G n'est pas régulier, alors d'après le corollaire 2.2 l'assertion est vraie. Supposons G est k-régulier :

1. Si G admet un point d'articulation x, soient $C_1, C_2, C_3, ..., C_p, p \geq 2$, les composantes connexes de $G - x$. Soit, pour tout i, le sous-graphe induit $G_i = G[C_i \cup \{x\}]$.
   Le degré $d_{G_i}(x)$ de x dans chaque G_i est strictement inférieur à k car $k = \sum_{i=1}^{p} d_{G_i}(x)$.
   De plus pour tout $i \in \{1, 2, 3, ..., p\}$, G_i n'est pas régulier car le sommet x est adjacent aux autres composantes connexes, alors que les sommets de C_i ne le sont pas. Donc, d'après le corollaire 2.2, chaque G_i peut être colorié avec au plus k couleurs. En permutant les couleurs des colorations de chaque G_i de telle sorte que x ait toujours la même couleur pour chaque G_i, on peut construire une coloration de G avec au plus k couleurs.
2. Si G n'admet pas de point d'articulation, d'après le lemme 2.1, il existe un sommet x ayant deux voisins x_1 et x_2 non adjacents tels que $G[V \setminus \{x_1, x_2\}]$ soit connexe. Par conséquent, d'après la proposition 1.2, il admet un graphe partiel qui est un arbre, on ordonne les sommets de cet arbre (et donc de $G[V \setminus \{x_1, x_2\}]$) de la façon suivante : $t_1 = x$ est le plus grand élément, puis on ordonne (arbitrairement) les voisins de x : $t_2 > t_3 > \cdots > t_\alpha$, puis les sommets voisins des t_i dans G : $t_{\alpha+1} > t_{\alpha+2} > ... > t_\beta$, etc. jusqu'à t_{n-2}, on complète cet ordre en forçant $t_{n-2} > t_{n-1} = x_1 > t_n = x_2$. Ainsi V_G est ordonné :

   $$x = t_1 > t_2 > t_3 > \cdots > t_{n-2} > t_{n-1} = x_1 > t_n = x_2$$

   de sorte que

   $$\forall i, 2 \leq i \leq n, \exists t_j > t_i \text{ tel que } t_i \text{ soit adjacent à } t_j$$

   .
   On construit maintenant une coloration $f : V \longrightarrow \{1, 2, ..., k\}$:
   $f(t_n) = f(t_{n-1}) = 1$, $f(t_{n-2}) = 2$, cela est possible car x_1 et x_2 ne sont pas adjacents, ensuite t_{n-2} a un voisin plus grand que lui, donc au plus $k - 1$ voisins plus petits (ici ce sont au plus t_{n-1} et t_n), on peut donc le colorier avec l'une des k couleurs, de façon générale t_i, ayant un voisin plus grand, a au plus $k - 1$ voisins plus petits, on peut donc le colorier, sans conflit, avec l'une des k couleurs, on arrive ainsi à t_2 ; enfin $t_1 = x$ a k voisins, dont t_2, t_{n-1}, t_n, mais ces deux derniers sont de la même couleur, donc les voisins de x nécessitent au plus $k-1$ couleurs et on peut le colorier sans conflit.

□

Example 2.4. *Ce graphe (figure 2.8) possède un sous-graphe qui est un cycle d'ordre* 3*, donc* $\chi(G) \geq 3$.
Il n'est pas complet et n'est pas non plus un cycle d'ordre impair, donc par Brooks, $\chi(G) \leq 3$. *Donc* $\chi(G) = 3$.

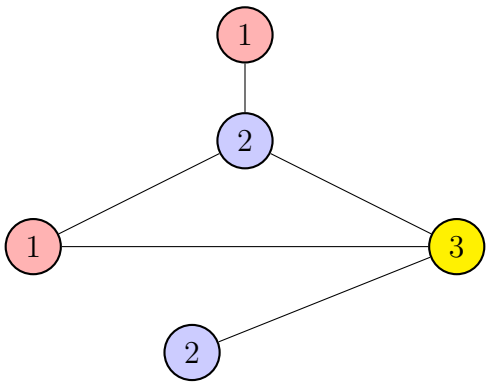


FIGURE 2.8 – Exemple d'application de théorème de Brooks $\chi(G) = 3$.

La borne obtenue par le théorème précédent peut être améliorée en considérant le degré minimum des sous-graphes induits :

Théorème 2.2 (Halin 1967)

Soit G un graphe simple k-dégénéré. Alors $\chi(G) \leq k + 1$.

Remarque 2.3. *On remarque que tout arbre est* 1*-dégénéré (on l'épluche par ses feuilles...). Dans le cas des arbres justement, le Théorème de Halin donne une borne de nombre chromatique égal à* 2*, donc optimale, alors que le Théorème de Brooks donne un majorant arbitrairement grand (le degré maximum de l'arbre).*

2.5 Coloration de quelques classes de graphes

2.5.1 Graphes planaires

Définition 2.6

Un graphe est planaire s'il est possible de le représenter dans le plan de sorte que les arêtes ne se coupent pas.

Example 2.5. *Considérons les graphes suivants :*

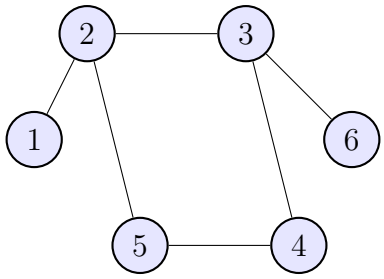


FIGURE 2.9 – Exemple de graphe planaire.

Ce graphe est clairement planaire, car il n'existe pas d'intersection entre deux arêtes.

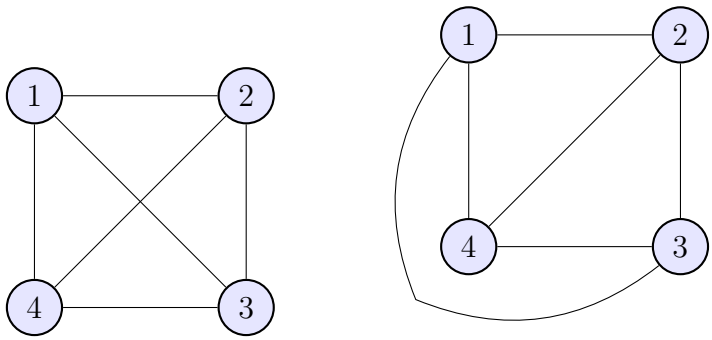


FIGURE 2.10 – Le graphe K_4 est planaire .

C'est un graphe complet à quatre sommets (K_4). Il est planaire : si on déplace le sommet 4 dans le triangle $[1, 2, 3]$, on constate qu'il n'y a plus d'intersection d'arêtes.

Example 2.6. *Un exemple simple pour illustrer l'intérêt des graphes planaires est une énigme, dite des trois maisons initialement posée sous forme mathématique par H. E. Dudeney en 1917 [16]. Elle prend la forme suivante : « Un lotissement de trois maisons doit être équipé d'eau de gaz et d'électricité. La réglementation interdit de croiser les canalisations pour des raisons de sécurité. Comment faut-il faire ? »*

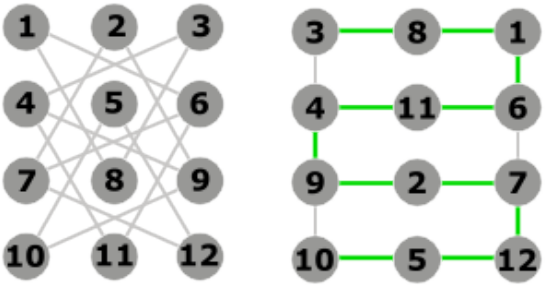


FIGURE 2.11 – Un graphe de mouvement brut à gauche, le même graphe réorganisé à droite.

Définition 2.7

Un graphe planaire découpe le plan en plusieurs régions.
Une **face** d'un graphe est par définition une région du plan limitée par des arêtes et qui ne contient ni sommet, ni arête dans son intérieur.

Example 2.7. *Un arbre n'admet qu'une face : le plan en entier. En fait, il est possible de trouver un chemin continu de n'importe quel point du plan a un autre sans couper une arête de l'arbre. C'est possible car l'arbre n'a pas de cycles.*

Example 2.8. *Considérons le graphe suivant :*

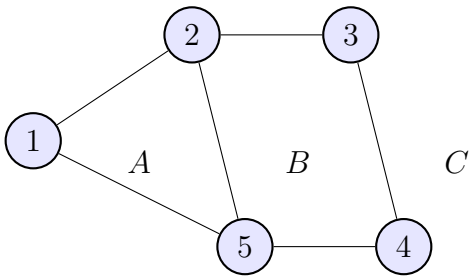


FIGURE 2.12 – Exemple de graphe planaire.

Ce graphe planaire comporte 5 *sommets,* 6 *arêtes, divise le plan en* 3 *faces* A*,* B *et* C*. On remarque que les faces* A *et* B *sont limitées, alors que la face* C*, extérieure, est illimitée.*

En 1852, Francis Guthrie posa le fameux "**problème des quatre couleurs**" : est-il possible de colorier n'importe quelle carte en quatre couleurs de façon telle que les pays ayant une frontière commune aient des couleurs distinctes ? Ce problème fut communiqué à quelques mathématiciens célèbres de l'époque (De Morgan, Hamilton, Cayley, etc.).
En 1879, Kempe en proposa une solution qui fut démontrée fausse en 1890 par Heawood (dans son premier article). Ce n'est qu'en 1976 que Appel et Haken produisirent une preuve de cette conjecture, preuve utilisant massivement l'ordinateur.
Ce problème de coloration de carte peut facilement être transformé en un problème de coloration des sommets d'un graphe : on associe un sommet à chaque pays de la carte (sa capitale), deux sommets étant reliés si et seulement si les pays correspondants ont une frontière commune. Toute coloration correcte de la carte correspond alors à une coloration propre du graphe associé, et réciproquement : On s'aperçoit aisément que le graphe associé à une telle carte est nécessairement planaire. Ainsi, le problème des quatre couleurs peut également être formulé ainsi :

Tout graphe planaire est-il coloriable avec quatre couleurs ?

Une face d'un graphe planaire dessiné est une portion connexe du plan délimitée par des arêtes. La formule d'Euler fournit un invariant des graphes planaires très utile :

Théorème 2.3 (Formul d'Euler [40, 37])

Si G est un graphe planaire dessiné ayant n sommets, m arêtes et f faces, alors $n - m + f = 2$.

Démonstration: Par récurrence sur $m - n$.
Pour $m - n$=-1, G est donc un arbre donc la propriété est vrai.
Supposons que la formule est vrai jusq'à $m - n = k$, soit G tel que $m - n = k + 1 \geq 0$, le graphe G contient donc un cycle, soit e une arête appartenant à ce cycle, et G' le graphe $G - e$.
Dans G', nous avons $n' - m' + f' = 2$, soit $n - (m-1) + (f-1) = 2$, d'où $n - m + f = 2$. □

Proposition 2.6

1. Tout graphe planaire à n sommets, $n \geq 3$, possède au plus $3n - 6$ arêtes.
2. Tout graphe planaire contient un sommet de degré au plus 5.

Démonstration: 1. Chaque face est bordée par au moins 3 arêtes, d'où $f \leq 2m/3$. Par la formule d'Euler, nous avons alors $m + 2 = n + f \leq n + 2m/3$, soit $m \leq 3n - 6$.

2. Raisonnons par l'absurde et supposons que G est un graphe planaire dont tous les sommets sont de degré au moins 6. Par le Lemme 1.1, nous avons alors $2m = \sum_{v \in V_G} d_G(v) \geq 6n$. Par ailleurs, toute face étant bordée par au moins trois arêtes, nous avons $2m \geq 3f$. En injectant les deux inégalités $n \leq m/3$ et $f \leq 2m/3$ dans la Formule d'Euler, nous obtenons :

$$2 = n - m + f \leq \frac{m}{3} - m + \frac{2m}{3} = 0$$

d'où la contradiction.

□

Ainsi, les graphes planaires sont 5-dénégérés, et par le Théorème de Halin 2.2, nous obtenons :

Corollaire 2.3

Tout graphe planaire est 6-coloriable.

La technique utilisée par Kempe[..] dans sa preuve fausse du théorème des quatre couleurs permettait cependant de démontrer que tout graphe planaire est 5-coloriable :

Théorème 2.4 (Kempe)

Tout graphe planaire est 5-coloriable.

Démonstration: Par récurrence sur le nombre de sommets du graphe $G = (V_G, E_G)$. Le résultat étant trivialement vrai si G a un seul sommet. Par 2 de la proposition 2.6, G contient un sommet x de degré au plus 5 ($d_G(x) \leq 5$). Par hypothèse de récurrence, le graphe $G - x$ (le graphe obtenu en supprimant le sommet x dans G) est 5-coloriable. Soit c une coloration de $G - x$, à partir de c nous allons construire une 5-coloration de G.
Si une des 5 couleurs, disons i, n'est attribuée à aucun voisin de x alors on peut étendre c en posant $c(x) = i$.(C'est en particulier le cas si $d_G(x) \leq 4$).

Nous pouvons supposer donc que les voisins de x, notés dans l'ordre cyclique x_1, x_2, x_3, x_4, x_5, et que ceux-ci sont colorés d'une couleur différente. Quitte à renommer les couleurs on peut supposer que $c(x_i) = i$.
Notons G_{ij}, pour $1 \leq i < j \leq 5$, le sous-graphe de $G - x$ engendré par les sommets de couleur i ou j.
Supposons que x_1 et x_3 n'appartiennent pas à la même composante connexe de $G_{1,3}$, on suppose que $x_1 \in C^1_{1,3}$ et $x_3 \in C^3_{1,3}$. Dans ce cas, en échangeant les couleurs des sommets de la composante $C^1_{1,3}$ (1 devient 3, 3 devient 1), et colorer x avec la couleur 1, ainsi on obtient une 5-coloration de G.

Si x_1 et x_3 appartiennent à la même composante connexe $C_{1,3}$ de $G_{1,3}$, alors il existe un chemin bicolore [1] $P_{1,3}$ reliant x_1 à x_3 dans $C_{1,3}$ (de tels chemins sont appelés des chaînes

1. Un chemin bicolore est un chemin dont les sommets sont coloriées avec deux couleurs (disons jaune et violet). On cherche des chemins dont les couleurs alternent : un sommet jaune est toujours suivie d'un sommet violet, et vice-versa.

de Kempe), ce chemin forment un cycle C avec les arêtes xx_1 et xx_3, et x_2 et x_4 sont séparés par C. Alors la composante connexe $C^2_{2,4}$ de x_2 dans $G_{2,4}$ ne contient pas x_4, sinon une arête du chemin de x_2 à x_4 dans $C^2_{2,4}$ couperait une des arêtes de C (voir la figure 2.13).

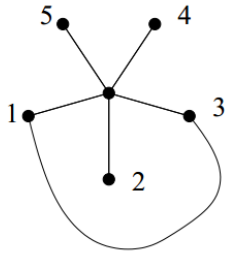


FIGURE 2.13 – 5-coloration de G.

On peut donc permuter les couleurs 2 et 4 dans $C^2_{2,4}$ et colorer x avec la couleur 2. □

La conjecture des quatre couleurs a été démontrée par Appel et Haken en 1976 [3] :

Théorème 2.5 ([3] Appel and Haken)

Tout graphe planaire est 4-coloriable.

La preuve de ce théorème est très compliquée. Elle consiste en une réduction à un certain nombre de graphes (plus de 1000) pour lequel le théorème a été prouvé à l'aide d'un calcul par ordinateur.

2.5.2 Graphes parfaits

Définition 2.8

On définit la classe des graphes parfaits comme la classe des graphes G tels que pour tout H sous-graphe de G, alors

$$\omega(H) = \chi(H)$$

Les théorèmes des graphes parfaits

Les graphes parfaits, outre leur utilité indéniable dans le monde de l'algorithmique, sont connus pour les 3 théorèmes suivants, qui furent pendant longtemps des conjectures. Chacun d'eux est aujourd'hui résolu, et nous nous attacherons à montrer certains d'entre eux, dans des cas plus ou moins généraux.

Théorème 2.6 (La Conjecture faible des graphes parfaits[29])

Soit G un graphe parfait. Alors $\overline{G}$ le complémentaire de G est également parfait.

Ce théorème a été énoncé par Berge et résolu 12 ans plus tard en 1972 par Laslo Lovasz.

Théorème 2.7 (La Conjecture forte des graphes parfaits [33, 38])

G est parfait si et seulement si ni G ni $\overline{G}$ ne contiennent de trou de longueur impaire.

Ce théorème conjecturé par Claude Berge en 1961, il faudra attendre plus de quarante ans (en 2002) pour que Maria Chudnovsky, Neil Robertson, Paul Seymour et Robin Thomas annoncent une preuve[33] , qu'ils publieront en 2006.
La démonstration de ce théorème fait aujourd'hui 148 pages...

Théorème 2.8 (La reconnaissance en temps polynomial des graphes parfaits [15])

Il existe un algorithme terminant en temps polynomial déterminant si un graphe simple G est parfait ou non.

Le problème de l'existence d'un algorithme de reconnaissance été résolus en 2002 par Seymour et Al. La preuve est particulièrement complexe et longue.

Quelques classes de graphes parfaits

Nous allons ici étudier plusieurs classes de graphes qui se trouvent être toutes contenues dans la classe des graphes parfaits. Nous rappellerons l'importance de chacune de ces classes, établiront l'inclusion dans la classe des graphes parfaits, et quand le résultat est simple à établir, nous montrerons le théorème fort ou faible des graphes parfaits dans le cas particulier de la sous-classe étudiée.

Graphes bipartis

Les graphes bipartis constituent l'une des classes les plus simples de graphes parfaits. Leur caractérisation extrêmement simple les rend facile à étudier, et les algorithmes les utilisant très rapides. Les graphes bipartis disposent d'application dans des domaines aussi variés que la chimie, les réseaux ou l'informatique théorique.
Nous verrons que les graphes bipartis possèdent des solutions très simples aux problèmes que nous nous posons sur les graphes parfaits.

Théorème 2.9 (Perfection des graphes bipartis [29][17])

Tout graphe biparti est également parfait.

Démonstration: Remarquons que tout sous-graphe d'un graphe biparti est lui-même biparti. Ainsi nous n'aurons pas à vérifier la condition de la définition des graphes parfaits. Il suffit de vérifier la condition d'égalité pour tout graphe biparti.
Soit $G = (S_1, S_2; E_G)$ un graphe biparti. On peut colorier le graphe en n'utilisant que deux couleurs, une couleur pour les sommets de S_1 et une pour les sommets de S_2. Si le graphe n'est pas trivial ($|E_G| \geq 1$), on a donc $\chi(G) = 2$. De plus, une clique dans un graphe biparti ne peut contenir plus de 2 sommets. En effet, soit C une clique contenant 3 sommets u, v et w. Supposons sans perte de généralité que $u \in S_1$, alors $uv \in E_G$ implique que $v \in S_2$. De plus on a $vw \in E_G$ (car C est une clique) donc $w \in S_1$. Enfin, de la même manière, $uw \in E_G$, donc $w \in S_2$. Or $S_1 \cap S_2 = \varnothing$, ce qui constitue la contradiction. Ainsi pour G, on a bien $\chi(G) = \omega(G)$. Donc G est parfait. □

On donne une autre caractérisation des graphes bipartis.

Théorème 2.10 (Caractérisation [29][17])

Un graphe biparti ne contient pas de cycle impair.

Démonstration: Si $G = (S_1, S_2; E_G)$ est un graphe biparti contenant un cycle impair $C = (x_1, ..., x_{2n+1}, x_1)$. Alors supposons sans perte de généralité que $x_1 \in S_1$. Alors pour tout $k, x_{2k} \in S_2$ et $x_{2k+1} \in S_1$ par induction immédiate. Or $x_{2n+1}x_1 \in E_G$, car on est en présence d'un cycle. Donc $x_{2n+1} \in U$ implique $x_1 \in S_2$, ce qui est absurde. Il n'existe donc pas de tel graphe biparti. □

Et on donne maintenant une preuve dans le cas particulier des graphes bipartis de la conjecture forte des graphes parfaits.

Théorème 2.11 (Théorème fort des graphes parfaits - graphes bipartis [29][17])

Si $G = (S_1, S_2; E_G)$ est un graphe biparti, ni lui ni $\overline{G}$ ne contient de trou de longueur impaire.

Démonstration: Par la caractérisation précédente (Théorème 2.10), G ne contient pas de cycle impair, donc à fortiori aucun trou impair.
Considérons les arêtes de $\overline{G}$. Soit e une telle arête. Elle relie, soit 2 sommets x et y de S_i avec le même i. Soit elle relie un sommet de S_1 et un de S_2 qui n'étaient pas reliés dans G. Soit $(x_1; :::; x_{2k+1})$ un trou impair $(2 \leq k)$. Deux possibilités : Si tous les $x_k \in S_i$ avec $i = 1, 2$. Alors on sait que ce n'est pas possible car aucun des sommets n'étant reliés dans

G, on a à la place dans $\overline{G}$ un maillage de triangles. Donc $(x_1, ..., x_{2k+1})$ ne peut être un trou de taille supérieure à 4. Parmi les x_k on ne peut trouver que 2 sommets provenant de S_1 et 2 sommets provenant de S_2, car à partir de 3, les 3 seraient reliés, donc on aurait une corde au milieu du trou. Donc, le trou est nécessairement de taille 4 (c'est la taille la plus petite possible), il n'est donc pas impair. □

Graphes de comparabilité

Définition 2.9

En mathématiques, **un préordre** est une relation binaire réflexive et transitive. C'est-à-dire que si E est un ensemble, une relation binaire $\mathcal{R}$ sur E est un pré-ordre lorsque :
- $\forall x \in E \quad x\mathcal{R}x$ (réflexivité) ;
- $\forall (x, y) \in E^2, (x\mathcal{R}y \wedge y\mathcal{R}z) \Rightarrow x\mathcal{R}z$ (transitivité).

Définition 2.10

Un **graphe de comparabilité** est un graphe associé à une relation de pré-ordre de la façon suivante :
soit $\leq$ un pré-ordre sur l'ensemble X, on définit le graphe simple $G = (V_G, E_G)$ en prenant :
- $V_G = X$;
- $xy \in E_G$ si et seulement si $x \leq y$ ou $y \leq x$ (i.e. x et y sont distincts et comparables).

Example 2.9. *Le graphe représenté en figure 2.14 est un graphe de comparabilité.*

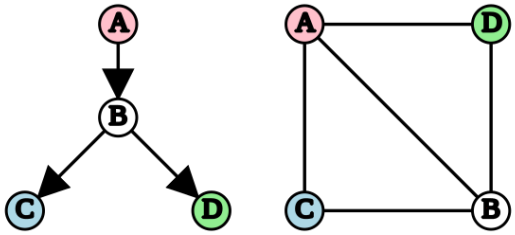


FIGURE 2.14 – Une représentation d'un ordre partiel et le graphe de comparabilité associé.

Example 2.10. *Le graphe représenté en figure 2.15 n'est pas un graphe de comparabilité : quel que soit l'ordre choisi sur les sommets du triangle, l'une des arêtes adjacentes à ce triangle ne pourra être orientée correctement : il manquera l'arête de transitivité.*

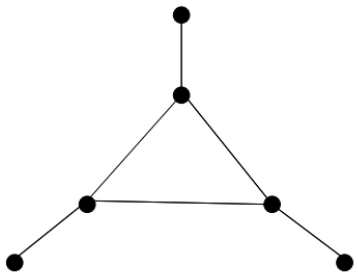


FIGURE 2.15 – Un graphe qui n'est pas un graphe de comparabilité.

Théorème 2.12 (Perfection des graphes de comparabilité [29][17])

Les graphes de comparabilité sont parfaits.

Démonstration: Comme tout sous-graphe induit d'un graphe de comparabilité est de comparabilité, il suffit de voir que $\chi(G) = \omega(G)$ pour un graphe de comparabilité $G = (V_G, E_G)$ associé au préordre $\leq$: soit $x \in V_G$, notons $r(x)$ le plus grand entier r pour lequel il existe r sommets $x_i, i = 1, \ldots, r$, tels que $x_1 \leq x_2 \leq \ldots \leq x_r = x$. Notons que la transitivité de $\leq$ implique que le sous-graphe induit par les sommets x_i est complet, donc c'est une clique de G. Soit $k = \max_{x \in V_G} r(x)$ et $y \in V$ tel que $r(y) = k$. La taille d'une clique maximum est donc $\omega(G) = k$ et l'application

$$r : V \longrightarrow \{1, 2, 3, \ldots, k\}$$

est une coloration de G : si $x \leq y$ il est clair que $r(x) \leq r(y)$, donc si $xy \in E_G$ on a $r(x) \neq r(y)$. Ainsi $\chi(G) \leq \omega(G)$ puis $\chi(G) \geq \omega(G)$ par la proposition 2.4. D'où l'égalité . □

Graphes d'intervalles

En théorie des graphes, un graphe d'intervalles est le graphe d'intersection d'un ensemble d'intervalles de la droite réelle. Chaque sommet du graphe d'intervalle représente un intervalle de l'ensemble, et une arête relie deux sommets lorsque les deux intervalles correspondants s'intersectent. Plus formellement :

Définition 2.11

Soit $\{I_1, I_2, ..., I_n\}$ un ensemble d'intervalles de $\mathbb{R}$. On lui associe un graphe simple $G = (V_G, E_G)$ tel que :
- $V_G = \{I_1, I_2, ..., I_n\}$
- pour tout $1 \leq i, j \leq n$, $(i \neq j)$, $I_i I_j \in E_G$ si et seulement si $I_i \cap I_j \neq \varnothing$.

Ce graphe est appelé **graphe d'intervalles**.

Example 2.11. *Considérons un ensemble de tâches (Figure 2.16) ayant chacune une heure de début et une heure de fin bien précises. Supposons qu'on demande à des employés d'effectuer ces tâches, mais qu'aucun employé ne puisse effectuer deux tâches à la fois. Pour modéliser cette situation il suffit de considérer le graphe dans lequel chaque tâche est représentée par un sommet et une arête relie deux sommets lorsque les tâches correspondantes se chevauchent dans le temps. Le graphe ainsi construit est un graphe d'intervalles (Figure 2.17).*

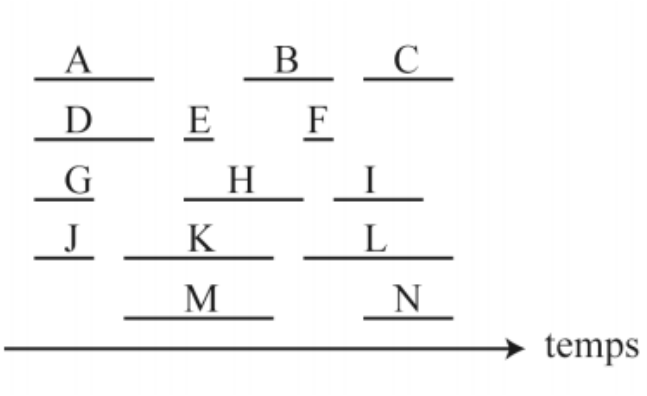


FIGURE 2.16 – L'ensemble des 14 taches à effectuer.

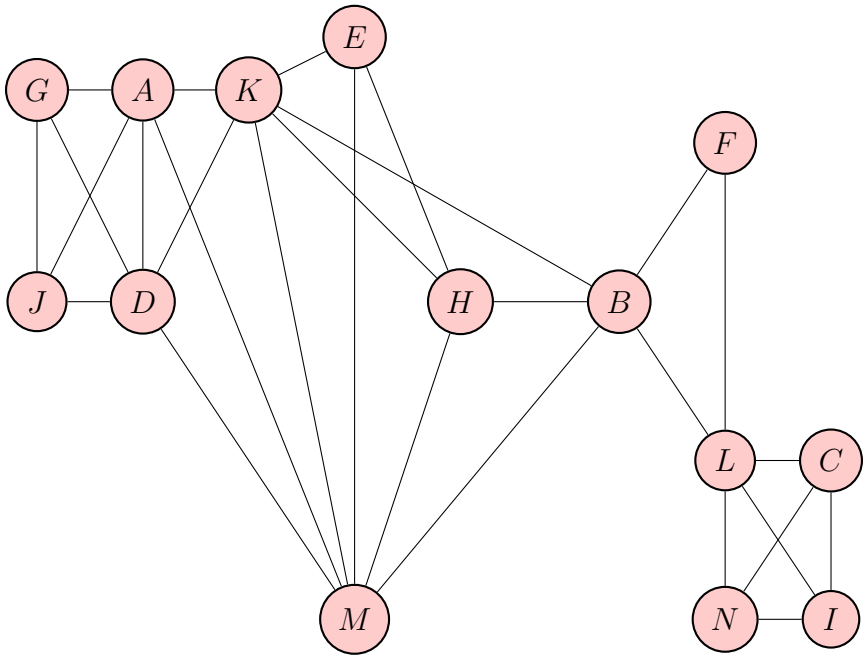


FIGURE 2.17 – Graphe d'intervalles correspondant.

Intérêt des graphes d'intervalle : les graphes d'intervalle sont tout indiqués pour représenter les problèmes d'ordonnancement. En effet, un problème d'ordonnancement se représente souvent comme un certain nombre de tâches, avec un temps de début,de fin, une durée, des chevauchements temporels de tâches. C'est exactement ce qu'est un graphe d'intervalle : un graphe associé à un ensemble d'intervalles qui peuvent s'intersecter.

Proposition 2.7 ([29][17])

Le complémentaire d'un graphe d'intervalles est un graphe de comparabilité.

Démonstration: Soit $G = (V_G, E_G)$ un graphe d'intervalles.
Définissons un préordre sur $V_G = \{I_1, I_2, \ldots, I_n\}$ de la manière suivante :

$I_i \leq I_j$ si et seulement si $i = j$ ou $\forall x \in I_i, \forall y \in I_j : x < y$.
Ce préordre fournit un graphe de comparabilité $H = (V_G, E_H)$. Soit $i = j$:

– si $I_iI_j \in E_H$ on a $I_i \leq I_j$ ou $I_j \leq I_i$ donc $I_i \cap I_j = \varnothing$ et $I_iI_j \notin E_G$,
– si $I_iI_j \notin E_G$, alors $I_i \cap I_j = \varnothing$, comme $\mathbb{R}$ est totalement ordonné,
on a $\forall x \in I_i$, $\forall y \in I_j$, $x < y$ ou le contraire, i.e. $I_i \leq I_j$ ou $I_j \leq I_i$ d'où $I_iI_j \in E_H$, donc $H = \overline{G}$ est un graphe de comparabilité.

□

Théorème 2.13 (Perfection des graphes d'intervalles [29][17])

Tout graphe d'intervalles est parfait.

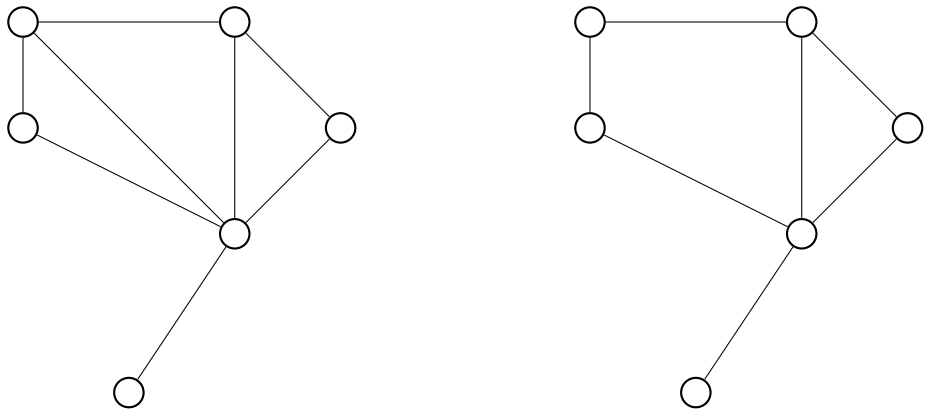
Démonstration: La preuve va être facile à faire car il y a une notion de précédance. On va pouvoir colorier de la gauche vers la droite.
Soit $G = (V; E; I)$ un graphe d'intervalle et $k = \omega(G)$ couleurs. On va puiser dans ces couleurs pour colorier les intervalles. Chaque intervalle $I_v \in I$ est noté $I_v = [v_1, v_2]$.
On commence par colorier l'intervalle le plus à gauche avec la première couleur. Supposons maintenant être au début de l'intervalle I_v et que pour l'on a réussi à colorier avec moins de k couleurs tout intervalle $I_u \in I$ tel que $u_1 \leq v_1$. Alors comme on a supposé que tous les intervalles avaient des bornes distinctes, tout intervalle contenant v_1 a débuté avant v_1. Le nombre n d'intervalles s'intersectant en v_1 est nécessairement $\leq k$ car k est le cardinal maximum d'une clique de G, ce qui on le rappelle correspondant au nombre maximal d'intervalles s'intersectant en un point de $\mathbb{R}$. Donc $n \leq k$. On a réussi à colorier les $n-1$ intervalles intersectant I_v en v_1 (avec donc au plus $n-1 \leq k$ couleurs), il suffit donc de colorier l'intervalle I_v avec une des couleurs disponibles restante.
Par principe d'induction, on a réussi à montrer qu'une coloration avec au plus k couleurs est possible.
Ainsi, comme la classe des graphes d'intervalle possède la propriété d'hérédité (i.e. tout sous graphe d'un graphe intervalles est un graphe d'intervalle), car lorsque l'on enlève un sommet, on obtient toujours un graphe d'intersection d'intervalles, donc tout graphe d'intervalle est parfait. □

Les graphes triangulés

Définition 2.12

Un graphe est **triangulé** si tout cycle élémentaire de longueur au moins 4 admet une corde.

Example 2.12. *Le graphe (a) de la figure 2.20 est un graphe triangulé, par contre (b) n'est pas triangulé, car il contient un cycle longueur* 4 *qui n'a pas de corde.*



(a) **(b)**

FIGURE 2.18 – (a) Graphe triangulé, (b) n'est pas triangulé

Proposition 2.8

Les graphes d'intervalles sont des graphes triangulés.

Démonstration: Soit $G = (V_G, E_G)$ un graphe d'intervalles avec $V_G = \{I_1, I_2, ..., I_n\}$. Si par exemple $C = [I_1, I_2, ..., I_k, I_1]$ $(k \geq 4)$ est un cycle élémentaire dans G, alors tout i $(1 \leq i \leq k-1)$ on a

$$I_i \cap I_{i+1} \neq \varnothing$$

de plus $I_k \cap I_1 \neq \varnothing$, donc tous les intervalles $I_2, I_3....I_{k-1}$ coupent $I_1 \cup I_k$, en particulier pour I_2 et I_3 on a , $I_2 \cap I_k \neq \varnothing$ ou $I_3 \cap I_1 \neq \varnothing$, dans les deux cas C contient une corde (I_2I_k ou I_1I_3). Donc G est triangulé.

□

Définition 2.13

Un sous-ensemble S de sommets dans un graphe connexe $G = (V_G, E_G)$ est un **séparateur** si $G - S$ (le sous-graphe induit par $V_G \setminus S$) n'est pas connexe, i.e. tel qu'il existe deux sommets $a, b \in V_G \setminus S$ qui ne sont pas reliés par un chemin dans $G - S$, S est dit aussi un (a, b)-séparateur de G.
On dit que S est un **séparateur minimal** de G si S est minimal pour l'inclusion parmi tous les séparateurs de G.

Example 2.13. *Considérons le graphe suivant*

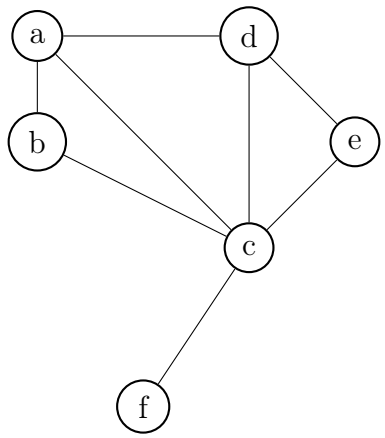


FIGURE 2.19 – Un graphe connexe G

Dans ce graphe $S_1 = \{d, c\}$ *est un séparateur de* G, $S_2 = \{c\}$ *est un séparateur minimal de* G

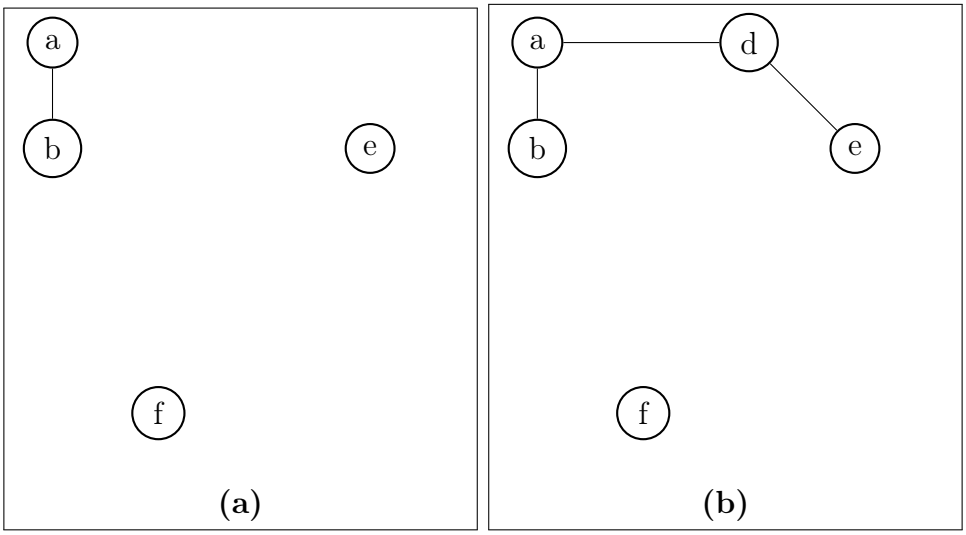


FIGURE 2.20 – (a) le graphe $G - S_1$, (b) le graphe $G - S_2$

Théorème 2.14

Soit G un graphe connexe.
Le graphe G est triangulé si et seulement si tout séparateur minimal de G est une clique.

Démonstration: Supposons tout d'abord que tout séparateur est une clique. Soit $C = [x_1, x_2, ..., x_k, x_1](k \geq 4)$ un cycle dans G et soit S un séparateur minimal de x_1 et x_3. S doit contenir x_2 et au moins un des sommets $x_4, ..., x_k$. Comme S est une clique, il existe une corde dans C. Inversement, supposons que G est triangulé et soit S un séparateur minimal. Supposons que S ne soit pas une clique. Soient G_1 et G_2 deux composantes connexes de $G - S$ et soient x et y deux sommets non adjacents dans S. Comme S est minimal, x et y ont chacun au moins un voisin dans C_1 et dans C_2. Soient a_1 et a_2 les voisins de x dans C_1 et C_2, et soient b_1 et b_2 ceux de y dans C_1 et C_2 (voir la figure2.21) . Comme C_1 et C_2 sont connexes, il existe un chemin reliant a_1 à b_1 dans C_1 et un chemin reliant a_2 à b_2 dans C_2.

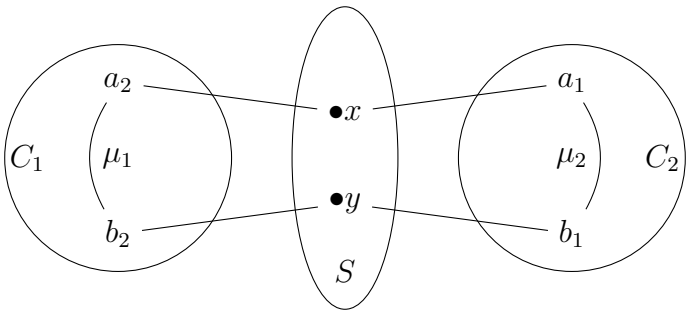
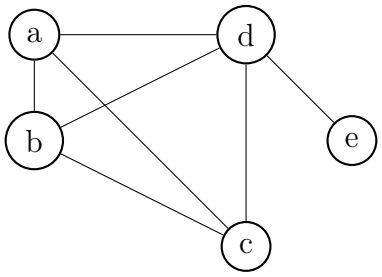


FIGURE 2.21 – Les deux composantes connexes de $G - S$.

Il existe donc un chemin μ_1 sans corde reliant x à y dans $G-S$ ainsi qu'un chemin sans corde μ_2 reliant x à y dans $G - S$. L'union de μ_1 et μ_2 est un cycle sans corde contenant au moins 4 sommets, contradiction. □

Définition 2.14

Un sommet x d'un graphe simple G est dit **simplicial** si son voisinage $N_G(x)$ est une clique.



Le sommet c est simplicial dans ce graphe

FIGURE 2.22 – Un exemple d'un sommet simplicial

Théorème 2.15

Soit G un graphe triangulé d'ordre n, $(n \geq 2)$. Si G n'est pas complet alors il possède au moins deux sommets simpliciaux non adjacents.

Démonstration: Si G ne contient que deux sommets, alors G est constitué de deux sommets isolés qui sont simpliciaux non adjacents. Supposons donc le théorème vrai pour tout graphe ayant moins de n sommets. Soit S un séparateur minimal et C_1 et C_2 deux composantes connexes de $G[V - S]$. On a vu que S est une clique (théorème 2.14).

- Si $C_1 \cup S$ est une clique alors choisissons x dans C_1 : x est simplicial dans $G[C_1 \cup S]$.
- Sinon, par hypothèse d'induction, il existe deux sommets simpliciaux non adjacents dans $G[C_1 \cup S]$, et comme S est une clique, l'un de ces sommets qu'on appellera x est dans C_1.

Dans chacun des deux cas on a déterminé un sommet x simplicial dans $G[C_1 \cup S]$. De même, on peut déterminer un sommet y simplicial dans $G[C_2 \cup S]$. Ces deux sommets x et y sont simpliciaux dans G et non-adjacents. □

Définition 2.15

Soit $G = (V_G, E_G)$ un graphe simple d'ordre n.
Un **schéma d'élimination parfait** (ou bien un **ordre d'élimination simplicial**) est une énumération $v_1, v_2, ..., v_k$ des sommets de V_G, tel que pour tout $1 \leq i \leq n$, v_i est simplicial dans $G[\{v_i, ..., v_n\}]$.

Théorème 2.16 (Fulkerson et Gross [29][17])

Un graphe est triangulé si et seulement si il possède un schéma d'élimination parfait.

Démonstration: Supposons que G est triangulé, on procède par induction sur le nombre de sommets de G pour montrer que G admet un schéma d'élimination parfait. Pour $n = 1$ c'est trivial.
Supposons que $n > 1$ et pour tout graphe triangulé ayant à moins de n sommets admet un schéma d'élimination parfait. G est triangulé, donc d'après le théorème 2.15 il possède un sommet simplicial noté v. $G - v$ est donc triangulé ayant à moins de n sommets, par hypothèse d'induction, $G - v$ admet un schéma d'élimination parfait noté μ. Soit μ_v un ordre de sommets de G qui commence par v et suivi par les sommets dans l'ordre déterminé par μ. d'où μ_v est un schéma d'élimination parfait de G.

Inversement, supposons que G admet un schéma d'élimination parfait donné par $v_1, v_2, ..., v_n$. Soit $C = \{x_1, x_2, ..., x_k, x_1\}$ un cycle élémentaire arbitraire avec $k \geq 4$.
Sans perte de généralité on peut supposer que $x_1 = v_i$ apparaît avant $x_2, ..., x_k$ dans le schéma d'élimination parfait. Mais alors x_2 est relié à x_k car x_1 est simplicial dans le graphe $G[\{v_i, ..., v_n\}]$ qui contient $x_2, ..., x_k$, le cycle C a donc une corde. Donc G est triangulé. [38] [33][38] [33] [29] □

Proposition 2.9

1. Tout sous-graphe induit d'un graphe triangulé est triangulé (propriété d'hérédité).
2. Tout graphe triangulé est parfait.

Démonstration: Soit $G = (V_G, E_G)$ un graphe simple.

1. Si μ est une schéma d'élimination parfait de G, alors $\mu - x$ est un schéma d'élimination parfait de $G - x$, car enlever un sommet ne détruit pas les cliques, ainsi un sous graphe induit de G n'être que un graphe $G - S$ où $S \subset V_G$, d'où le résultat.
2. Si $\omega(G) = k$, pour colorier G avec k couleurs il suffit de traiter les sommets de G dans l'ordre donné par un schéma d'élimination parfait, vérifie le même invariant que pour les graphes d'intervalles lorsque l'on veut colorier x, ses seuls voisins déjà coloriés sont des prédécesseurs dans le schéma d'élimination parfait, et ils forment donc une clique, on a réussi à montrer qu'une coloration avec au plus k couleurs est possible, donc $\chi(G) = k = \omega(G)$.
   Ainsi, comme la classe des graphes triangulé possède la propriété d'hérédité (d'aprés 1.), donc tout graphe triangulé est parfait.

□

2.6 Conclusion

Dans ce chapitre nous avons étudié le problème de la coloration des graphes, ainsi la caractérisation et la coloration des différentes classes de graphes à savoir les graphes planaires et les graphes parfaits. Les relations d'inclusion entre ses classes de graphes sont résumées dans la figure 2.23

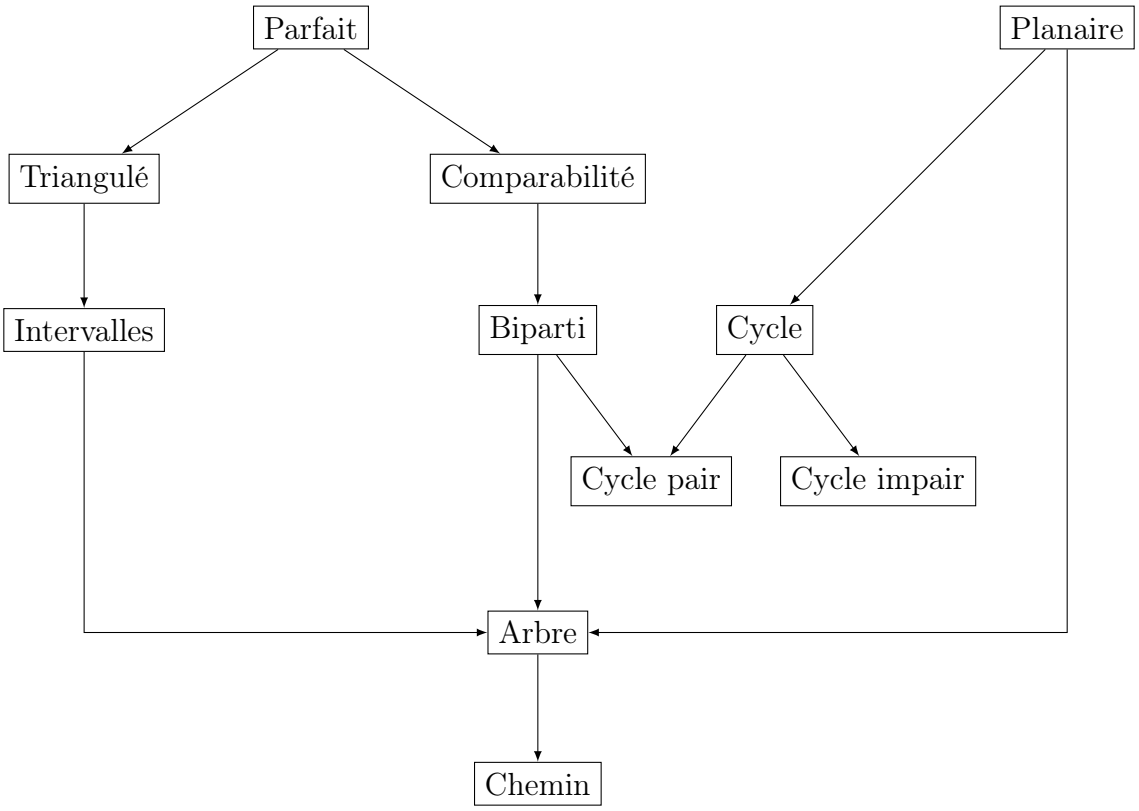


FIGURE 2.23 – Relations d'inclusion de différentes classes de graphes classiques : $A \rightarrow B$ signifie que $B \subset A$

Ces classes de graphes sont très importantes dans l'étude générale de la coloration de graphes. Différentes propriétés ont été montrées sur ces classes particulières de manière élémentaire, et bien que des résultats généraux existent.

CHAPITRE 3

Algorithmes et complexité

3.1 Introduction

Reconnaître une classe de graphes, c'est donner un algorithme permettant de décider si un graphe appartient ou non à la classe. Il est toujours intéressant de disposer de ce genre d'algorithme. En effet, est-il vraiment utile de disposer d'algorithmes performants de coloration pour une classe si l'on est même pas capable de décider si l'algorithme fonctionne pour un graphe donné ?
Nous présentons dans ce chapitre quelques algorithmes de reconnaissances et de coloration pour différentes classes de graphes (biparti,parfait,intervalles,triangulé...). Ainsi, nous allons étudier la complexité de chaque algorithme.

3.2 Algorithmes de parcours

Les **piles** (stack) et les **files** (queue) constituent des structures de données. Elles vont, comme leur nom l'indique, nous permettre de stocker diverses données, comme pourrait le faire un tableau.

Une **pile** permet de réaliser ce que l'on nomme une **LIFO** (Last In First Out), ce qui signifie en clair que les derniers éléments à être ajoutés à la pile seront les premiers à être récupérés.

Une **file**, quant à elle, permet de réaliser une **FIFO** (First In First Out), ce qui veut dire que les premiers éléments ajoutés à la file seront aussi les premiers à être récupérés.

3.2.1 Algorithme de parcours en largeur BFS

L'**algorithme de parcours en largeur** (ou BFS, pour Breadth First Search en anglais) permet le parcours d'un graphe (en particulier un arbre) de la manière suivante : on commence par explorer un nœud 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 nœuds depuis un nœud source dans un graphe simple. Il peut aussi servir à déterminer si un graphe est connexe (ou bien déterminer le nombre de ses composantes connexes).

Le principe de l'algorithme est à partir d'un sommet s, il liste d'abord les voisins de s pour ensuite les explorer un par un. Ce mode de fonctionnement utilise donc une file dans laquelle il prend le premier sommet et place en dernier ses voisins non encore explorés.

Les sommets déjà visités sont marqués afin d'éviter qu'un même sommet soit exploré plusieurs fois. Dans le cas particulier d'un arbre, le marquage n'est pas nécessaire.

Algorithme 1 Algorithme de parcours en largeur BFS

Entrées: un graphe $G = (V_G, E_G)$, un sommet x de G ;
Sorties: Un ordre λ sur les sommets de G ;
Initialisation :
Soit F une file vide
Ajouter x à F,$\lambda(1) = x, i = 2$
Tantque F est non vide **faire**
retirer le sommet x de F qui minimise λ ;
Pour tout voisin y non-numéroté de x **faire**
Ajouter y à F
$\lambda(i) = y$
$i = i + 1$
Fin pour
Fin tantque
Complexité : $O(n + m)$.

Example 3.1. *Sur le graphe suivant, cet algorithme va alors fonctionner ainsi :*

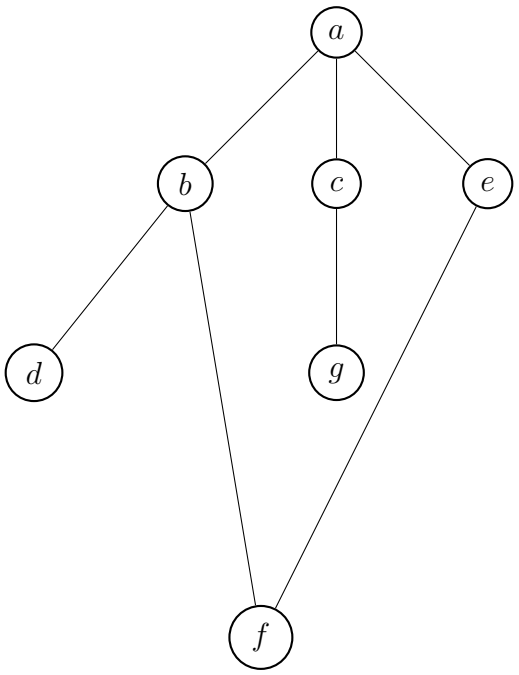


Figure 3.1 – Application de l'algorithme **BFS** pour un graphe

Il explore dans l'ordre les sommets a, b, c, e, d, f, g.

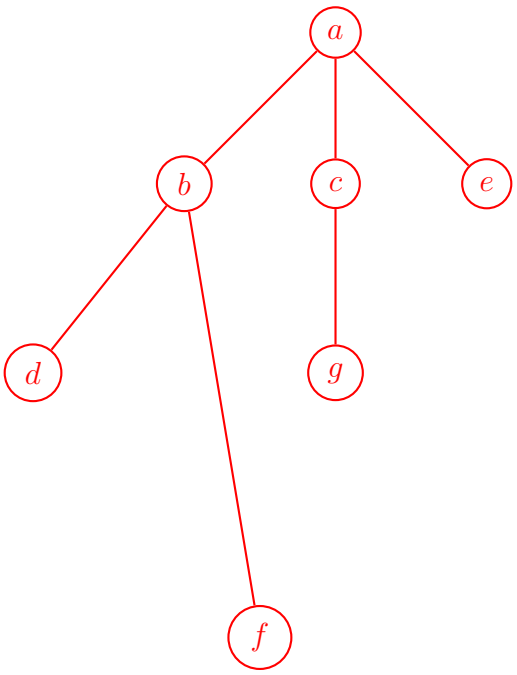


FIGURE 3.2 – Arbre couvrant.

3.2.2 Algorithme de parcours en profondeur DFS

L'**algorithme de parcours en profondeur** (ou DFS, pour Depth First Search) est un algorithme de parcours d'arbre, et plus généralement de parcours de graphe, qui se décrit naturellement de manière récursive. Son application la plus simple consiste à déterminer s'il existe un chemin d'un sommet à un autre.

C'est un algorithme de recherche qui progresse à partir d'un sommet s en s'appelant récursivement pour chaque sommet voisin de s.

Le nom d'algorithme en profondeur est dû au fait que, contrairement à l'algorithme de parcours en largeur, il explore en fait « à fond » 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.

Algorithme 2 Algorithme de parcours en profondeur DFS

Entrées: un graphe $G = (V_G, E_G)$, un sommet x de G ;
Sorties: Un ordre λ sur les sommets de G ;
Initialisation : i=1 ;
Fonction DFS(G,x)
$\lambda(i) = x$;
$i = i + 1$;
Pour tout voisin y non-numéroté de x **faire**
DFS(G,y) ;
Fin pour
Fin Fonction
Complexité : $O(n + m)$.

Example 3.2. *Voyons concrètement le fonctionnement de cet algorithme sur le graphe suivant :*

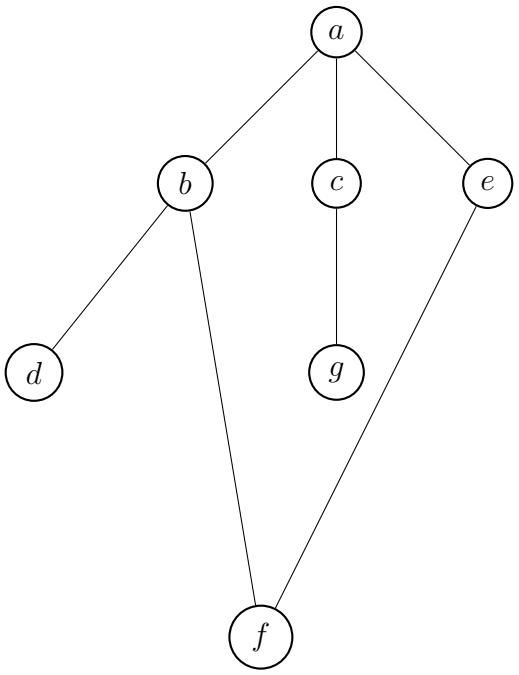


Figure 3.3 – Application de l'algorithme **DFS** pour un graphe

*L'algorithme **DFS** commence au sommet a, nous conviendrons que les sommets à gauche sur ce graphe seront choisis avant ceux de droite. Si l'algorithme utilise effectivement un marquage des sommets pour éviter de tourner indéfiniment en boucle, on aura alors l'ordre de visite suivant : a, b, d, f, e, c, g, contrairement à l'algorithme de parcours en largeur qui cherche dans cet ordre : a, b, c, e, d, f, g.*

Supposons maintenant que nous n'utilisions pas la méthode de marquage, on aurait alors la visite des sommets suivants dans l'ordre : a, b, d, f, e, a, b, d, f, e,.. etc, indéfiniment, puisque l'algorithme ne peut sortir de la boucle a, b, d, f, e et n'atteindra donc jamais c ou g.

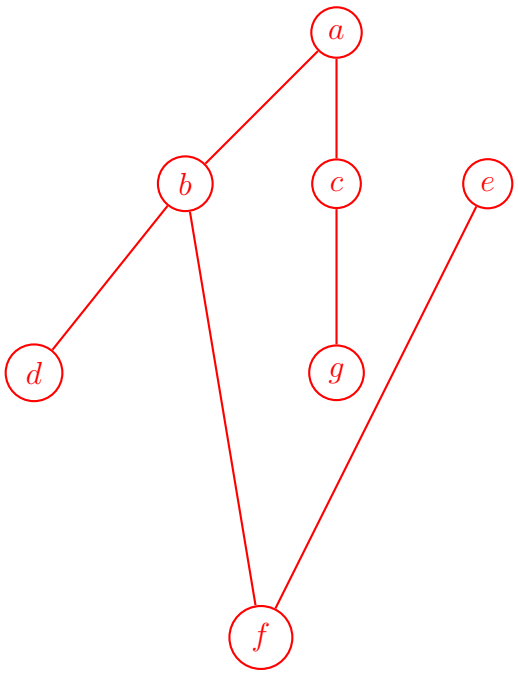


FIGURE 3.4 – Arbre couvrant.

3.3 Algorithmes de reconnaissance

Soit $\mathcal{G}$ une classe de graphe (par exemple les graphes parfaits).

Définition 3.1 (algorithme de reconnaissance)

Soit G un graphe simple quelconque.
Un **algorithme de reconnaissance** pour la classe de graphes $\mathcal{G}$ est un algorithme qui vérifie si le graphe G appartient à la classe $\mathcal{G}$ ou non. C'est-à-dire, soit l'algorithme accepte l'entrée G en sortant "**OUI**", soit l'algorithme refuse l'entrée G en sortant "**NON**".
Souvent l'algorithme sort une certaine représentation du graphe G si G est dans la classe $\mathcal{G}$.

Graphes bipartis

Soit $G = (V_G, E_G)$ un graphe simple connexe, on définit la **distance** entre deux sommets x et y de G par :
– $d_G(x, y)$ est la plus petite longueur de chaîne entre x et y, si $x = y$ on a $d_G(x, x) = 0$.
d appelée la distance de graphe G.

Les graphes bipartis constituent l'une des classes les plus simples de graphes. Leur caractérisation extrêmement simple les rend facile à étudier, et les algorithmes les utilisant très rapides. D'après le théorème de caractérisation des graphes bipartis (théorème 2.10, un graphe simple biparti ne contient pas de cycle de longueur impaire, le théorème suivant donne la condition suffisante.

Théorème 3.1

un graphe simple est biparti si et seulement s'il ne contient pas de cycle de longueur impaire.

Démonstration: Un graphe est biparti si et seulement si toutes ses composantes connexes sont biparties. On peut donc supposer que $G = (V_G, E_G)$ est connexe.
La **condition nécessaire** : voir le théorème 2.10.
La **conditions suffisante** : supposons réciproquement que G ne contient pas de cycle impair. Soit x un sommet de G choisi arbitrairement. Soit d la distance du graphe G, on définit $V_1 = \{y \in V_G / d(x, y)\ est\ pair\}$ et $V_2 = \{y \in V_G / d(x, y)\ est\ impair\}$ de sorte que $V_G = V_1 \cap V_2$ et $x \in V_1$. Montrons qu'il n'y a aucune arête entre les sommets de V_1, entre les sommets de V_2. Pour cela on raisonne par l'absurde :

- s'il existe une arête $a = uv$ avec $u, v \in V_1$, le graphe étant connexe, il existe C_u et C_v deux chemins élémentaires de longueur minimum reliant d'une part x à u et d'autre part x à v, par définition de V_1, $d(x, u)$ et $d(x, v)$ sont pairs, donc C_u et C_v ont des longueurs paires, les chemins C_u et C_v ont éventuellement des sommets communs : partant de x, il existe un dernier sommet z commun à C_u et C_v. Par minimalité, les longueurs des sous-chemins de C_u et C_v allant de x à z sont toutes deux égales à $d(x, z)$ et ont en particulier même parité. Il s'ensuit que les longueurs des sous-chemins de C_u et C_v joignant z à u et z à v, qui sont de longueurs respectives égales à $d(z, u)$ et $d(z, v)$ (à nouveau par minimalité), ont même parité. Ainsi la chaîne obtenue en allant de z à u en suivant C_u, puis de u à v selon uv, puis de v à z suivant C_v est de longueur $d(z, u) + 1 + d(v, z)$ impaire : contradiction.
- s'il existe une arête $a = uv$ avec $u, v \in V_2$, le graphe étant connexe, il existe C_u et C_v deux chemins élémentaires de longueur minimum reliant d'une part x à u, d'autre part x à v , par définition de V_2, $d(x, u)$ et $d(x, v)$ sont impairs, donc C_u et C_v ont des longueurs impaires , comme dans le cas précédent, il existe un sommet z commun à C_u et C_v tel que la chaîne obtenue en allant de z à u en suivant C_u, puis de u à v selon uv, puis de v à z suivant C_v est alors un cycle de longueur impaire : contradiction.

□

L'algorithme **GrapheBiparti** nous donne si un graphe simple quelconque est biparti (le graphe est connu à travers les listes A_y des sommets adjacents à y, $y \in V_G$).

Algorithme 3 Algorithme **GrapheBiparti**

Entrées: un graphe $G = (V_G, E_G)$, un sommet x de G ;
Sorties: biparti (booléen vrai ou faux), V_1 et V_2 (le cas échéant) ;

```
Initialisation :
L = ∅ ; (liste intermédiaire initialisée au vide)
d(x) = 0 ; (distance de x à x fixée à 0)
biparti = vrai, V1 = ∅ ; (variable booléenne, première partie de la partition des sommets) ;
Ajouter x à L ;
Tantque L n'est pas vide Et biparti == vrai faire
  Enlever le premier sommet y de L ;
  Pour tout u ∈ A_y faire
    Si d(u) n'est pas initialisée alors
      d(u) = d(y) + 1 ;
      Ajouter u à L ;
    Sinon
      Si d(u) == d(y) alors
        biparti = faux ;
      Finsi
    Finsi
  Fin pour
Fin tantque
Si biparti == vrai alors
  Pour tout x ∈ V_G faire
    Si d(x) ≡ 0 (mod 2) alors
      V1 = V1 ∪ {x} ;
    Finsi
  Fin pour
  V2 = V_G \ V1 ;
Finsi
```

Théorème 3.2

L'algorithme **GrapheBiparti** détermine si un graphe simple d'ordre n fini et de taille m, est biparti ou non. Si c'est le cas, il donne une 2-partition avec une complexité en **O(m)**.

Démonstration: Il suffit d'utiliser le fait que $m = O(n^2)$. □

Reconnaissance des graphes parfaits

Comme on l'a dit dans le chapitre 2, le problème de la reconnaissance des graphes parfaits est resté ouvert jusqu'en 2002. Puis, Maria Chudnovsky, Paul Seymour, Gérard Cornuéjols, Xinming Liu et Kristina Vušković ont simultanément proposé deux algorithmes de reconnaissance des graphes de Berge [1]en temps polynomial [31],[8], [15]. Ce problème semble après coup moins difficile que la preuve de la conjecture forte des graphes parfaits.

On appelle **pyramide** tout graphe F dont l'ensemble des sommets est l'union de trois ensembles induisant des chemins P_1, P_2, P_3 tels que pour $i = 1, 2, 3$, P_i est de longueur

au moins 1, d'extrémités a et b_i, et tels que $\{b_1, b_2, b_3\}$ induit un triangle, tels que a est l'unique sommet commun aux trois chemins, et tels qu'il n'y ait aucune arête entre des sommets de P_i et P_j $(1 \leq i < j \leq 3)$ autre que celles du triangle et celles d'extrémité a. De plus, un seul chemin parmi P_1, P_2, P_3 est autorisé à être de longueur 1, les deux autres doivent être de longueur au moins 2. Si F est sous-graphe induit d'un graphe G, alors on dit que les trois chemins P_1, P_2, et P_3 forment une pyramide de G. On dit que le sommet a est le **coin** de la pyramide.

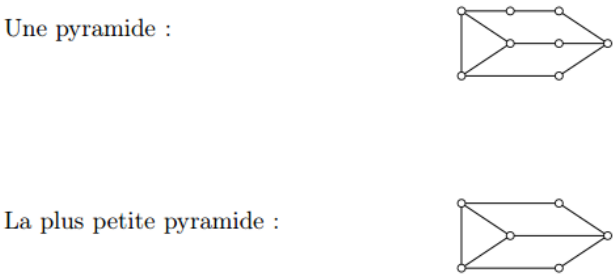


FIGURE 3.5 – Des pyramides

Il faut noter que les deux algorithmes reconnaissent les graphes de Berge, et ne reconnaissent donc les graphes parfaits que dans la mesure où le théorème fort des graphes parfaits est vrai. Il faut également noter que le problème de la reconnaissance des graphes sans trou impair est encore ouvert.
Pour fournir un aperçu des méthodes employées, il nous faut deux définitions.

Soient G un graphe et C un trou de G. On dit qu'un sommet v de G est **majeur** sur C si l'ensemble de ses voisins sur C n'est inclut dans aucun P_3 de C. On appelle **raquette** pour C un sommet majeur sur C qui voit exactement trois sommets de C.

Considérons le graphe suivant :

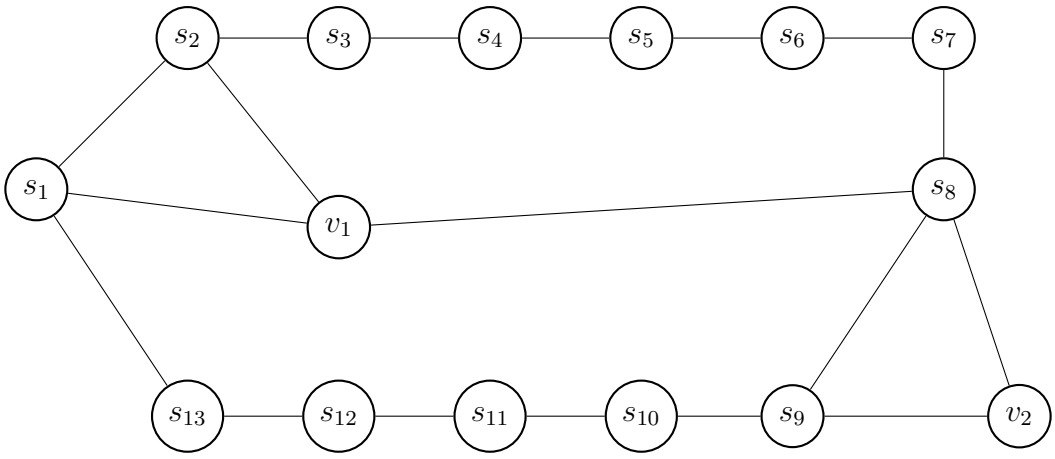


FIGURE 3.6 – Un exemple de de sommet majeur et de raquette

Dans ce graphe, le cercle $C = [s_1, s_2, ..., s_{13}, s_1]$ est un trou impair, v_1 est un sommet majeur sur C. En effet, l'ensemble des voisins de v_1 dans C est $\{s_1, s_2, s_8\}$ n'est inclut dans aucun P_3 de C, par contre v_2 n'est pas majeur sur C car l'ensemble de ses voisins $\{s_8, s_9\}$ est inclut dans $P = [s_7, s_8, s_9]$, ce chemin est un P_3 de C.

1. Les graphes ne contenant pas de trous ou antitrous impairs comme sous-graphes induits seront dorénavant appelés graphes de Berg

On remarquera que si v est une raquette pour un trou impair C de G de taille minimale, alors l'ensemble $V(C) \cup \{v\}$ induit une pyramide.

En effet dans l'exemple précédent $V(C) \cup \{v_1\}$ est un pyramide, la figure 3.7 illustre ce pyramide :

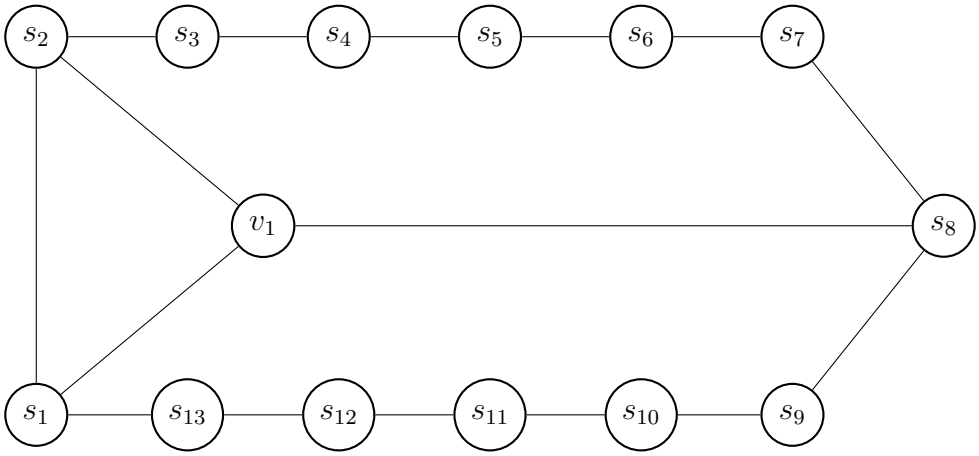


FIGURE 3.7 – Le pyramide induit par $V(C) \cup \{v_1\}$

Les deux algorithmes reposent sur une première phase commune dite de nettoyage du graphe [31]. Cette première phase permet soit de conclure en temps polynomial que le graphe n'est pas de Berge, soit de fournir une liste d'au plus $O(n^8)$ sous-ensembles de V_G avec la propriété suivante : si C est un trou impair de taille minimale de G, et si C est sans raquette, alors l'un des ensembles est disjoint de C et contient tous les sommets majeurs de C. On dit alors que le trou C est nettoyé (de ses voisins majeurs), et on dit que le graphe G est nettoyé. La phase de nettoyage permet donc de ramener la reconnaissance des graphes de Berge à deux problèmes plus simples : la détection des trous impairs possédant une raquette, puis la détection des trous impairs nettoyés de taille minimale. Ces problèmes ont été résolus de deux manières différentes.
D'une part, Chudnovsky et Seymour ont proposé un algorithme direct de recherche de pyramides dans un graphe quelconque, puis de trou impair dans un graphe nettoyé [8]. D'autre part, Cornuéjols, Liu et Vušković [15] ont proposé un algorithme de reconnaissance fondé sur leur théorème de décomposition des graphes sans trou impair [32]. Nous mentionnons pour mémoire l'algorithme suivant qui utilise des techniques de plus courts chemins.

Algorithme 4 Chudnovsky et Seymour [8]

ENTRÉES: Un graphe simple G ;
SORTIES: Si G est sans pyramide, l'algorithme retourne "Pas de pyramide". Sinon, il retourne une pyramide de G de taille minimale ;
COMPLEXITÉ : $O(n^9)$.

Enfin nous citons le principal résultat sur la reconnaissance des graphes de Berge :

Algorithme 5 ([[31],[8]] Chudnovsky, Seymour, Cornuéjols, Liu, Vušković)

ENTRÉES: Un graphe simple G ;
SORTIES: Si G est de Berge, l'algorithme retourne Berge. Sinon, il retourne Non Berge.
COMPLEXITÉ : $O(n^9)$.

Graphes triangulés

Un graphe est triangulé si tout cycle de longueur strictement supérieure à 3 possède une corde [18]. Les graphes triangulés sont caractérise par l'existence d'un schéma d'élimination simplicial [14]. On dit qu'un sommet est simplicial si son voisinage induit un sous-graphe complet (i.e. une clique). Les graphes triangulés peuvent donc être construit (ou réduit) en ajoutant (ou supprimant) un à un des sommets simpliciaux. Lex-BFS (algorithme de parcours en largeur lexicographique [11]) est le premier algorithme linéaire de reconnaissance des graphes triangulés. Il procède en deux étapes : construction d'un ordre d'élimination simplicial si et seulement si le graphe est triangulé puis vérification. **LexBFS** parcourt les sommets d'un graphe et les numérote un par un de n à 1. Lors de l'étape générale, chaque sommet non numéroté a une étiquette, correspondant à l'ensemble des numéros de ses voisins déjà numérotés. Un ordre lexicographique est défini sur les étiquettes : l'étiquette $L(a)$ est strictement plus grande que l'étiquette $L(b)$ s'il existe un entier $i \in L(a) \setminus L(b)$ tel que $\forall j > i$, $j \in L(a) \cap L(b)$ ou $j \notin L(a) \cup L(b)$. Le prochain sommet qui est numéroté est le sommet dont l'étiquette est maximale selon l'ordre lexicographique. Voici une description formelle de l'algorithme.

Algorithme 6 Lex-BFS [11]

Entrées: Un graphe simple $G = (V_G, E_G)$ et un sommet s de G.
Sorties: Un ordre d'élimination simplicial λ si et seulement si G est triangulé.

Pour tout sommet $x \in V_G$ **faire**
  étiquette$(x) = \varnothing$.
Fin pour
étiquette$(s) = n$.
Pour $i = n - 1$ jusqu'à 1 **faire**
  Choisir un sommet non numéroté $x \in V_G$ d'étiquette maximum ;
  $\lambda(i) \leftarrow x$
  Pour tout pour chaque voisin non numéroté y de x **faire**
    Ajouter i à étiquette(y) ;
  Fin pour
Fin pour

Théorème 3.3 (Caractérisation LexBFS [11])

Un graphe simple G est triangulé ssi tout ordre LexBFS de G est simplicial.

Proposition 3.1 (Complexité[18])

Soit G un graphe triangulé d'ordre n et de taille m, alors

On sait construire un ordre avec **LexBFS** en $O(m + n)$. La vérification que cet ordre est simplicial peut se faire avec la même complexité.
La complexité finale est donc $O(m + n)$.

Sur le graphe triangulé de la figure 3.8 on peut vérifier que l'ordre construit par **LexBFS** est bien un schéma d'élimination parfait.

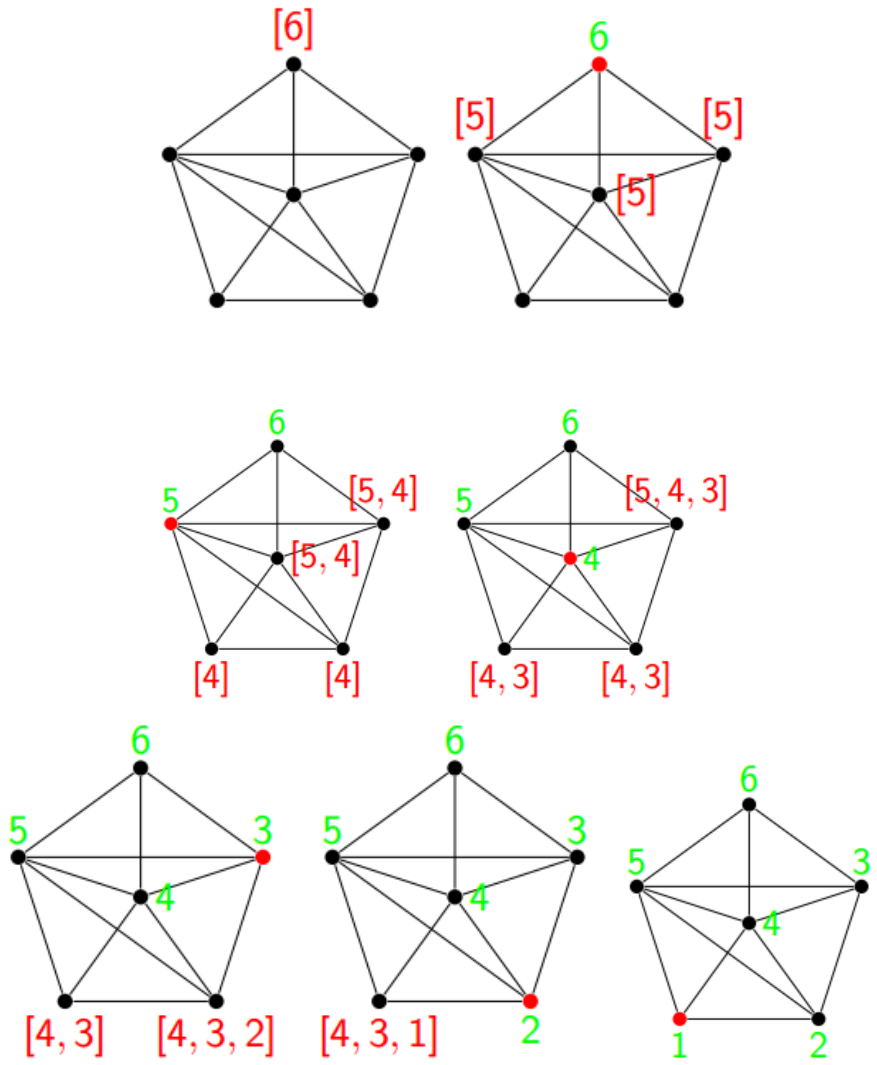


FIGURE 3.8 – Construction d'un schéma d'élimination parfait par parcours en largeur Lexicographique(**LexBFS**).

Reconnaissance des graphes d'intervalles

De nombreuses classes de graphes admettent des caractérisations par un ordre sur les sommets du graphe. L'algorithme de reconnaissance est alors de la forme :
- calculer un ordre,
- vérifier l'ordre trouvé.

Les deux étapes peuvent avoir une complexité différente.
On a déjà vu dans le théorème 3.3 que pour reconnaître un graphe triangulé on peut chercher un ordre d'élimination simpliciale de ses sommets.

Déterminer si un graphe $G = (V_G, E_G)$ donné est un graphe d'intervalle peut être fait en complexité temporelle $O(n + m)$ en recherchant un ordonnancement des cliques maximales de G qui est consécutif en respectant les inclusions des nœuds. De manière formelle, un graphe G est un graphe d'intervalle si et seulement si les cliques maximales $M_1, M_2, \ldots, M_k$ de G peuvent être ordonnées telles que pour tout $v \in M_i \cap M_k$, alors $v \in M_j$ pour tout entier j, $i \leq j \leq k$.

L'algorithme original permettant de savoir si un graphe est un graphe d'intervalles en

temps linéaire, dû à Booth et Lueker[4] est basé sur un arbre PQ complexe, mais Habib et al[19] ont montré comment résoudre plus simplement le problème, en utilisant le fait qu'un graphe est un graphe d'intervalles si et seulement si il est cordal et son graphe complémentaire est un graphe de comparabilité.

Un historique non exhaustif des algorithmes de reconnaissance des graphes d'intervalles est présenté en tableau 4.1.

Auteurs, références	Complexité	Commentaires
Lekkerkerker, Boland[27]	$O(n^4)$	sommets simpliciaux et triplets astéroïdes
Fulkerson, Gross [13]	$O(n^4)$	1-consécutifs pour matrice des cliques max
Booth, Lueker [5, 6]	$O(n+m)$	utilisation de PQ-trees
Korte, Mohring [24, 25]	$O(n+m)$	utilisation de MPQ-trees et LexBFS
Ramalingam, Pandu Rangan [36]	$O(n^2)$	algo séquentiel et parallèle
Hsu, Ma [22, 23]	$O(n+m)$	utilisation de décomposition modulaire et LexBFS
Simon [39]	$O(n+m)$	4 LexBFS successifs, contrexemple trouvé par Ma
Hsu [21]	$O(n+m)$	variante de [22] pour la déc. modulaire
Corneil, Olariu, Stewart [10]	$O(n+m)$	4 LexBFS successifs
Habib, Paul, McConnell, Viennot [34]	$O(n+m)$	LexBFS et affinage de partition sur les cliques max.

TABLE 3.1 – Historique de la reconnaissance des graphes d'intervalles.

Notons qu'un ensemble astéroïde d'un graphe G est un stable A tel que pour tous sommet $v \in A$, il existe une composante connexe de $G \setminus N_G(v)$ contenant tous les sommets de $A \setminus \{v\}$. Le nombre astéroïde $an(G)$ du graphe G est la taille maximum d'un ensemble astéroïde de G. En particulier, un graphe est dit sans triplet astéroïde si son nombre astéroïde est au plus deux.

3.4 Algorithmes de coloration

Dans cette partie quelques algorithmes de coloration sont présentés, ils sont utilisés pour colorier les graphes simples. Chaque algorithme est illustré par un exemple d'application dans un graphe simple.

3.4.1 L'algorithme COLOR

L'algorithme de coloration COLOR (ou bien l'algorithme de coloration séquentielle) est l'algorithme le plus naturel permettant de colorier les sommets d'un graphe. Étant donné un ordre sur les sommets du graphe, cet algorithme parcourt les sommets selon cet ordre et donne à chaque sommet la plus petite couleur non attribuée à ses voisins.

Voici une description formelle de l'algorithme.

Algorithme 7 Algorithme COLOR

Entrées: Un graphe simple G d'ordre n et un ordre λ sur ses sommets.
Sorties: Une coloration des sommets de G.
Pour $i = 1$ jusqu'à n **faire**
Choisir le sommet non colorié x qui est minimum pour λ.
Colorier x avec la plus petite couleur qui n'est pas présente dans son voisinage.
Fin pour
Complexité : $O(n + m)$.

Il est facile de voir que pour tout graphe, il existe un ordre tel que l'algorithme COLOR appliqué à celui-ci donne une coloration optimale. Un tel ordre peut être obtenu à partir d'une coloration optimale en ordonnant les sommets par ordre croissant de leur couleur. La figure 3.9 donne un exemple d'ordre des sommet de P_4 tel que la coloration obtenue par l'algorithme COLOR n'est pas optimale.

Ordre $\lambda : a - d - b - c$

Figure 3.9 – Exemple d'exécution de l'algorithme COLOR pour P_4

Soient G un graphe et λ un ordre sur les sommets de G. L'ordre λ est dit **parfait** si, pour tout sous-graphe induit H de G, la coloration obtenue par l'algorithme COLOR sur le graphe H et l'ordre λ restreint à H est optimale.
Un graphe G est dit **parfaitement ordonnable** s'il possède un ordre parfait.
Il a été démontré (par Chvátal [9]) que si un graphe ne contient aucun quadruplet de sommets a, b, c, d ayant la structure d'un P_4 (c'est-à-dire que a est relié à b mais pas à c et d, b est relié aussi à c mais pas à d et c est également relié à d), alors l'algorithme COLOR produira une coloration optimale en $\chi(G)$ couleurs, quel que soit l'ordre utilisé.

3.4.2 Algorithme Welsh-Powell

il s'agit d'utiliser l'algorithme COLOR mais cette fois-ci avec un ordre judicieux, afin d'obtenir une coloration propre la plus "acceptable" possible. L'algorithme de Welsh-Powell consiste ainsi à colorer 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-Powell :

1. On classe d'abord les sommets du graphe dans l'ordre décroissant de leur degré. On obtient ainsi une liste $x_1, x_2, \ldots x_n$ de sommets telle que $d_G(x_1) \geq d_G(x_2) \geq \ldots \geq d_G(x_n)$.
2. On choisit une couleur c_1 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.

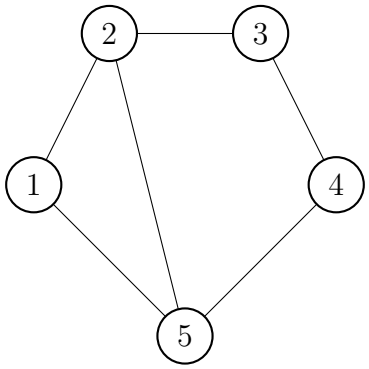
Algorithme 8 Algorithme Welsh-Powell

Entrées: Un graphe simple G d'ordre n.
Sorties: Une coloration des sommets de G.
λ : ordre décroissant du degré des sommets.
exécuter l'algorithme Color (G,n,λ).

La complexité de l'algorithme devient $O(nlogn + m)$ en utilisant un tri par comparaison, mais reste $O(n + m)$ avec un tri par dénombrement. Cependant on peut parfois aboutir aux pires colorations possibles.

Exemple

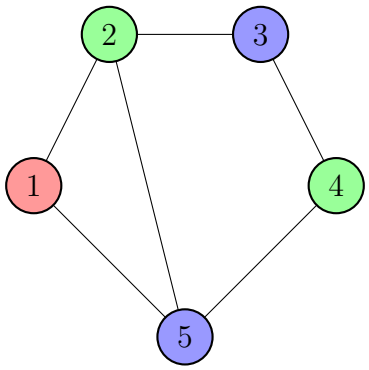
Prenons le graphe suivant



Sommet	Degré	Successeurs
1	2	$\{2,5\}$
2	3	$\{1,3,5\}$
3	2	$\{2,4\}$
4	2	$\{3,5\}$
5	3	$\{1,2,4\}$

Couleur/Sommet	5	2	4	3	1
$c1$: bleu	$c1$			$c1$	
c2 : vert	X	$c2$	$c2$	X	
c3 : rouge	X	X	X	X	$c3$

On obtient alors la coloration suiavant :



couleur blue : $5, 3$
couleur vert : $2, 4$
couleur rouge : 1.
Ordre λ : $5 - 2 - 4 - 3 - 1$

FIGURE 3.10 – Exemple d'exécution de l'algorithme Welsh-Powell

Contre-exemple

Afin de tester l'efficacité de l'algorithme de Welsh-Powell, nous allons mettre en place un contre exemple.

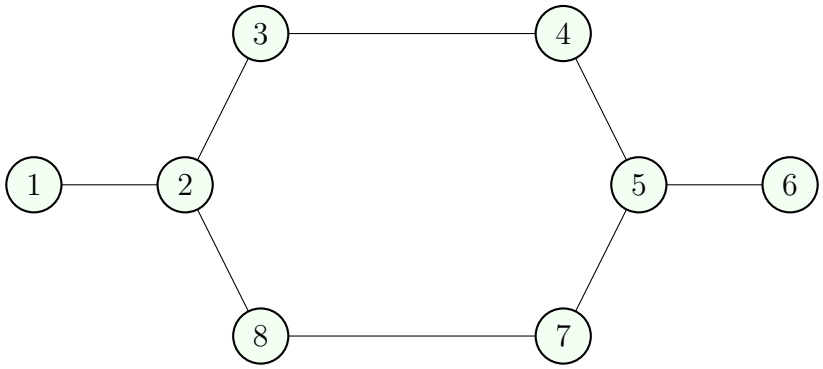


FIGURE 3.11 – Exemple de graphe qui trompe l'algorithme Welsh-Powell

Nous allons dérouler l'algorithme de Welsh-Powell normalement :

Sommet	Degré	Successeurs
1	1	2
2	3	$\{1, 3, 8\}$
3	2	$\{2, 4\}$
4	2	$\{3, 5\}$
5	3	$\{4, 6, 7\}$
6	1	5
7	2	{5,8}
8	2	$\{2, 7\}$

Couleur/sommet	5	2	8	7	4	3	6	1
$c1$ (rouge)	$c1$	$c1$						
$c2$ (vert)	x	x	$c2$		$c2$		$c2$	$c2$
$c3$ (blue)	x	x	x	$c3$	x	$c3$	x	x

Avec l'application de l'algorithme, on obtient alors cette coloration :

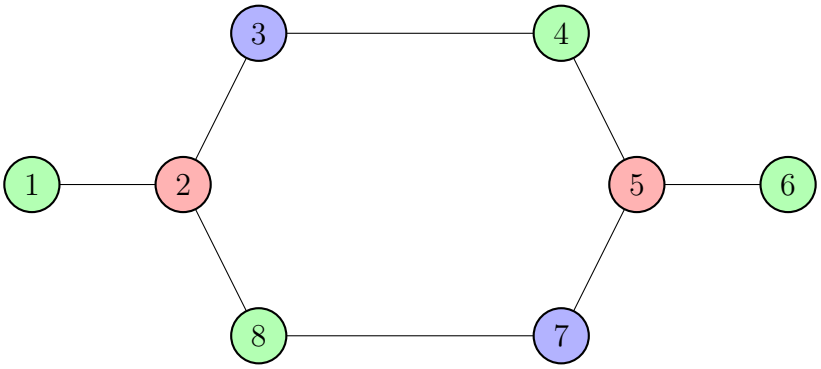
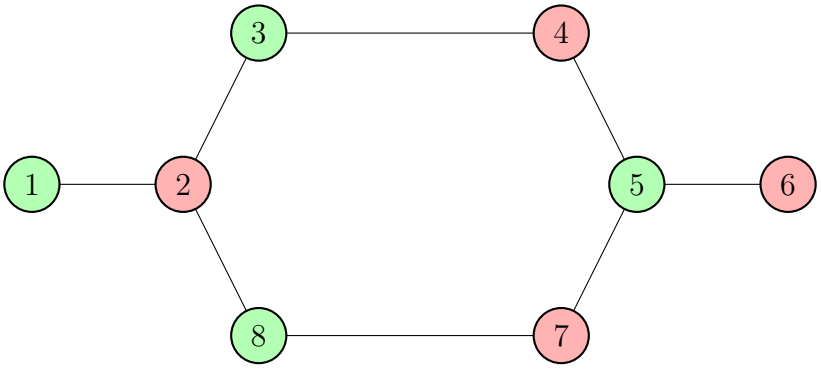


FIGURE 3.12 – 3-coloration obtenu par l'algorithme Welsh-Powell.

Cette coloration n'est pas optimale. En effet, seul deux couleurs suffisant pour colorier ce graphe, tel que :



couleur vert : 1,3,5,8
couleur rouge : 2,4,6,7.

FIGURE 3.13 – 2-coloration de graphe et non optimalité de l'algorithme Welsh-Powell.

L'algorithme de Welsh-Powell permet d'avoir une coloration des sommets cohérente.

Néanmoins, le contre-exemple montre bien que la coloration proposée n'est pas toujours optimale, car dans certains cas, un nombre plus restreint de couleurs peuvent être utilisées (voir ci-dessus).

Il est donc parfois plus simple d'étudier le graphe sans ce dernier et de trouver soi-même une coloration optimale, et ensuite vérifier sa coloration avec l'algorithme pour voir si elle est optimale ou non (à condition que le graphe soit facile de compréhension comme l'est notre contre-exemple).

3.4.3 Algorithme DSATUR

Cette fois, on considère toujours qu'un sommet est plus "difficile" à colorer s'il a beaucoup de voisins, mais on prend également en compte le fait qu'un sommet qui possède déjà beaucoup de voisins de couleurs différentes sera aussi "problématique". On modifie alors l'algorithme de coloration COLOR pour colorer en priorité de tels sommets. Concrètement, on définit dans un graphe simple G, la notion de degré de saturation d'un sommet v de G, notée $DSAT_G(v)$, calculé à un certain moment de l'algorithme comme suit :
- Si v n'a aucun voisin colorié alors $DSAT_G(v) = d_G(v)$.
- Sinon, $DSAT_G(v)$ =nombre de couleurs différentes utilisées pour colorer les voisins de v.

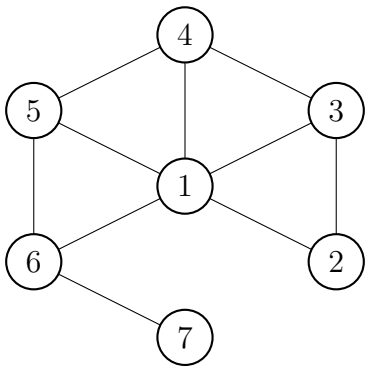
L'algorithme est consiste alors à colorer séquentiellement le graphe en prenant à chaque étape un sommet non coloré de degré de saturation maximale, où on impose de prendre un sommet de degré maximal en cas de conflit, c'est-à-dire dans le cas où plusieurs sommets sont de même degré de saturation. Il ne faudra donc pas oublier de mettre à jour à chaque étape le degré de saturation d'un sommet. On remarque également que lorsqu'on colore un sommet v, il n'est pas nécessaire de mettre à jour $DSAT$ que pour les sommets voisins de v.

Algorithme 9 Algorithme **DSATUR**[35]

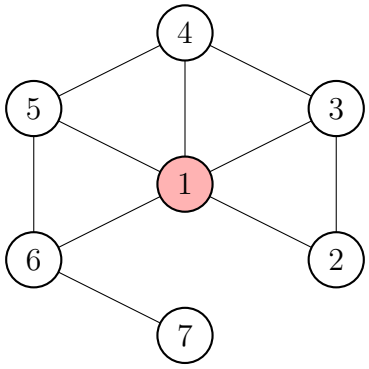
ENTRÉES: Un graphe simple $G = (V_G, E_G)$.
SORTIES: Une coloration propre de G.
Ordonner les sommets par ordre décroissant de degré.
Colorer un sommet de degré maximum avec la couleur 1.
Tantque tous les sommets ne sont pas colorés **faire**
Choisir un sommet non coloré x avec $DSAT(x)$ maximum.
Si il existe un conflit avec un autre sommet **alors**
Choisir celui avec degré maximum (noté x).
Colorer ce sommet par la plus petite couleur possible.
Mettre à jour DSAT pour les voisins de x
Finsi
Fin tantque
COMPLEXITÉ $O(n^2)$.

Exemple

Considérons le graphe suivant :



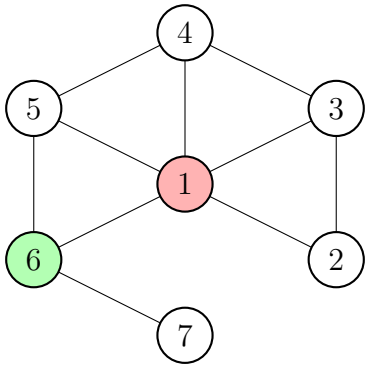
On prend le sommet de plus haut degré, soit le sommet 1. On lui attribue la couleur minimale, soit $c_1(rouge)$:



Étape 1

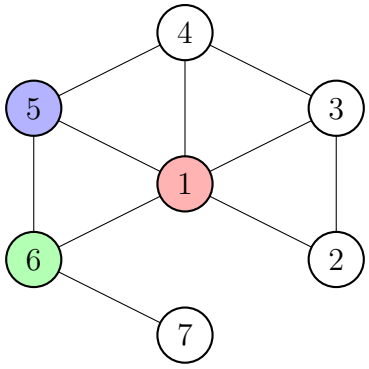
Puis, on regarde plus un voisin du sommet 1 étant de plus haut degré : Nous allons prendre le sommet 6.

Il faut lui attribuer une couleur minimale, mais différente de la couleur du sommet 1 : On prend $c_2(vert)$:



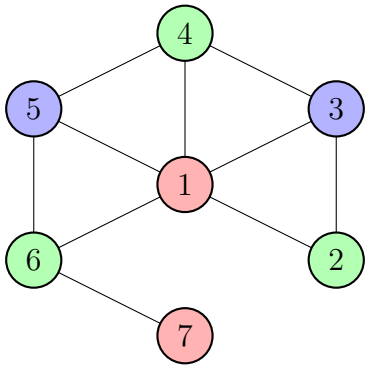
Étape 2

On fait la même chose pour le voisin du sommet 6 étant de plus haut degré : le sommet 5. On lui attribue une couleur minimale différente de celle de ses voisins : la couleur $c_3(blue)$:



Étape 3

On réédite l'opération pour tous les autres sommets, puis on obtient :



Graphe colorié par l'algorithme DSATUR
$rouge = \{1, 7\}$, $vert = \{2, 4, 6\}$, $blue = \{3, 5\}$.

L'algorithme DSATUR étant classé parmi les heuristiques il ne fournit pas forcement une solution optimale. Il produit donc en temps polynomial une solution réalisable. Son auteur a montré qu'il était capable de fournir en un temps court (relativement aux autres heuristiques et aux méthodes exactes) une coloration optimale dans plus de 90% des cas.

Cet algorithme est exact pour les graphes bipartis, les graphes-cycle, les arbres ...

Contre exemple :

Le plus petit graphe pour lequel DSATUR ne fournit pas la solution exacte au problème possède 8 sommets et 12 arêtes.

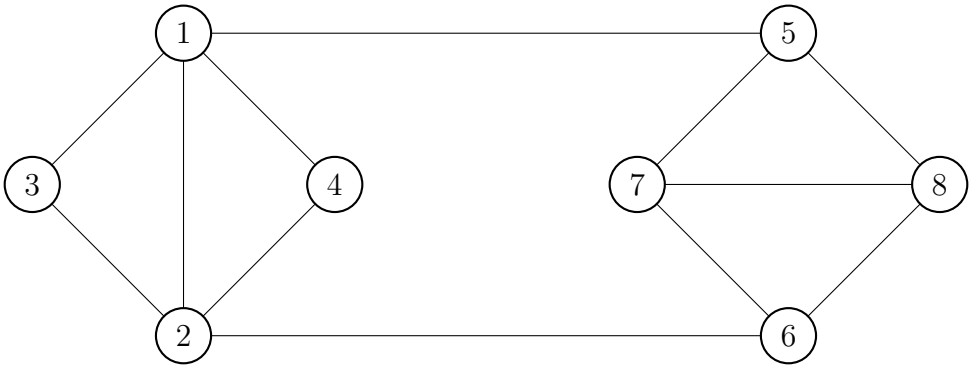
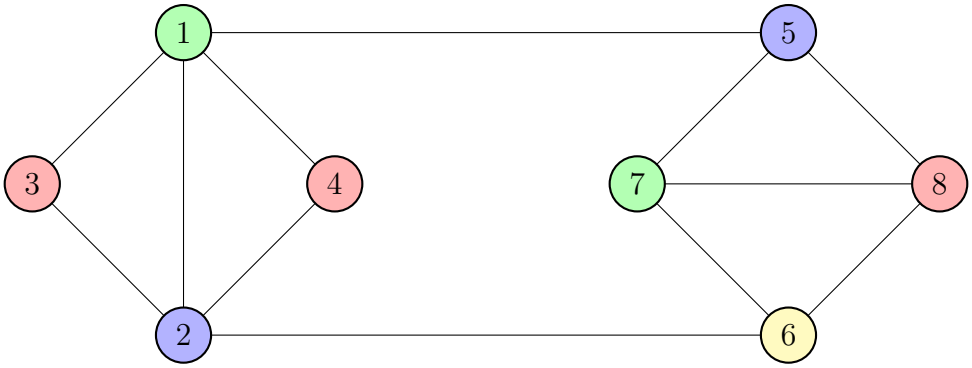


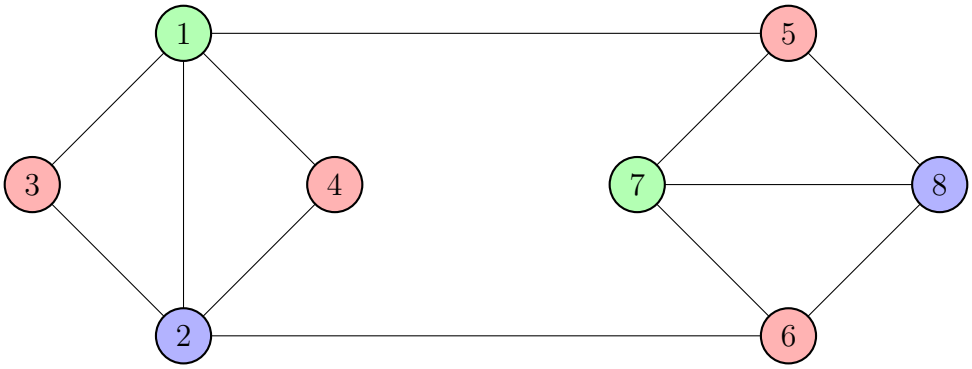
FIGURE 3.14 – Le plus petit graphe 3-colorable qui trompe l'algorithme DSATUR qui trouve une 4-coloration.

En effet,



$$rouge = \{3, 4, 8\},\ vert = \{1, 7\},\ blue = \{2, 5\}\ ,\ jaune = \{6\}.$$

Ainsi ce graphe est 3-coloriable,



$$rouge = \{3, 4, 5, 6\},\ vert = \{1, 7\},\ blue = \{2, 8\}.$$

3.4.4 Algorithme LexBFS-COLOR

Nous avons vu que l'algorithme LexBFS a été defini dans le but de reconnaître les graphes triangulé.

L'algorithme **LexBFS-COLOR** désigne l'algorithme COLOR (séquentielle) appliqué à l'ordre inverse de celui obtenu par l'algorithme LexBFS.

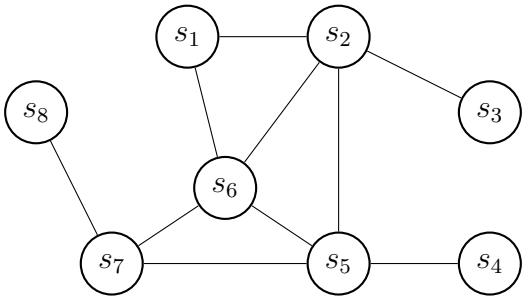
Algorithme 10 Algorithme LexBFS-COLOR

Entrées: Un graphe simple G d'ordre n.
Sorties: Une coloration des sommets de G.
exécuter l'algorithme LexBFS (G,n)
λ : ordre obtenu par LexBFS et λ^{-1} son ordre inverse.
exécuter l'algorithme Color (G,n,λ^{-1}).
Comlixité $O(n + m)$.

Une coloration obtenue par l'algorithme **LexBFS-COLOR** sur la classe des graphes triangulé est optimale.

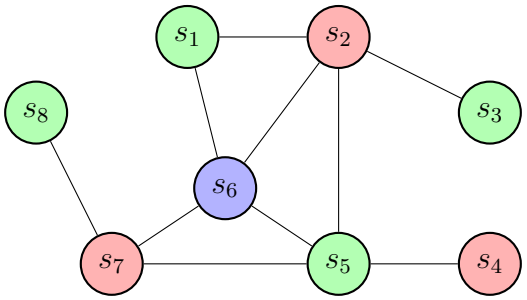
Exemple

considérons le graphe suivant



Etapes/sommets	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
Etape 0	[8]	[∅]	[∅]	[∅]	[∅]	[∅]	[∅]	[∅]
Etape 1	8	[7]	[∅]	[∅]	[∅]	[7]	[∅]	[∅]
Etape 2	8	7	[6]	[∅]	[6]	[7,6]	[∅]	[∅]
Etape 3	8	7	[6]	[∅]	[6,5]	6	[5]	[∅]
Etape 4	8	7	[6]	[4]	5	6	[5,4]	[∅]
Etape 5	8	7	4	[4]	5	6	[5,4,3]	[∅]
Etape 6	8	7	4	[4]	5	6	3	[2]
Etape 7	8	7	4	2	5	6	3	[2]
Etape 7	8	7	4	2	5	6	3	1

L'ordre obtenu par LexBFS est $\lambda = [s_8, s_4, s_7, s_3, s_5, s_6, s_2, s_1]$, et l'ordre inverse est $\lambda^{-1} = [s_1, s_2, s_6, s_5, s_3, s_7, s_4, s_8]$, ainsi le graphe colorié par l'algorithme LexBFS-COLOR est le suivant,



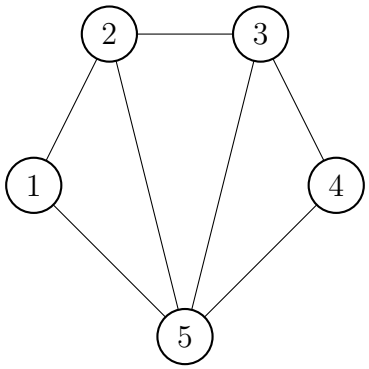
Rouge= $\{s_4, s_7\}$; Vert= $\{s_1, s_3, s_5, s_8\}$; Blue= $\{s_6\}$.

Retour au cas de coloration t-impropre

Les résultats précédents sur la coloration classique des graphes (coloration propre) sont généralisables sur la coloration t-impropre des graphes, et ceci quelque soit le paramètre t.

Exemples

Prenons le graphe de la figure 3.16



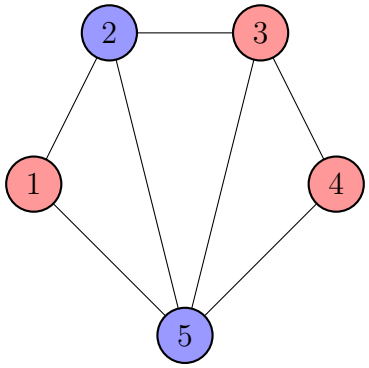
Ordre λ : $5 - 2 - 3 - 4 - 1$

FIGURE 3.15 – Exemple d'exécution de l'algorithme Welsh-Powell (coloration 1-impropre)

Sommet	Degré	Successeurs
1	2	$\{2, 5\}$
2	3	$\{1, 3, 5\}$
3	3	$\{2, 4, 5\}$
4	2	$\{3, 5\}$
5	4	$\{1, 2, 3, 4\}$

Couleur/Sommet	5	2	3	4	1
$c1$: bleu	$c1$	$c1$			
$c2$: rouge	X	X	$c2$	$c2$	$c2$

L'algorithme Welsh-Powell donne 2-coloration 1-impropre.



Ordre $\lambda : 5 - 2 - 3 - 4 - 1$

FIGURE 3.16 – Exemple d'exécution de l'algorithme Welsh-Powell (coloration 1-impropre)

3.5 Conclusion

Dans ce chapitre nous avons présenté quelques algorithmes de reconnaissances et de coloration pour différentes classes de graphes (biparti,parfait,intervalles,triangulé...). Ainsi, nous avons étudier la complexité de chaque algorithme.

CHAPITRE 4

Applications et résultats expérimentaux

4.1 Problème d'Alcatel et coloration impropre

Alcatel a proposé le problème suivant. Un satellite envoie des données à des antennes réceptrices situées sur terre. Chaque antenne écoute une fréquence. En raison de difficultés d'ordre technique, il est impossible de focaliser le faisceau de données exactement sur l'antenne u à laquelle il est destiné. Une partie du signal se propage dans une zone $B(u)$ autour du récepteur. Ceci perturbe la réception des antennes situées dans cette zone $B(u)$ et écoutant la même fréquence. Toutefois, l'intensité des signaux perturbateurs crées par l'envoi de données au récepteur u décroît très rapidement avec l'éloignement de u. Ainsi, si v est un autre récepteur situé dans $B(u)$ et écoutant la fréquence $F(u)$, l'intensité des signaux perturbateurs reçus par v et émanant de l'envoi de données à u est très faible comparée à l'intensité d'un signal destiné à v. Un récepteur est capable de capter correctement le signal qui lui est envoyé tant que l'intensité cumulée de tous les bruits qu'il capte n'excède pas une certaine limite fixée T. Le but est d'assigner une fréquence à chaque récepteur de sorte que tous puissent recevoir proprement leur signal, tout en minimisant le nombre de fréquences utilisées. Nous considérons le cas fondamental où l'intensité I du bruit créé dans $B(u)$ par l'envoi d'un signal à l'antenne u est indépendante de la fréquence du signal, et où un récepteur v est dans la zone de bruit $B(u)$ d'un récepteur u si, et seulement si, u est dans la zone de bruit $B(v)$ du récepteur v. Ainsi, pour qu'un récepteur puisse distinguer son signal des bruits qu'il perçoit, il faut qu'il soit dans la zone de bruit d'au plus $k := \lceil \frac{T}{I} \rceil$ récepteurs écoutant la même fréquence que lui.

Étant donné un réseau d'antennes, le graphe de bruit associé est le graphe dont les sommets correspondent bijectivement aux antennes, avec une arête entre deux sommets si, et seulement si, les antennes correspondantes peuvent interférer. Chaque fréquence est représentée par un entier, appelé couleur. Le but est donc d'assigner à chaque sommet une couleur de sorte que chaque sommet ait au plus k voisins de la même couleur que lui, tout en minimisant le nombre de couleurs utilisées. De telles colorations sont connues dans la littérature sous le nom de *improper colouring*, ou encore *defective colouring*. Le terme **coloration impropre** sera celui utilisé ici. Lorsque k est nul, il s'agit de la notion classique de coloration propre. Utiliser la coloration impropre suppose que tous les récepteurs peuvent écouter sur chacune des fréquences disponibles (aucune restriction n'est imposée quant à la couleur qui peut a priori être attribuée à un sommet, indépendamment de celle de ses voisins).

4.1.1 Modélisation mathématique

La modélisation est une traduction des paramètres du problème dans un langage accessible par la méthode de résolution utilisée, ou bien c'est une façon de décrire le problème sous une forme qui introduit sa réalisation. Enfin, la modélisation d'un problème doit pouvoir donner une interprétation aux solutions concrètes répondant aux besoins du problème réellement posé.

Les données du problème :

n : nombre antennes.
N_F : nombre de fréquences disponibles.
I : l'intensité du bruit.
T : limite de l'intensité cumulée.
$k := \lceil \frac{T}{I} \rceil$.

Contrainte :

Respecter la contrainte d'interférence, i.e. chaque sommet ait au plus k voisins de la même couleur.

Objectifs :

Trouver un bon plan de fréquence qui doit minimiser l'ensemble des interférences.

Soit un graphe $G = (V_G, E_G)$ défini par :
V_G : L'ensemble des sommets du graphe représentent les antennes.
E_G : L'ensemble des arêtes du graphe représentent les risques d'interférences, Il existe une arête $x_i x_j$ de E_G ssi x_i est voisin avec x_j.

Pour résoudre le problème d'affectation de fréquences, on utilise la coloration impropre des sommets qui consiste à affecter à tout sommet du graphe une couleur (fréquence) de façon que chaque sommet ait au plus k voisins de la même couleur (coloration k-impropre) . Le nombre minimum de couleurs nécessaires pour colorier ce graphe en respectant cette contrainte, est appelé le nombre chromatique k-impropre $\chi_k(G)$.

4.2 Allocation de registres

4.2.1 Quelques définitions

Un **compilateur** est, en informatique, le terme utilisé pour désigner un programme qui transforme un code source écrit dans un langage de programmation (le langage source) en un autre langage informatique (appelé langage cible)1. Pour qu'il puisse être exploité par une machine, le compilateur traduit le code source, écrit dans un langage de haut niveau d'abstraction, facilement compréhensible par l'humain, vers un langage de plus bas niveau, un langage d'assemblage ou langage machine. Inversement, un programme qui traduit un langage de bas niveau vers un langage de plus haut niveau est un décompilateur.

Un **processeur** (ou unité centrale de traitement, UCT, en anglais central processing unit, CPU) est un composant présent dans de nombreux dispositifs électroniques qui exécute les instructions machine des programmes informatiques. Avec la mémoire, c'est notamment l'un des composants qui existent depuis les premiers ordinateurs et qui sont présents dans tous les ordinateurs. Un processeur construit en un seul circuit intégré est un **microprocesseur**.

Un **registre** est un emplacement de mémoire interne à un processeur. Les registres se situent au sommet de la hiérarchie mémoire

Dans un compilateur, l'**allocation de registres** est une étape importante de la génération de code. Elle vise à choisir judicieusement dans quel registre du processeur seront

enregistrées les variables durant l'exécution du programme que l'on compile.

Les registres sont des mémoires internes au processeur, généralement capables de contenir un mot machine. Les opérations sur des valeurs rangées dans des registres sont plus rapides que celles sur des valeurs en mémoire vive, quand ce ne sont pas les seules possibles. Mais un processeur compte rarement plus de quelques dizaines de registres, bien trop peu pour toutes les variables d'un programme. Il est donc crucial de choisir avec pertinence les variables qui doivent résider dans les registres à chaque étape de l'exécution. Cela étant difficile dans les premières étapes de la compilation, on commence par faire comme si l'on disposait d'un nombre illimité de registres, puis l'on attribue à ces registres virtuels des registres réels ou des emplacements en mémoire.

4.2.2 Allocation de registres par coloration de graphe

L'une des méthodes classiques d'allocation de registres consiste à ramener le problème de la coloration du graphe d'interférence des variables. Les sommets de ce graphe sont les variables du programme, et deux sommets sont reliés par une arête si les variables correspondantes interfèrent.

On dit que deux variables interfèrent lorsque l'une est définie pendant que l'autre est active (c'est-à-dire susceptible d'être utilisée dans la suite de l'exécution). Deux variables qui interfèrent ne peuvent pas être placées dans le même registre à une nuance technique près : sur certains processeurs, les registres ne sont pas équivalents. On étend donc aux registres la relation d'interférence, en convenant qu'ils interfèrent tous entre eux, et qu'une variable interfère avec les registres qui lui sont interdits. Ainsi, on ne peut colorer avec le même registre deux variables qui sont voisines dans le graphe d'interférence. Attribuer k registres aux variables équivaut à colorer avec k couleurs le graphe d'interférences.

Modélisation mathématique

Les données du problème :

n : nombre de variables du programme.

N : nombre de registres disponibles $(R_1, R_2, ..., R_N)$.

Contrainte :

Respecter la contrainte d'interférence, i.e. deux variables qui interfèrent ne peuvent pas être placées dans le même registre

Objectifs :

Trouver un bon plan d'utilisation de registres qui doit minimiser l'ensemble des interférences entre les variables de programme.

Soit un graphe $G = (V_G, E_G)$ défini par :

V_G : L'ensemble des sommets du graphe représentent les variables.

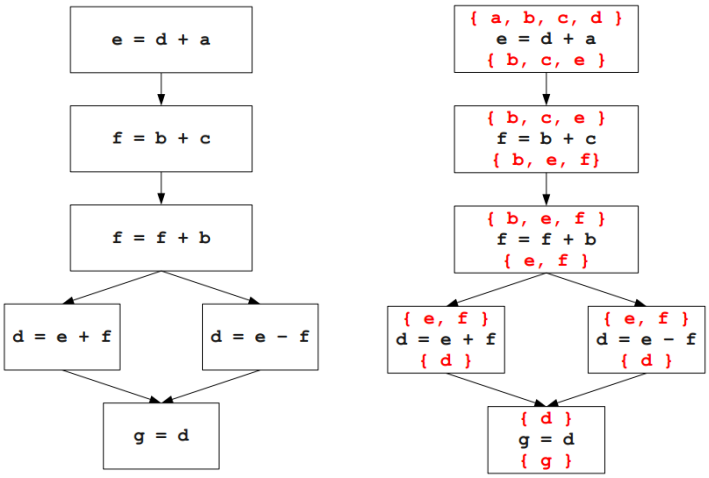
E_G : L'ensemble des arêtes du graphe représentent les risques d'interférences, Il existe une arête $x_i x_j$ de E_G ssi x_i est voisin avec x_j.

Pour résoudre le problème d'allocation des registres, on utilise la coloration propre des sommets qui consiste à affecter à tous les sommets du graphe une couleur (registre) de façon que chaque paire de sommets adjacents soit de couleurs différentes(ne sont pas dans un même registre).

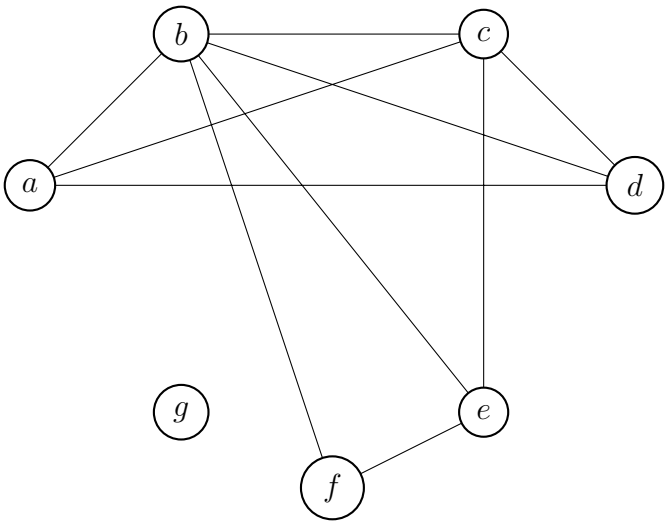
Entrée : a, b, c, d ;
$e = d + a$
$f = b + c$
$f = f + b$
Si (condition sur e)
$d = e + f$
Sinon
$d = e - f$
Finsi
$g = d$
Sortie : g ;

TABLE 4.1 – Exemple de programme.

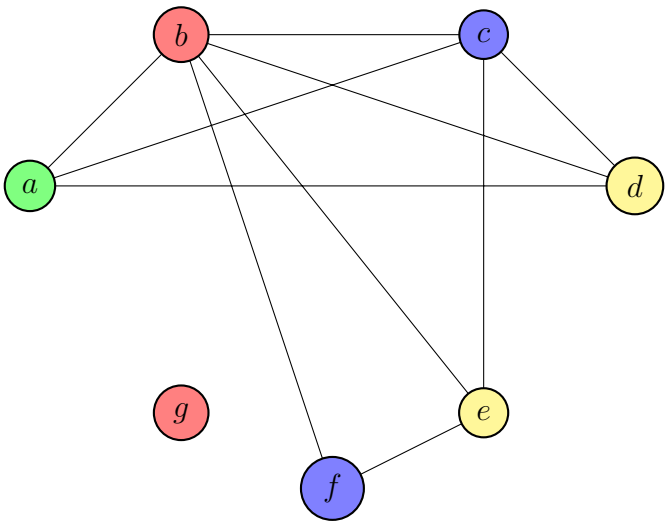
On construit les graphes suivants,



Ainsi le graphe d'interférence



La coloration propre par l'algorithme Welsh-Powell



$rouge = \{b, g\}$, $vert = \{a\}$, $blue = \{c, f\}$, $jaune = \{d, e\}$.

on peut donc utilisé 4 registre par exemple R1(rouge),R2(vert),R3(blue) et R4(jaune), donc le code sera :

Entrée : a, b, c, d ;	**Entrée** : $R2, R1, R3, R4$;
$e = d + a$	$R4 = R4 + R2$
$f = b + c$	$R3 = R1 + R3$
$f = f + b$	$R3 = R3 + R1$
Si (condition sur e)	**Si** (condition sur $R4$)
$d = e + f$	$R4 = R4 + R3$
Sinon	**Sinon**
$d = e - f$	$R4 = R4 - R3$
Finsi	**Finsi**
$g = d$	$R1 = R4$
Sortie : g ;	**Sortie :** $R1$;

4.3 Mini Sudoku

4.3.1 Description du problème

On considère le mini-sudoku de 16 cases suivant :

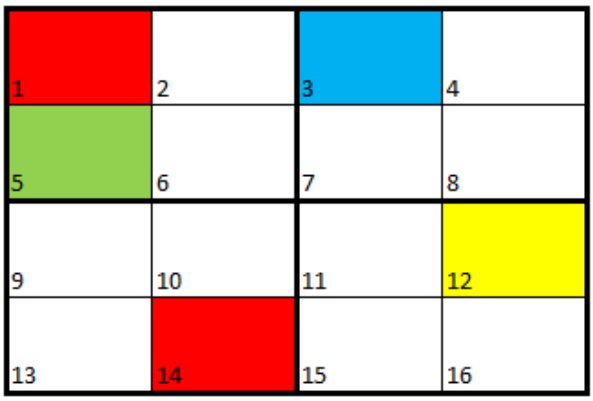


FIGURE 4.1 – grille de sudoku

On cherche à remplir ce sudoku avec les 4 couleurs rouge, bleu, vert, jaune, en considérant les contraintes habituelles :

1. deux cases d'une même ligne ne peuvent pas avoir la même couleur
2. deux cases d'une même colonne ne peuvent pas avoir la même couleur
3. deux cases d'un même carré ne peuvent pas avoir la même couleur.

Par exemple : le sommet 10 est relié aux sommets $2, 6, 9, 11, 12, , 13, 14$.

4.3.2 Résolution

On considère un graphe G à 16 sommets, correspondant à une grille de mini-sudoku, sachant que deux sommets sont reliés dans ce graphe ssi ils ne peuvent pas avoir la même couleur. Par utilisation l'algorithme DSATUR, on construire le graphe colorié par 4 couleurs qui nous donne la solution de la grille mini-sudoko

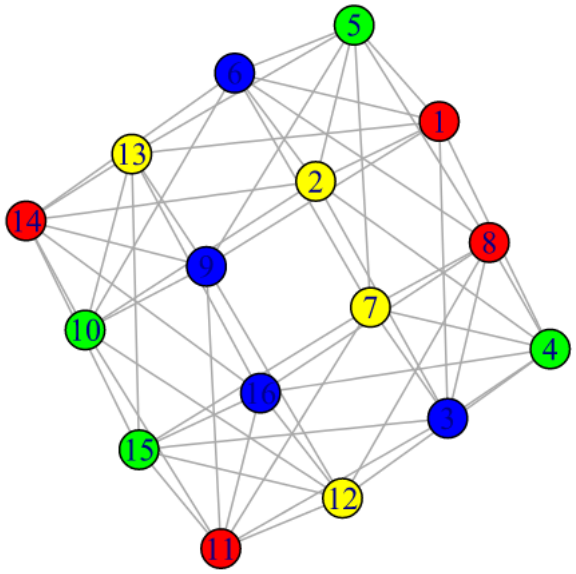


FIGURE 4.2 – Solution de Mini-sudoko

Rouge= $\{1, 8, 11, 14\}$, Vert= $\{4, 5, 10, 15\}$, Blue= $\{3, 6, 9, 16\}$,June= $\{2, 7, 12, 13\}$

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

FIGURE 4.3 – Mini-Sudoko résolu

4.4 Résultats expérimentaux

4.4.1 Sur des graphes pseudo-aléatoires

Définition 4.1

La **densité** d'un graphe est définie par le rapport entre le nombre d'arêtes divisé par le nombre d'arêtes possibles.

Dans le cas d'un graphe simple G d'ordre n et de taille m, la densité est le rapport :

$$D_G = \frac{2m}{n(n-1)}$$

Le numérateur compte le nombre d'arêtes existantes et le multiplie par deux, étant donné que chaque arête est liée à une paire de sommets. Le dénominateur dénombre le total d'arêtes nécessaires pour que chaque sommet soit relié à tous les autres.

Remarque 4.1. *La densité 0 correspond au graphe où tous les sommets sont isolés, et la densité 1 au graphe complet.*

Les trois algorithmes présentés ont été implémentés de plusieurs manières par usage de différents tris pour Welsh-Powell, différentes structures de données pour DSATUR, etc.

Dans un premier temps nous avons donc essayé ces programmes sur des graphes générés de manière pseudo-aléatoire d'ordres variés, et de densité moyenne $\frac{1}{2}$. Les tableaux qui suivent présentent les temps d'exécution obtenus.

Les tests ont été menés sur un ordinateur fonctionnant sous **Windows XP SP3**, équipé d'un **Intel R CoreTM 2 Duo E6400 2.13 GHz et de 2.00 Go de RAM**

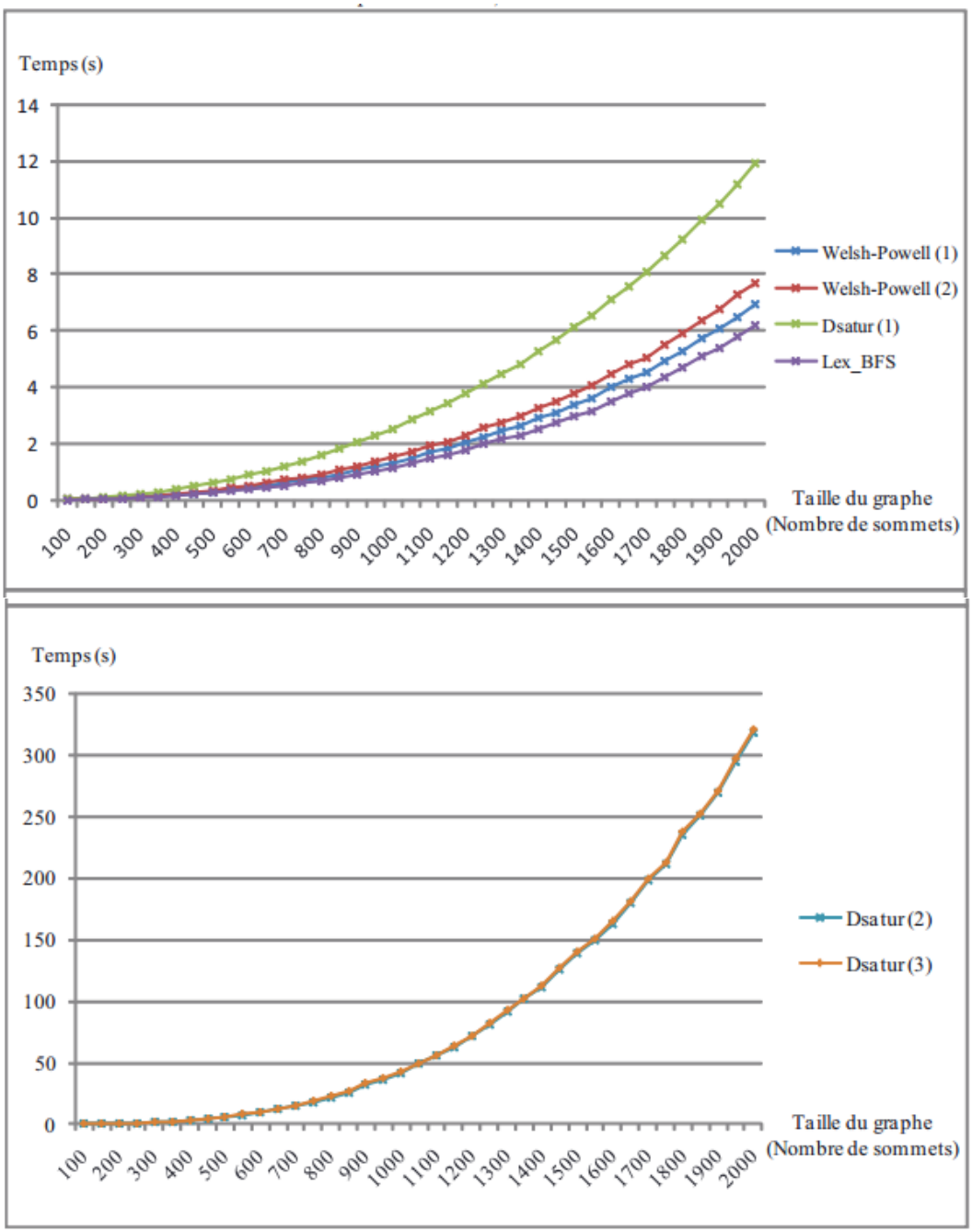


FIGURE 4.4 – Temps d'exécution, de 100 à 2000 sommets.

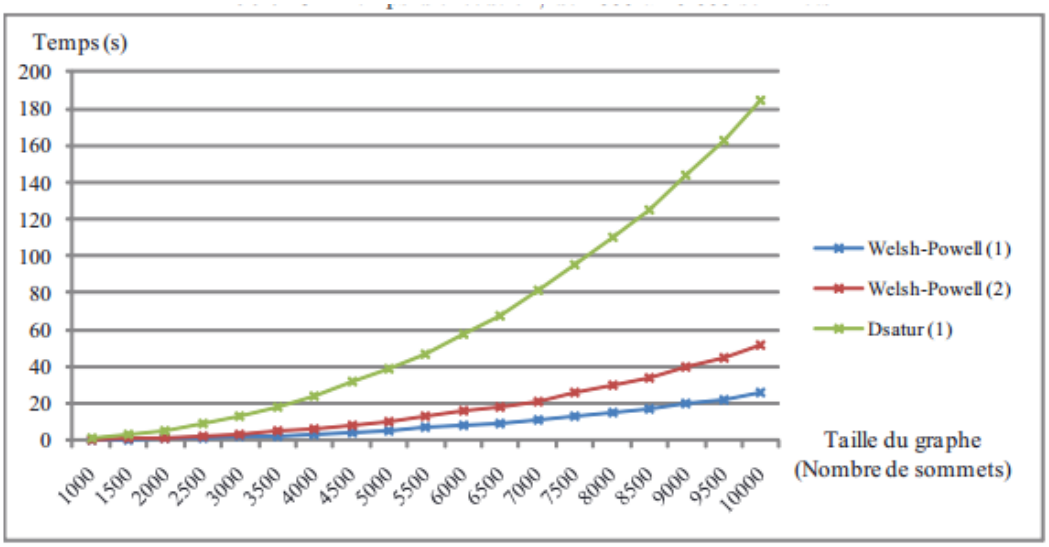


FIGURE 4.5 – Temps d'exécution, de 1000 à 10 000 sommets.

Des problèmes de dépassement de capacité de mémoire n'ont pas permis de mener certains tests jusqu'à 10 000 sommets. Nous avons tenté d'adapter une structure de données pour Dsatur3. Remarquons que le tri par fusion utilisé pour Welsh-Powell1 s'avère plus efficace sur ces graphes que le tri par dénombrement utilisé pour Welsh-Powell2. Globalement nous voyons également que Welsh-Powell et LexBFS sont plus rapides que Dsatur, même si ce dernier offre souvent de meilleurs colorations sur un graphe quelconque.

4.4.2 Efficacité moyenne

Bien que de tels essais ne soient pas nécessairement très significatifs sur des graphes pseudo-aléatoires, on a recueilli également les informations sur les colorations obtenues lors des tests de complexité temporelle. On peut donc voir la différence d'efficacité en terme de nombre de couleurs utilisées par les différents algorithmes sur les deux séries de graphes pseudo-aléatoires. Cela confirme notamment que DSATUR semble plus efficace en général que Welsh-Powell, et qu'une coloration séquentielle selon un ordre lexicographique (Lex-BFS) fourni également un résultat plus mauvais que Welsh-Powell

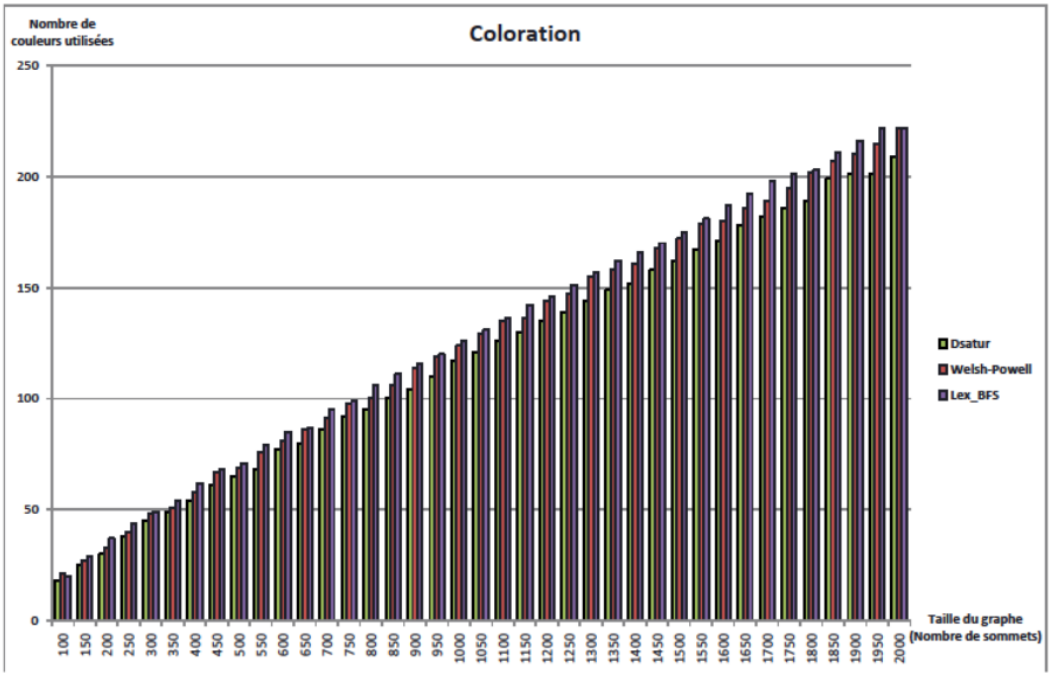


FIGURE 4.6 – Colorations obtenues sur des graphes pseudos-aléatoires, de 100 à 2000 sommets.

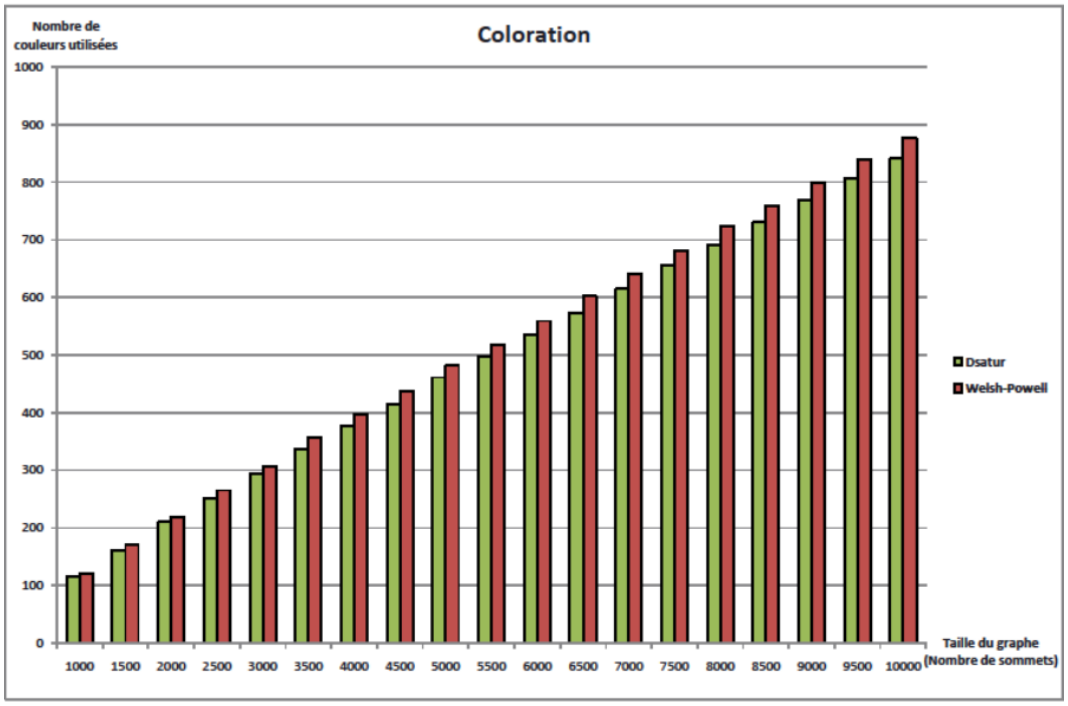


FIGURE 4.7 – Colorations obtenues sur des graphes pseudos-aléatoires, de 1000 à 10000 sommets..

4.4.3 Sur des graphes de DIMACS

Afin de mesurer l'efficacité des algorithmes en terme de nombre de couleurs utilisées, nous avons utilisé des graphes publiés par le centre DIMACS[28] (pour Center for Discrete Mathematics and Theoretical Computer Science). L'intérêt est double : ces graphes

fournissent un benchmark (un banc d'essai permettant de mesurer les performances d'un système pour le comparer à d'autres) efficace pour tester l'efficacité des colorations sur des problèmes concrets, et ils soulignent les applications multiples du problème de coloration dans la vie réelle : allocation de registre lors de la compilation de code, planification d'horaire, un problème d'échec, configuration de réseaux optiques, problème des carrés latins, etc

Graphe	Problème correspondant	Nombre de :			Nb Couleurs		Temps d'exécution (s)				
		Sommets	Arêtes	Coloration	WP	Dsatur	Welsh-Powell		Dsatur		
		n	m	$\chi(G)$			1	2	1	2	3
inithx.i.1.col	Allocation de registre	864	18707	54	54	54	0.047	0.031	0.359	1.297	0.125
mulsol.i.4.col		185	3946	31	31	31	0	0	0.031	0.125	0.016
zeroin.i.1.col		211	4100	49	49	49	0.016	0	0.031	0.187	0.016
school1_nsh.col	Planification	352	14612	???	34	27	0	0.015	0.094	0.547	0.047
jean.col	Livre	80	508	10	10	10	0	0	0	0.015	0
queen11_11.col	Échec	121	3960	11	17	15	0	0	0.015	0.11	0.015
myciel4.col	Graphes de	23	71	5	5	5	0	0	0	0	0.016
myciel7.col	Mycielski	191	2360	8	8	8	0	0	0.016	0.047	0.015
mug100_1.col	-	100	166	4	4	4	0	0	0	0.015	0
wap05a.col	Réseaux	905	43081	???	51	51	0.032	0.046	0.454	2.515	0.156
wap07a.col	optiques	1809	103368	???	52	46	0.093	0.094	1.547	6.562	0.391
qg.order40.col	Carrés latins	1600	62400	40	64	40	0.062	0.047	1.094	3.531	0.25
qg.order100.col		10000	990000	100	128	100	0.797	0.766	39.375	142.219	4.062

TABLE 4.2 – Comparaison de différents algorithmes sur des graphes de la librairie DIMACS

Nous pouvons voir que dans la plupart des cas, WP (pour Welsh-Powell) et DSATUR fournissent la coloration optimale du graphe considéré. Cependant lorsqu'il y a des différences, on voit que DSATUR est toujours plus performant que WP (school1-nsh.col, queen11-11.col et wap07a.col par exemple). Lorsque le graphe présente une structure plus "régulière" (des symétries par exemple pour qg.order40.col), DSATUR fournit bien une meilleure coloration que WP. Au niveau des temps d'exécution, les graphes présentés étant de faibles tailles (généralement moins de 100 000 arêtes, et presque un million pour qg.order100.col), les temps obtenus sont assez peu élevés et ne présentent pas de grandes différences. Toutefois celles-ci deviennent très marquées pour le dernier exemple ; on peut même s'étonner de voir Dsatur3, se distinguer : sans doute les symétries que présente qg.order100.col sont propices à sont exécution.

Conclusion générale

La puissance de la théorie des graphes est de permettre de modéliser étonnement simplement un nombre impressionnant de problèmes auxquels nous sommes quotidiennement (consciemment ou non) confrontés : problèmes de transport, de routage dans les réseaux, d'emploi du temps, de gestion de projets, allocation des fréquences, etc.

Le problème de coloration des graphes est un domaine très actif de la théorie des graphes du fait de ses nombreuses applications.

Dans ce manuscrit, nous avons après avoir rappelé les notions de base de théorie de la complexité et de théorie des graphes, nous nous intéressons a la coloration (propre et impropre) de graphes, nous avons vu que la classe des graphes parfaits était une classe de graphes importante, en particulier parce qu'elle contient de nombreuses autre classes usuelles, dont les graphes bipartis, les graphes d'intervalles, les graphes de comparabilité et les graphes triangulés, le problème de reconnaissance et de coloration peut être facilement résolu dans cette classes de graphes particulières. Nous avons présenté quelques algorithmes de reconnaissance de cette classes de graphes, ainsi les algorithmes de coloration de graphe à savoir COLOR, Welsh-Powell, Dsatur et LexBFS-COLOR.

Nous avons abordé quelques applications de la coloration des graphes dans le monde réel et nous avons appliqué les différents algorithmes à ces applications. Nous conclurons que les différences efficacités constatées, ne dépendent pas seulement des algorithmes considérées, mais aussi de l'implémentation effectuée et que dans la plupart des cas, Welsh-Powell et DSATUR fournissent la coloration optimale du graphe considéré. Cependant lorsqu'il y a des différences, on voit que DSATUR est toujours plus performant que Welsh-Powell, et qu'une coloration séquentielle selon un ordre lexicographique (LexBFS-COLOR) fourni également un résultat plus mauvais que Welsh-Powell.

Perspectives

Après avoir présenté une idée générale sur la coloration de graphes, nous visons à
- étudier des classes d'instances qui admettent des algorithmes exacts efficaces ;
- développer des algorithmes d'approximation de temps polynomial ;
- développer des algorithmes heuristiques ou métaheuristique.

On pourrait tenter a prouver les résultats énoncés sans démonstration, ainsi de généralisé les propriétés et les algorithmes de la coloration propre à la coloration impropre.

Bibliographie

[1] Alain Bretto François Hennecart Alain Faisant. *Éléments de théorie des graphes.* Springer-Verlag France, 2012.

[2] Andrews J. A. and Jacobson M. S. On a generalization of chromatic number. *Congressus Numerantium, vol. 47, pp. 33–48*, 1985.

[3] Appel K. and Haken W. Solution of the four color map problem. *Scientific American, vol. 237, no. 4, pp. 108–121*, 1977.

[4] Booth G. S. K. S. ; Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *dans J. Comput. System Sci., vol. 13,p. 335–379*, 1976.

[5] Booth Kellogg S. and Lueker. George S. Linear algorithms to recognize interval graphs and test for the consecutive ones property. *In Proceedings of the seventh Annual ACM Symposium on Theory of Computing (STOC'75), pages 255-265*, 1975.

[6] Booth Kellogg S. and Lueker. George S. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of Computer and System Science, 13 :335-379*, 1976.

[7] Brooks R. L. On colouring the nodes of a netwok. *Proc. Cambridge Phil. Soc., 37 :194–197*, 1941.

[8] Chudnovsky M. and Seymour. P. Recognizing berge graphs. *Manuscript*, 2002.

[9] Chvàtal V. Perfectly ordered graphs. *Topics on perfect graphs 63-65, North-Holland, Amsterdam-New York*, 1984.

[10] Derek G. Corneil Stephan Olariu and Stewart. Lorna K. The ultimate interval graph recognition algorithm. *In Proceedings of the ninth annual ACM-SIAM Symposium on Discrete algorithms (SODA'98), pages 175-180*, 1998.

[11] Donald J. Rose R. Endre Tarjan and Leuker. George S. Algorithmic aspects of vertex elimination on graphs. *SIAM J. of Comput.*, 1976.

[12] Erdős P. and Hajnal A. On decomposition of graphs. *Acta Mathematica Hungarica, vol. 18, no. 3, pp. 359–377*, 1967.

[13] Fulkerson Delbert R. and Gross. O. A. Incidence matrices, interval graphs, and seriation in archaeology. *Pacific Journal of Mathematics, 28 :565-570*, 1969.

[14] Fulkerson D.R. and Gross. O.A. Incidence matrices and interval graphs. *Paci c Journal Math.(15) 835-855*, 1965.

[15] G. Cornuéjols X. Liu and Vušković. K. A polynomial algorithm for recognizing perfect graphs. 2002.

[16] Gardner Martin. *The Sixth Book of Mathematical Games from Scientific American.* University of Chicago Press, p. 92-94, 1984.

[17] Golumbic Martin. *Algorithmic Graph Theory and Perfect Graphs.* North Holland, 2004.

[18] Golumbic. M.C. Algorithms graph theory and perfect graphs. *Academic Press, New York University*, 1980.

[19] Habib Ross ; Paul Christophe ; Viennot Laurent Michel ; McConnell. Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition, and consecutive ones testing. *Theor. Comput. Sci., vol. 234, p. 59–84*, 2000.

[20] Harary F. Conditional colorability in graphs. *Graphs and Applications, Proc. First Colo. Symp. graph theory (F. Harary and J. Maybee eds), Wiley intersci., Publ. NY*, 1985.

[21] Hsu. Wen-Lian. A simple test for interval graphs. *In Proceedings of the 18th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'92), volume 657 of Lecture Notes in Computer Science, pages 11-16. Springer-Verlag*, 1993.

[22] Hsu Wen-Lian and Ma. Tze-Heng. Substitution decomposition on chordal graphs and applications. *In Proceedings of the second International Symposium on Algorithms (ISA'91), volume 557 of Lecture Notes in Computer Science, pages 52-60. Springer-Verlag*, 1991.

[23] Hsu Wen-Lian and Ma. Tze-Heng. Fast and simple algorithms for recognizing chordal comparability graphs and interval graphs. *SIAM Journal on Computing, 28(3) :1004-1020*, 1999.

[24] Korte Norbert and Möhring. Rolf H. A simple linear-time algorithm to recognize interval graphs. *In Proceedings of the twelfth International Workshop on Graph-Theoretic Concepts in Computer Science (WG'86), volume 246 of Lecture Notes in Computer Science, pages 1-16*, 1986.

[25] Korte Norbert and Möhring. Rolf H. An incremental linear-time algorithm for recognizing interval graphs. *SIAM Journal on Computing, 18(1) :68-81*, 1989.

[26] L. Cowen W. Goddard and Jesurum. C. E. Defective coloring revisited. *J. Graph Theory, 24(3) :205–219*, 1997.

[27] Lekkerkerker Cornelis G. and Boland. J. Ch. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae, 51 :45-64*, 1962.

[28] les graphes de benchmark : DIMACS. `http://mat.gsia.cmu.edu/COLOR02/`.

[29] Lovász László. A characterization of perfect graphs. *Journal of Combinatorial Theory 13 (2)*, 1972.

[30] Lovász László. Three short proofs in graph theory. *J. Combin. Th., Series B, vol. 19, p. 269-271*, 1975.

[31] M. Chudnovsky X. Liu P. Seymour G. Cornuéjols and Vušković. K. Cleaning for bergeness. *Manuscript*, 2003.

[32] M. Conforti G. Cornuéjols and Vušković. K. Decomposition of berge graphs by double star cutsets and 2-joins. *Manuscript*, March 2001.

[33] Maria Chudnovsky Paul Seymour et Robin Thomas Neil Robertson. The strong perfect graph theorem. *Annals of Mathematics 164*, 2002.

[34] Michel Habib Ross M. McConnell Christophe Paul and Viennot. Laurent. Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science, 234 :59-84,*, 2000.

[35] page web(Dsatur) Philippe Rolland. `http://prolland.free.fr/Cours/Cycle2/Maitrise/GraphsTheory/TP/PrgGraphDsat/dsat.html`.

[36] Ramalingam Ganesan and Rangan. Chandrasekaran Pandu. New sequential and parallel algorithms for interval graph recognition. *Information Processing Letters, 34(4) :215-219*, 1990.

[37] Sakarovitch M. Graphes et programmation linéaire. *Hermann enseignement des sciences, Paris*, 1984.

[38] Seymour Paul. How the proof of the strong perfect graph conjecture was found. *Gazette des mathématiciens 106*, 2006.

[39] Simon. Klaus. A new simple linear algorithm to recognize interval graphs. *In Proceedings of the International Workshop on Computational Geometry (CG'91), volume 553 of Lecture Notes in Computer Science, pages 289-308*, 1992.

[40] T. Nishizeki Md.S.Rahman. planar graph drawing. *World Scientific Publishing*, 2004.