Table des matières

République Algérienne Démocratique et Populaire .
Ministère de l' Enseignement Supérieur et de la Recherche Scientifique .
Université Mouloud Mammeri de Tizi-Ouzou

Faculté des Sciences
Département des Mathématiques

Mémoire de Master

Spécialité : MATHÉMATIQUE

Option : Recherche Opérationnelle

Intitulé du mémoire

Étude synthèse et génération des cliques maximales dans les graphes non orientés

---

Réalisé par :

BELHOCINE Lounis
ACHOUR Zehour

Dirigé par :

M. BELHADJ Abdelaziz

Devant le jury d'examen composé de :

Mme. KHEFFACHE Rezika **Maître assistant A** **U.M.M.T.O** **Présidente**

Mme. AKLOUCHE Fariza **Maître assistant A** **U.M.M.T.O** **Examinatrice**

Année Universitaire : 2016/2017

Remerciements

Avant d'entamer cette présentation, nous tenons à exprimer notre sincère gratitude envers tous ceux qui nous ont aidés à faire en sorte que ce projet arrive à terme.

Tout d'abord, nous tenons à remercier **M. Belhadj Abdelaziz**, notre encadreur, pour son aide, sa générosité et le temps qu'il nous a consacré.

Nous sommes reconnaissants également à tous nos enseignants pour leur disponibilité, leur soutien et leur précieuse contribution à notre formation.

Un grand merci à tous ceux qui ont contribué de prés ou de loin à l'aboutissement de ce travail.

Dédicace

Nous dédions ce modeste travail :
A nos familles,
A nos amis,
Et nos professeurs.

Table des figures

Liste des symboles

$\varnothing$	Ensemble vide.
$\|A\|$	Cardinalité de l'ensemble A.
$\{x/xtelque...\}$	Ensemble des x tels que...
$x \in A$	x appartient à l'ensemble A.
$x \notin A$	x n'appartient pas à l'ensemble A.
$A \cup B$	Réunion de A et B.
$A \cap B$	L'intersection de A et B.
$A \subset B$	L'ensemble A est une partie de l'ensemble B.
$(1) \Longrightarrow (2)$	(1) implique (2).
$(1) \Longleftrightarrow (2)$	(1) est équivalent à (2).
$\Gamma(x)$	Ensemble des voisins du sommet x.
$Log(p)$	Logarithme népérien de p.

Introduction générale

Au XIXème siècle, avec l'avènement de la révolution industrielle, l'humanité avait grand besoin d'un puissant outil de modélisation et de résolution de problèmes concrets. C'est alors qu'en 1822, le mot "graphe" a été introduit par l'anglais J. J. Sylvester. Un graphe est un schéma constitué par un ensemble de points et par un ensemble de flèches reliant chacune deux de ceux ci. Les points sont appelés les sommets du graphe, et les flèches les arcs du graphe.

D'une 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, ...

Ainsi, la théorie des graphes est l'un des outils utilisés pour résoudre les problèmes combinatoires allant de l'analyse mathématique jusqu'à la recherche opérationnelle, et comme on ne peut pas résoudre un problème de l'optimisation combinatoire sans passer par l'utilisation d'un algorithme , on peut dire qu'il ya une grande relation entre celle-ci et l'informatique.

L'idée du problème clique maximale est de trouver le plus grand groupe de sommets dans un graphique qui sont tous reliés les uns aux autres. La génération des cliques maximales est fortement utilisée dans divers domaines comme la biotechnologie, l'intelligence artificielle, le data-mining, et même dans l'analyse d'échantillons de groupes d'individus dans le domaine des sciences sociales.

Le problème de génération des cliques maximales exige l'étroite collaboration des deux sciences : la théorie des graphes et l'algorithmique. En effet, pour traiter ce problème, il faut utiliser des algorithmes de génération qui procéderons à la recherche des cliques maximales existantes dans un graphe donné. Il existe plusieurs algorithmes différents qui ont la capacité de parcourir un graphe à la recherche des cliques. Ils sont classés suivant trois critères :
1. L'ordre lexicographique (le respect de cet ordre lors de la génération des cliques) ;
2. Le délai polynomial (le temps pris par l'algorithme pour générer deux choix successifs est polynomial selon la taille du problème) ;
3. L'espace polynomial (l'espace occupé par l'algorithme est polynomial selon la taille du problème).

Ce mémoire traite un problème de génération des cliques maximales dans les graphes non orientés, et il est composé de trois chapitres organisés comme suit :
Le premier traite quelques notions de base de la théorie des graphes, on y introduira d'importantes définitions (cliques maximales, graphe distance héréditaire, complexité). Ces notions seront nécessaires à la compréhension de la notion de génération des cliques maximales. On présentera ensuite au deuxième chapitre les algorithmes de générations de cliques maximales dans les graphes non orientés, qui est notre objet d'étude dans ce mémoire .
Dans le dernier chapitre, comme application, on implémentera l'algorithme de johnson et al en utilisant le langage de programmation C++ .

Chapitre 1

Généralités

1.1 Introduction

Les graphes sont des outils irremplaçables pour modéliser et résoudre de nombreux problèmes concrets. En effet, ils permettent, d'une part de guider l'intuition lors d'un raisonnement, d'autre part de rattacher les résultats connus de la théorie des graphes.

1.2 Définitions et concepts de base

– **Graphe**
Un graphe G = (X,E) est un schéma constitué par un ensemble X (supposé ici fini) de points et un ensemble E reliant chacune deux de ceux-ci. Les points sont appelés les sommets du graphe, et les flèches les arcs du graphe. Et on distingue deux type de graphe orienté et non orienté.
notons :
$X = \{x_1, x_2, ..., x_n\}$ l'ensemble de sommets.
$E = \{e_1, e_2, ..., e_m\}$ l'ensemble des arêtes.
$|X| = n$ et $|E| = m$.

- **Graphe orienté**
  Un graphe orienté désigne où le couple (x, y) n'implique pas l'existence du couple (y, x) sur le dessin, les liens entre les sommets sont des flèches. Un ensemble $X = \{x_1, x_2, ..., x_n\}$ appelé ensemble des sommets, et une famille $E = \{e_1, e_2, ..., e_m\}$ qui est l'ensemble des arcs de G. Un arc u d'extrémité initiale x et d'extrémité terminale y sera noté $u = (x, y)$.

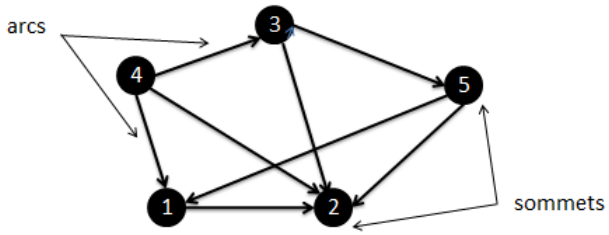


FIGURE 1.1 – Graphe orienté

- **Graphe non orienté**
  Lorsque le sens de parcours d'une relation entre deux sommets n'est pas important, on parle d'un graphe non orienté G = (X,E), avec $X = \{x_1, x_2, ..., x_n\}$ l'ensemble des sommets et $E = \{e_1, e_2, ..., e_m\}$ l'ensemble d'arêtes. Une arête c'est le nom d'un arc dans un graphe non orienté.

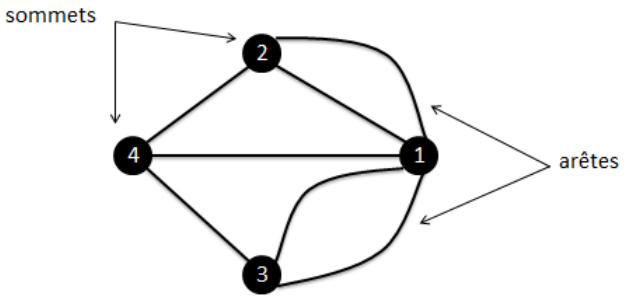


FIGURE 1.2 – Graphe non orienté

- **Ordre**

L'ordre d'un graphe est le nombre de ses sommets, noté par |X|.

- **Taille**

La taille d'un graphe est le nombre de ses arêtes.

- **Extrémité initiale et terminale(successeur et prédécesseur)**

Soit un arc (x, y), x est dit extrémité initiale de l'arc (x, y), et y extrémité terminale.
On dit aussi que y est successeur de x, et x prédécesseur de y. L'arc (x, y) est dit incident vers l'extérieur en x et vers l'intérieur en y.
On note $\Gamma^+(x)$ l'ensemble des successeurs de x, $\Gamma^-(x)$ l'ensemble des prédécesseurs de x, $d^+(x)$ demi-degré extérieur de x, c'est-à-dire le cardinal de $\Gamma^+(x)$, et $d^-(x)$ le demi-degré intérieur de x, c'est-à-dire le cardinal de $\Gamma^-(x)$.

- **Sommet adjacent**

Un sommet adjacent x est un sommet qui est relié à un autre sommet y par une arête $u = (x, y)$, et on dit que x et y sont adjacents(voisins).

- **Voisinage**

Le voisinage d'un sommet est l'ensemble de tous ses sommets adjacents. Cet ensemble est noté par $\Gamma(x)$, avec $\Gamma(x) = \{y \in X / \exists u \in E o\grave{u} u = (x, y) = (y, x)\}$

- **Chemin**

Soient $x, y \in X$. Un chemin de x à y dans G, est une suite S alternée de sommets et d'arcs, telle que : $S = (x = x_1, u_1, x_2, u_2, .., x_{p-1}, u_{p-1}, x_p = y)$; $p \leq n$, x_{i-1} est l'extrémité initiale de u_i et x_i son extrémité terminale, $\forall i, u_i = x_{i-1}x_i$

- **Circuit**

Un circuit de longueur (p+1) est un chemin de x à y avec x et y confondus ; $I(u_1) = T(u_{p-1})$

- **Chaîne**

Une chaîne de x à y dans G de longueur p est une suite alternée de sommets et d'arêtes $[x_1, e_1, x_2, e_2, .., x_{p-1}, e_{p-1}, x_p]$ telle que $p \leq n$,x_{i-1} est l'extrémité initiale de e_i et x_i son extrémité terminale, $\forall i, e_i = x_{i-1}x_i$.

- **Cycle**

Un cycle de longueur (p+1) est une chaîne dont les extrémités sont confondues $[x_1, e_1, x_2, e_2, .., x_{p-1}, e_{p-1}, x_p = x_1]$. La Chaîne et le cycle sont des notions non orientées, le chemin et le circuit des notions orientées.

- **Cycle eulérien**

Un cycle passant une et une seule fois par chaque arête du graphe.

- **Degré d'un sommet**

Le degré d'un sommet est le nombre d'arcs incident en ce sommet. Quand il n'y a pas de boucle en un sommet, c'est-à-dire d'arc dont l'extrémité initiale se confond avec l'extrémité terminale, son degré est la somme de son demi-degré intérieur et extérieur.

- **Clique**

Une clique est un sous-graphe complet(un ensemble de sommets deux-à- deux adjacents). Et elle est contenue dans une autre clique.

- **Clique maximale**

Une clique maximale est un sous-graphe complet, et cette clique n'est pas contenue dans une autre clique.

- **Clique maximum**

Une clique maximum est celle qui possède le plus grand nombre de sommets.

- **Sommet simplicial**

Soit un graphe G = (X,E) un sommet x est simplicial si $\Gamma(x)$ est une clique.

- **Racine**

G =(X,E) un graphe, $x_1 \in X$, x_1 est dit racine s'il existe un chemin de x_1 à x, $\forall x \in X$.

- **Connexité simple(non orienté)**

La relation $i\ C\ j$(lire i connecté à j) si $i = j$ ou s'il existe une chaîne allant de i à j est une relation d'équivalence dont les classes sont appelées les composantes connexes.

- **Connexité forte(orienté)**

La relation $i\ FC\ j$ si $i = j$ ou s'il existe un circuit passant par i et j est une relation d'équivalence dite de forte connexité. Les classes s'appellent les composantes fortement connexes.

- **Graphe connexe**

un graphe G=(X,E) est connexe si $\forall x, y \in X$ deux sommets distincts, $\exists$ une chaîne reliant ces deux sommets.

- **Graphe fortement connexe**

un graphe G=(X,E) est fortement connexe si $\forall x, y \in X$, $\exists$ un chemin de x à y et un chemin de y à x .

- **Graphe réduit**

Le graphe réduit GR est le graphe (X/FC,E) dont l'ensemble des sommets est l'ensemble des classes de forte connexité et dont tout arc relie deux classes distinctes dont deux éléments sont reliés dans G. Le graphe réduit est sans circuit.

- **Graphe partiel**

Un graphe $\overline{G} = (X, E^{'})$, $E^{'} \subset E$ est dit un graphe partiel de G = (X,E) où les sommets sont les points de X et les arêtes sont ceux de $E^{'}$. Autrement dit, on élimine de G les arêtes $(E - E^{'})$.

- **Graphe eulérien**

Un graphe est dit eulérien s'il admet un cycle eulérien.

- **Sous graphe**

un sous graphe d'un graphe G=(X,E) est une partie de sommets de X et les liens induits par E, si G représente les communes d'une wilaya, un sous-graphe engendré G_A possible, par exemple c'est les liaisons routières entre quatre communes choisies, telle que $G_A = (A, E_A)$ où $A \subseteq X$, et E_A= { $xy \in$ E /$x \in$ A et $y \in A$}.

- **Graphe complet**

Un graphe G = (X,E) est dit complet si chaque sommet est relié à tous les autres sommets, et qui est une clique maximale. Le graphe complet d'ordre n est noté K_n. Dans ce graphe chaque sommet est de degré $n - 1$.

- **Stable**

un ensemble $S \subset X$ est dit stable si deux sommets distincts de S ne sont pas voisins, autrement dit si $\Gamma_G \cap S = \emptyset$.

- **Solution partielle**

S^i est une solution partielle si et seulement si elle est une clique maximale du sous-graphe induit par les sommets $\{1, 2, ..., i-1\}$ et contenant i.

On suppose que les sommets de G sont numérotés de 1 à n.

- **Arborescence**

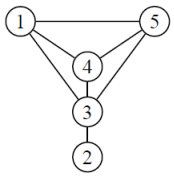
G =(X,E) est une arborescence si G est un arbre muni d'une racine.

- **Représentations d'un graphe**

On peut représenter un graphe avec plusieurs manières différentes.

1. **Matrice d'adjacence**

   On peut représenter un graphe par une matrice d'adjacences. Une matrice (n×m) est un tableau de n lignes et m colonnes. (i, j) désigne l'intersection de la ligne i et de la colonne j . Dans une matrice d'adjacences, les lignes et les colonnes représentent les sommets du graphe. Un « 1 » à la position (i, j) signifie que le sommet i est adjacent au sommet j. (une matrice d'adjacence est une matrice carrée).

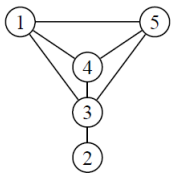


$$M = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

FIGURE 1.3 – matrice d'adjacence associé à ce graphe

2. **Liste d'adjacence**

   On peut aussi représenter un graphe simple en donnant pour chacun de ses sommets la liste des sommets auxquels il est adjacent.



1 : 3, 4, 5
2 : 3
3 : 1, 2, 4, 5
4 : 1, 3, 5
5 : 1, 3, 4

FIGURE 1.4 – liste d'adjacence associé à ce graphe

- **Ordre lexicographique**

Soient C_i, C_j, deux cliques induites par le graphe G = (X,E), x le premier sommet lexicographique de C_i, et y le premier sommet lexicographique de C_j, on dit que C_i est inférieure lexicographique à C_j et on note $C^i \leq_{lex} C_j$ si $i \leq_{lex} j$, $x \leq_{lex} y$.

1.3 Exemples

Exemple 1 : Graphe non orienté.

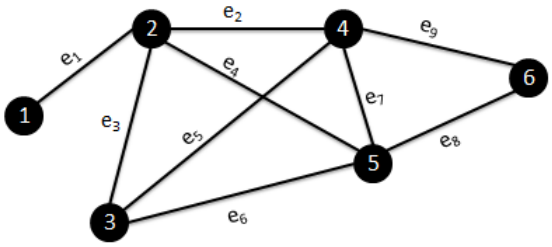


FIGURE 1.5 – Graphe d'exemple 1

Sommet adjacent	$\{6\}$ est adjacent à $\{4,5\}$	**Sous graphe complet**	$\{2,3,4,5\}$
Voisinage	voisins$\{4\}$ est l'ensemble $\{2,3,5,6\}$	**Ordre**	$n=6$
Sommet pendant	$\{1\}$	**Taille**	$m=9$
Chaine	$\{1,e_1,2,e_3,3,e_5,4,e_7,5,e_8,6\}$	**Stable**	$\{1,4\}$
Cycle	$\{4,e_7,5,e_8,6,e_9,4\}$	**Solution partielle**	$\{2,3,4,5\}$
Sommet simplicial	$\{6\}$	**Clique**	$\{2,3,4\}$
Connexité simple	$\{1,e_1,2,e_2,4,e_9,6\}$;$1C6$	**Clique maximale**	$\{4,5,6\}$
Graphe connexe	$\{1,2,3,4,5,6\}$	**Clique maximum**	$\{2,3,4,5\}$

Exemple 2 : Graphe orienté

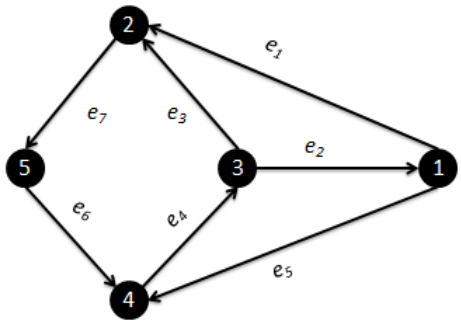


FIGURE 1.6 – Graphe d'exemple 2

Successeur	$\{2\}$ successeur de $\{1\}$	**Chemin**	$\{1,e_1,2,e_7,5,e_6,4\}$
prédécesseur	$\{1\}$ prédécesseur de $\{2\}$	**Circuit**	$\{2,e_7,5,e_6,4,e_4,3,e_3,2\}$
Degré d'un sommet	$d^+(1)+d^-(1)=3$	**Connexité forte**	$\{1,e_5,4,e_4,3,e_2,1\}$;$1FC4$

1.4 Classes de graphes

Il existent plusieurs classes de graphes, on en citera quelques unes .

- Graphe d'intervalles

Un graphe G = (X,E) est un graphe d'intervalle si et seulement si :
à $x \in X$, on associe un intervalle de la droite réelle I_x, et à $y \in X$, on associe I_y, alors $xy \in E \iff I_x \cap I_y \neq \emptyset$.

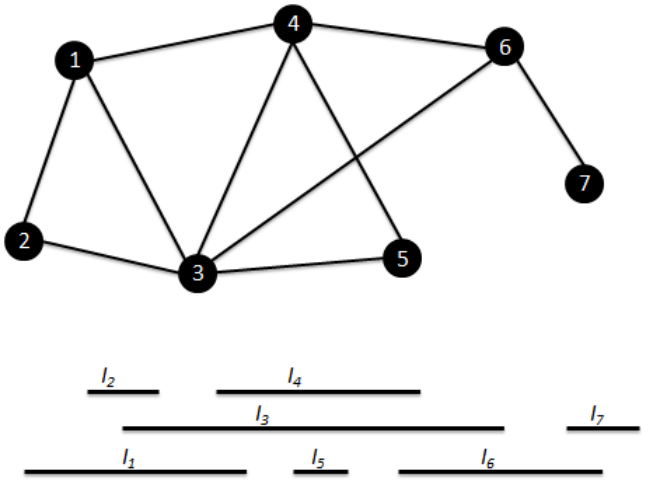
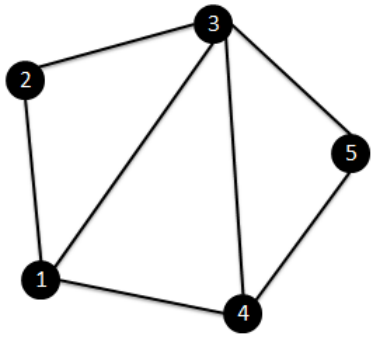


FIGURE 1.7 – Graphe d'intervalles

- Graphe triangulé

Un graphe G = (X,E) est dit triangulé s'il n'admet pas de cycle de longueur supérieure ou égale à 4 .



-

FIGURE 1.8 – Graphe triangulé

- Graphe de comparabilité

Un graphe est de comparabilité si on peut orienter ses arêtes de façon transitive, c'est-à-dire s'il existe un arc de i vers j et un arc de j vers k, alors il existe également un arc de i vers k.

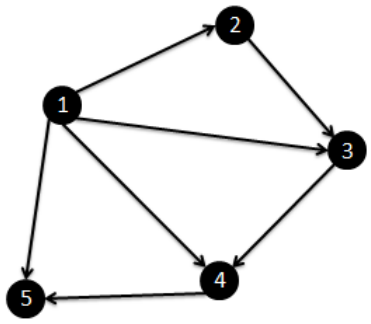


FIGURE 1.9 – Graphe de comparabilité

- Graphe planaire

Un graphe est planaire s'il est possible de le représenter sur le plan de telle sorte que ses arêtes ne se croisent pas. Et on a la taille d'un graphe planaire d'ordre $n \geqslant 3$ est liée par $m \leqslant 3(n-2)$.

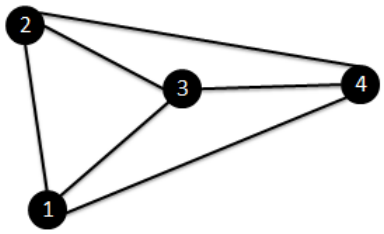


FIGURE 1.10 – Graphe planaire

- Graphe biparti

Un graphe est biparti si l'ensemble de ses sommets peut être partitionné en deux classes X_1 et X_2 de sorte que deux sommets de la même classe ne soient jamais adjacents. Il se note parfois $G = (X_1' X_2, U)$.

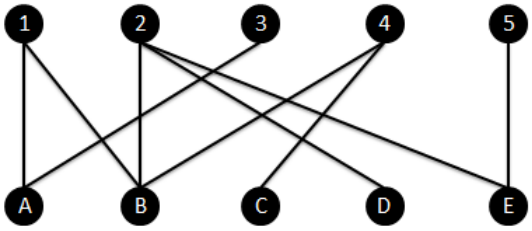


FIGURE 1.11 – Graphe biparti

- Graphe biparti complet

Un graphe est dit biparti complet s'il est biparti et contient le nombre maximal d'arêtes. En d'autres termes, il existe une partition de son ensemble de sommets en deux sous-ensembles X_1 et X_2 telle que chaque sommet de X_1 est relié à chaque sommet de X_2. Si X_1 est de cardinal m et X_2 est de cardinal n le graphe biparti complet est noté $K_{m,n}$.

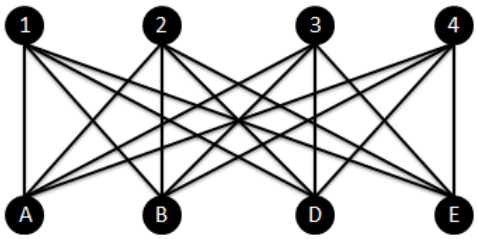


FIGURE 1.12 – Graphe biparti complet

- Graphe complémentaire

Le graphe complémentaire de G = (X,E) sera noté $\overline{G} = (X,\overline{E})$ avec :
$\overline{E} = \{xy/xy \notin E, x \neq y\}$.

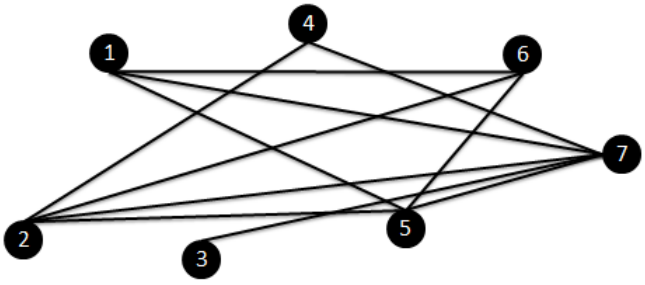


FIGURE 1.13 – Graphe complémentaire du graphe d'intervalles précédent

- **Arbre**

G = (X,E) est un arbre, s'il est connexe et sans cycle. Un arbre d'ordre n et de taille m est un graphe connexe sans cycle, alors $m = n - 1$.

Le sommet {1} est la racine de l'arbre.

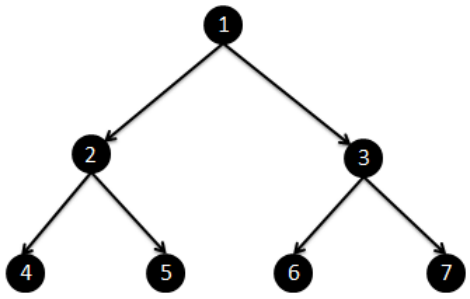


FIGURE 1.14 – Arbre

Parmi ces classes de graphes on essaiera de détailler celle de graphe distance héréditaire.

- **Graphe distance héréditaire**

La classe des graphes distance héréditaire a été introduite par Howorka[6], en 1977. On dit que G = (X,E) est distance héréditaire si la distance entre deux sommets quelconques x et y d'un sous-graphe connexe H de G est égale à la distance entre ces sommets dans G, $d_H(x,y) = d_G(x,y)$. Notons **DistH**[6] la classe des graphes à distance héréditaire.

Sommet pendant

Soit un graphe G(X,E), on appelle un sommet pendant $x \in X$, un sommet qui a un seul sommet adjacent, $\mid N(x) = 1 \mid$.

Sommets jumeaux

Deux sommets x et y sont des jumeaux si $N(x) \setminus y = N(y) \setminus x$. Si x et y sont adjacents, Alors ce sont de vrais jumeaux, Sinon s'ils ne sont pas adjacents, ce sont de faux jumeaux.

Séquence d'élagage

Est une séquence de suppression de sommets, telle que à chaque étape, le sommet supprimé est un sommet pendant ou un sommet qui a un jumeau.

Principe

Considérons les opérateurs suivants :

$xTy \Longleftrightarrow x$ et y sont de vrais jumeaux et que x doit être supprimé .

$xFy \Longleftrightarrow x$ et y sont de faux jumeaux et que x doit être supprimé.

$xPy \Longleftrightarrow x$ est un sommet pendant, ayant pour unique voisin y, et que x doit être supprimé.

Dans un graphe distance héréditaire, on peut toujours trouver deux sommets x et y non adjacents qu'ont le même voisinage, donc on va appliquer l'opérateur F successivement sur le graphe, après avoir supprimé tous les sommets pendants. On s'arrête lorsque le graphe contracté est une clique maximale, et pour générer toutes les cliques maximales du graphe de départ, on va décontracter le graphe obtenu.

Théorème(Bandelt-Mulder)[6]
Un graphe G = (X,E) est distance héréditaire si seulement s'il admet une séquence d'élagage de longueur n.

Exemple

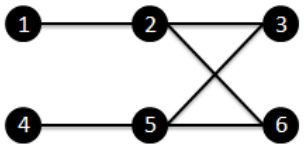


FIGURE 1.15 – Graphe de l'exemlpe 5

Étape 1 : On va vérifier si le graphe G est distance héréditaire :

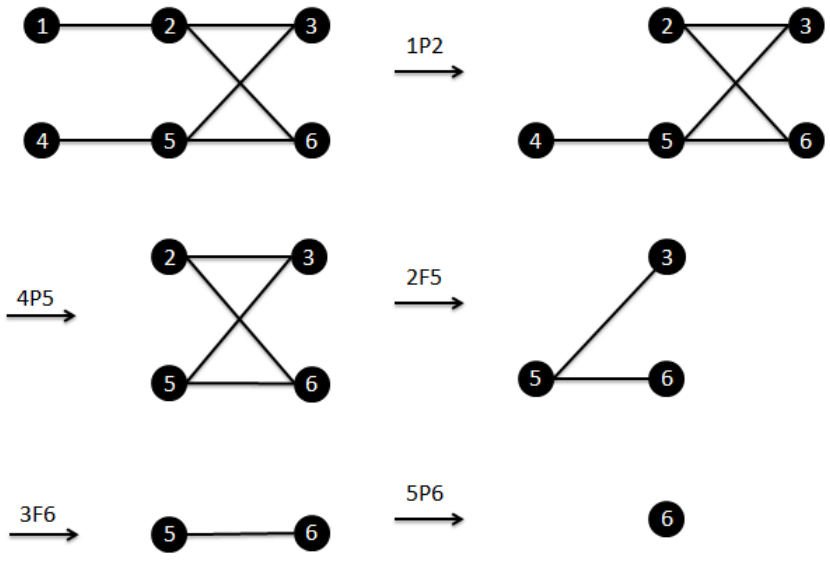


FIGURE 1.16 – Graphe de vérification

Le graphe admet une séquence d'élagage de longueur n = 6, donc le graphe est distance héréditaire.

Étape 2 : On va appliquer l'opérateur F successivement sur le graphe G.

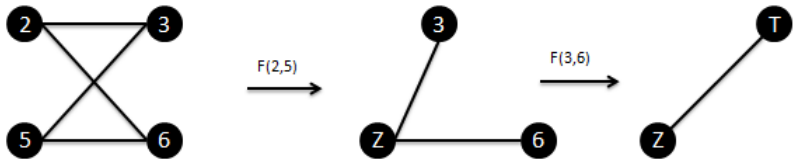


FIGURE 1.17 – Graphe aprés la contraction

Le graphe obtenu est une clique ZT, à partir de laquelle on génère toutes les cliques maximales du graphe de départ.

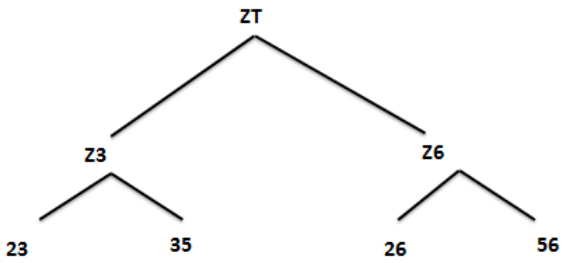


FIGURE 1.18 – Arbre de cliques

Les cliques maximales obtenues :$\{\mathbf{1,2}\}, \{\mathbf{4,5}\}, \{\mathbf{2,3}\}, \{\mathbf{2,6}\}, \{\mathbf{3,5}\}, \{\mathbf{5,6}\}$.

1.5 Complexité

Quelques définitions

Un problème est une question générale possède des paramètres dont le résultat n'est pas connu. Et parmi ces problèmes ce qu'on appelle les problèmes combinatoires discrets, continus et les problèmes aléatoire.
Un problème d'optimisation consiste à déterminer la solution optimale parmi toutes les solutions réalisables.
complexité temporelle
La complexité temporelle d'un algorithme correspond au nombre d'instructions élémentaires (opérations arithmétique, comparaison, affectation...) effectuées au cours de son exécution.
complexité spatiale
d'un algorithme correspond au nombre de cases mémoires occupées par les données manipulées par l'algorithme au cours de son exécution.
Algorithme
Un algorithme de résolution d'un problème (p) donnée est une procédure, décomposable en opérations élémentaires transformant une chaîne de caractères représentant les données de n'importe quel exemple du problème (p) en une chaîne de caractères représentant les résultats de (p).
Algorithme polynomial : Un algorithme est dit polynomial si le nombre d'opérations élémentaires nécessaire pour résoudre un problème de taille n est une fonction polynomiale.
Algorithme efficace
On dit qu'un algorithme est efficace si, et seulement si, il est polynomial.

Définition de 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 nous intéressons pour un problème donné , à déterminer la difficulté de résolution de ce problème. pour cela, on distingue deux catégories : ceux qui sont résolus optimalement par des algorithmes efficaces (rapides), et ceux dont la résolution peut prendre un temps exponentiel sur les grandes instances cas. On parle respectivement d'algorithmes polynomiaux et exponentiels, pour évaluer et classer les divers algorithmes disponibles pour un problème de graphe, il nous faut utiliser une mesure de performance indépendante du langage et de l'ordinateur utilisés. Ceci est obtenu par la notion de complexité d'un algorithme, qui consiste à mettre en évidence les possibilités et les limites théoriques du processus calculatoire, en évaluant le nombre d'opérations caractéristiques de l'algorithme dans le pire des cas, et elle est noté O.

Classes de complexité

On regroupe les différents problèmes algorithmiques en plusieurs classes. Tout d'abord, on se restreint aux problèmes de décision : ceux auxquels on répond par "oui" ou "non".

- La première classe est la "classe P" (polynomial time). Elle regroupe l'ensemble des problèmes de décision pouvant être résolus en un temps polynomial par rapport à la taille de l'entrée. On peut considérer les problèmes de cette classe comme des problèmes dits "faisable" ou sont simplement "faciles à résoudre".
- La deuxième classe est la classe des problèmes de décision prise en un temps polynomial par rapport à la taille de l'entrée, par un algorithme non déterministe. Elle porte le nom de "classe NP".
- La dernière classe, est la "classe NP-complet", qui contient les problèmes de la classe NP tels que n'importe quel problème de NP leur est réductible en un temps polynomial. On dit qu'un problème d'optimisation combinatoire est NP-difficile si son problème de décision correspond est NP-complet. Un très grand nombre des problèmes d'optimisation du monde réel sont NP-difficiles, et donc tout progrès dans le traitement des problèmes NP aura un impact très important sur de nombreuses applications.
- La relation entre les problèmes P et NP
  La question "P = NP ?" est l'une des questions les plus importantes qui n'ont pas encore été résolues en informatique théorique. En fait, la réponse à cette question a construit un champ de recherche ouvert. La réponse à la question "P = NP ?" par "oui", revient à montrer que tous les problèmes de la classe NP sont dans la classe P. Autrement dit, peut-on trouver en un temps polynomial ce qu'on peut prouver en temps polynomial ? Cependant, aucune démonstration n'a été faite pour prouver que $NP \subseteq P$, ni que $P \nsubseteq NP$. La relation évidente c'est que $P \subseteq NP$. En effet, un problème qui peut être résolu en un temps polynomial par un algorithme déterministe, pourra aussi être résolu par un algorithme non déterministe. Ce qui est admis et connu jusqu'à maintenant, c'est que $P \neq NP$. Néanmoins, aucune preuve n'a encore été prouvée jusqu'à ce jour. Si on arrivera à trouver un algorithme permettant la résolution d'un problème NP-complet, on pourra résoudre tous les problème NP-complets en temps polynomial. La figure suivante représente les problème de la classe P,NP, NP-complet et les problèmes à statut est indéterminé .

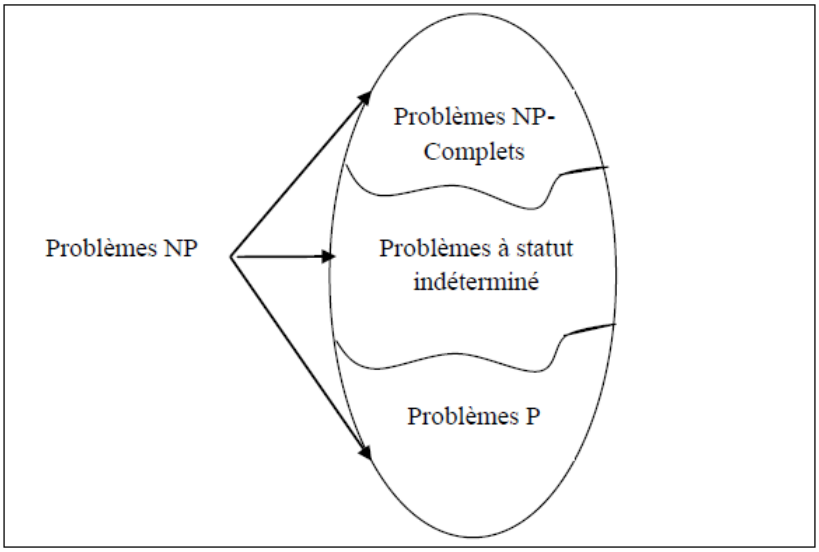


FIGURE 1.19 – Graphe de l'exemple 2

Échelle de comparaison de la complexité des algorithmes :

Valeur de la fonction	**Complexité**
1	constant
Log (n)	logarithmique
n	linéaire
n^2	quadratique
n^k	polynomiale
2^n	exponentielle

Chapitre 2

Algorithmes de génération des cliques maximales

2.1 Introduction

La génération des cliques maximales est un problème d'optimisation combinatoire, donc il sera mieux traité en utilisant la notion d'algorithme. Pour cela on va analyser et classer les algorithmes de génération de cliques maximales qui sont : l'algorithme de Johnson, All Clique, Tsukiyama, Dahlhaus. Pour découvrir le mode d'opération pour chaque algorithme suivant ces critères l'espace mémoire, le temps d'exécution et la génération suivant l'ordre lexicographique, avec l'application pour chaque algorithme.

2.2 Les techniques de génération

L'approche retour arrière
(backtracking) qui nous permet l'écriture d'algorithmes récursifs, simple à comprendre. L'idée utilisée est d'abord de générer les objets contenant un élément(ou vérifiant une propriété)puis, de générer les objets ne contenant pas cet élément (ou ne vérifiant pas cette propriété).
L'ordre lexicographique
est l'ordre naturel utilisé pour générer certains objets combinatoires. Les éléments sont dans un ordre total. L'idée principale est d'être capable de générer, d'une manière efficace, un objet commençant à partir d'un prédécesseur dans la suite. Et de savoir que le premier objet est trouvé.

2.3 Algorithme Lex-BFS

2.3.1 Introduction

L'algorithme LexBFS a été le premier algorithme linéaire de reconnaissance des graphes triangulés. Ces dernières années, il a été utilisé dans plusieurs applications. Il est à la base de la simplification du premier algorithme linéaire de reconnaissance des graphes d'intervalles. Il permet aussi d'ordonner un graphe G suivant l'ordre lexicographique avec la création d'un ordre entre les voisins d'un sommet sélectionné en utilisant l'étiquetage lexicographique des sommets à travers la recherche.

2.3.2 Principe

La première étape, c'est pour initialiser chaque sommet du graphe à un ensemble vide. La deuxième étape, consiste à créer un ordre σ de sommets de V, d'une manière suivante : choisir un sommet de départ s et l'initialiser à n, qui devient le plus grand suivant l'ordre lexicographique.

pour chaque sommet i de n à 1 faire

choisir un sommet non numéroté x d'étiquette maximum, et $\sigma(i) := x$

pour chaque voisin non numéroté y de x faire

on ajoute (i) à tous les sommets y.

2.3.3 L'algorithme

Algorithme 1 Lex-BFS

Entrée : Un graphe G = (V,E) et un sommet de source s.
Sortie : Un ordre σ de V.

pour chaque sommet $x \in V$ **faire**
  $étiquette(x) \leftarrow \varnothing$;
fin pour
$étiquette(s) \leftarrow \{n\}$
pour $i \leftarrow n$ à 1 **faire**
  choisir un sommet non numéroté $x \in V$ d'étiquette maximum
  $\sigma(i) = x$;
  pour chaque voisin non numéroté y de x **faire**
    $étiquette(y) := étiquette(y) \cup \{i\}$; // Concaténation de i à la fin de l'étiquette y.
  fin pour
fin pour

Les propriétés du parcours

Propriété 2.1 [8] *si G est triangulé, le dernier sommet visité par LexBFS est simplicial.*

Propriété 2.2 [8] *si G est un graphe de comparabilité, alors le dernier sommet visité par LexBFS sur $\overline{G}$ peut être choisi comme source d'une orientation transitive de G.*

Remarque 2.1 [8] *Les propriétés du parcours LexBFS sont particulièrement intéressantes sur les classes de graphes distances héréditaires(quand on enlève un sommet au graphe, il reste dans la classe).*

2.3.4 Exemple

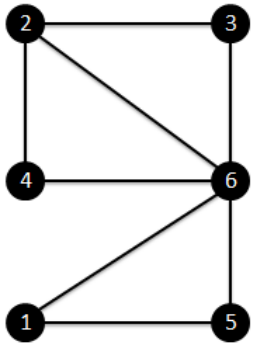


FIGURE 2.1 – Graphe non ordonné de l'exemple 1

entrée : G = (V,E) et un sommet de départ s.
sortie : un ordre σ de V.
Début
pour chaque sommet $x \in$ V **faire**
x = 1 : étiquette(1) = $\varnothing$;
x = 2 : étiquette(2) = $\varnothing$;
x = 3 : étiquette(3) = $\varnothing$;
x = 4 : étiquette(4) = $\varnothing$;
x = 5 : étiquette(5) = $\varnothing$;
x = 6 : étiquette(6) = $\varnothing$;

étiquette(1) = 6 ;

pour i = 6 à 1 **faire**

i = 6 : x = 1, $\sigma(6) = 1$

pour chaque voisin non numéroté y de 1 **faire**

Ajouter(6) à étiquette(5)

Ajouter(6) à étiquette(6)

i = 5 : x = 5, $\sigma(5) = 5$

pour chaque voisin non numéroté y de 5 **faire**

Ajouter(5) à étiquette(6)

i = 4 : x = 6, $\sigma(4) = 6$

pour chaque voisin non numéroté y de 6 **faire**

Ajouter(4) à étiquette(2)

Ajouter(4) à étiquette(3)

Ajouter(4) à étiquette(4)

i = 3 : x = 4, $\sigma(3) = 4$

pour chaque voisin non numéroté y de 4 **faire**

Ajouter(3) à étiquette(2)

i = 2 : x = 2, $\sigma(2) = 2$

pour chaque voisin non numéroté y de 2 **faire**

Ajouter(2) à étiquette(3)

i = 1 : x = 3, $\sigma(1) = 3$

n'existe pas un sommet non numéroté y de 3.

σ = (1, 5, 6, 4, 2, 3)

G(V,E) ordonné suivant l'ordre lex-BFS :

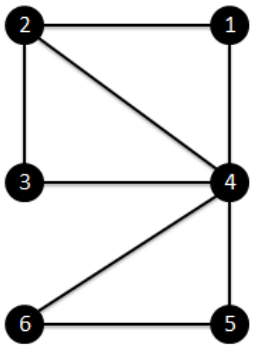


FIGURE 2.2 – Graphe ordonné de l'exemple 1

2.4 Algorithme de Johnson et al

2.4.1 Principe

L'algorithme de Johnson et al est utilisé dans les problèmes d'ordonnancement et notamment les files d'attente. Il est aussi utile dans la génération des cliques maximales ; cet algorithme utilise le parcours lexicographique, basé sur l'ordre total des sommets, la première étape consiste à déterminer la première clique suivant l'ordre lexicographique, et la deuxième étape utilise une structure permettant de stocker un ensemble de cliques maximales et d'en extraire la plus petite lexicographique.

Algorithme 2 première clique

Entrée : un graphe G = (X,E).
Sortie : la première clique maximale dans l'ordre lexicographique.

$C \leftarrow 1$;
pour i = 2 à n **faire**
  si $\Gamma(i) \cap C = C$ **alors**
    $C = C + \{i\}$
  fin si
fin pour

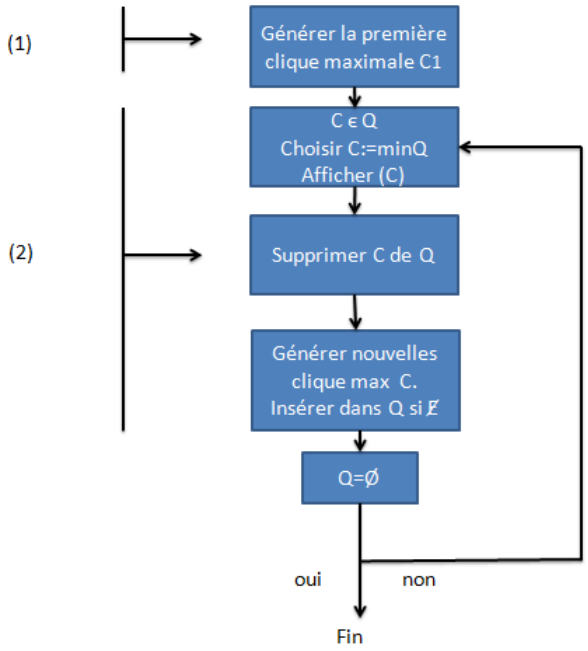


FIGURE 2.3 – Schéma représentant l'évolution des solutions partielles

L'étape A, soit C_1 la première clique obtenue à travers l'algorithme 2 qu'il va l'insérer dans Q qui est un ensemble vide au départ, donc C_1 la plus petite clique dans Q suivant l'ordre lexicographique, et celle qui affiche l'algorithme.
L'étape B, l'algorithme il génère à partir de C = min(Q) un ensemble de cliques maximales plus grandes suivant l'ordre lexicographique que C, et insérer dans Q celles qui n'existent pas, pour générer les cliques, l'algorithme il procède de la manière suivante :
pour chaque sommet i $\in C$ faire

pour tout sommet $j \notin \Gamma(i)$ et $i < j$ faire
$(C \cap \{1, ..., j\} \cap \Gamma(j)) \cup \{j\}$ est une clique maximale du sous-graphe induit par les sommet $\{1, ..., j\}$?
si oui, Alors $\exists\ C^{'}$, une clique maximale, telle que $C^{'} = (C \cap \{1, ..., j\} \cap \Gamma(j)) \cup \{j\}$
si $C^{'}$ n'existe pas dans Q, Alors insérer $(C^{'})$ et afficher $(C^{'})$
on passe au j suivant

on génère à partir de C = min(Q)
La condition d'arrêt de l'algorithme lorsque $Q = \varnothing$, et toutes les cliques maximales du graphe ont été générées dans l'ordre lexicographique.

2.4.2 L'algorithme

Algorithme 3 Algorithme de johnson et al.

Entrée : un graphe G=(X,E) non orienté.
Sortie : générer toutes les cliques maximales de G dans l'ordre lexicographique.

Début
soit C_1 la premiére clique maximale de G ;
insérer C_1 dans Q ;
tant que $Q \neq \varnothing$ **faire**
début
C := minQ ;
afficher (C) ;
pour chaque sommet $i \in C$ **faire**
début
pour tout sommet j de G non adjacent à i tels que $i < j$ **faire**
début
si $(C \cap \{1, ..., j\} \cap \Gamma(j) \cup \{j\})$ est une clique maximale du sous graphe induit par les sommets $\{1, ..., j\}$ **alors**
début
soit $C^{'}$ la première clique maximale suivant l'ordre lexicographique de G contenant $(C \cap 1, ..., j \cap \Gamma(j)) \cup j$
si $C^{'} \notin Q$ **alors** insérer $C^{'}$ dans Q et supprimer C
fin début
fin début
fin pour
fin début
fin pour
fin début
fin tant que
fin Début

Théorème 2.1 [4] *L'algorithme 3 génère toutes les cliques maximales d'un graphe G = (X,E) suivant l'ordre lexicographique en $O(n^3)$ pour chaque clique.*

2.4.3 Complexité

Pour extraire la première clique maximale dans la file Q, le temps demandé est $O(nlog(|C(G)|))$, suivi par le calcul de cliques maximales avec le temps $O(n+m)$ pour chaque nouvelle clique, ajoutons le temps de l'insertion et la comparaison aux cliques existantes dans la file Q qui est $O(nlog(|C(G)|))$. On obtient le délai total est $O(nlog(|C(G)|)+n+m+nlog(|C(G)|)) = O(n^3)$ qui est polynomial.

2.4.4 Exemple

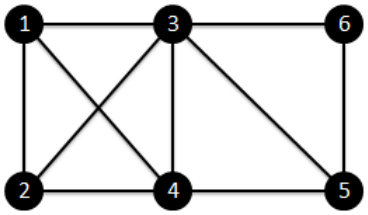


FIGURE 2.4 – Graphe de l'exemple 2

L'étape(1) : Trouver C_1 la première clique maximale suivant l'ordre lexicographique par l'algorithme 1.

$C = \{1\}$

pour i = 2 à 6 faire

i=2 : $\Gamma(2) \cap \{1\} = \{1,3,4\} \cap \{1\} = \{1\}$

Donc $C = \{1\} + \{2\} = \{1,2\}$

i=3 : $\Gamma(3) \cap \{1,2\} = \{1,2,4,5,6\} \cap \{1,2\} = \{1,2\}$

Donc $C = \{1,2\} + \{3\} = \{1,2,3\}$

i=4 : $\Gamma(4) \cap \{1,2,3\} = \{1,2,3,5\} \cap \{1,2,3\} = \{1,2,3\}$

Donc $C = \{1,2,3\} + \{4\} = \{1,2,3,4\}$

i=5 : $\Gamma(5) \cap \{1,2,3,4\} = \{3,4,6\} \cap \{1,2,3,4\} = \{3,4\} \neq \{1,2,3,4\}$

i=6 : $\Gamma(6) \cap \{1,2,3,4\} = \{3,5\} \cap \{1,2,3,4\} = \{3\} \neq \{1,2,3,4\}$

Donc $C_1 = \{1,2,3,4\}$ est la première clique maximale dans l'ordre lexicographique.

L'étape(2) :

$C_1 = \{1,2,3,4\}$

$Q = \{1,2,3,4\}$

$C := minQ \Rightarrow C := \{1,2,3,4\}$

afficher(C)

- pour i = 1 : les $j \notin \Gamma(i)$ et $i < j$:

  $\Gamma(1) = \{2,3,4\}$ donc $j = 5,6$

  $j = 5 : (\{1,2,3,4\} \cap \{1,2,3,4,5\} \cap \Gamma(5)) \cup \{5\}$

  $= (\{1,2,3,4\} \cap \{1,2,3,4,5\} \cap \{3,4,6\}) \cup \{5\}$

  $= \{3,4\} \cup \{5\} =$ **{3,4,5}** est une clique maximale du sous-graphe induit par les sommets

  $\{1,2,3,4,5\}$, donc **{3,4,5}** $\in Q$.

$j = 6 : (\{1,2,3,4\} \cap \{1,2,3,4,5,6\} \cap \Gamma(6)) \cup \{6\}$
$= (\{1,2,3,4\} \cap \{1,2,3,4,5,6\} \cap \{3,5\}) \cup \{6\}$
$= \{3\} \cup \{6\} = \{3,6\}$ n'est pas une clique maximale du sous-graphe induit par les sommets $\{1,2,3,4,5,6\}$

- pour i = 2 : les $j \notin \Gamma(i)$ et $i < j$:
$\Gamma(2) = \{1,3,4\}$ donc $j = 5,6$

$j = 5 : (\{1,2,3,4\} \cap \{1,2,3,4,5\} \cap \Gamma(5)) \cup \{5\}$
$= (\{1,2,3,4\} \cap \{1,2,3,4,5\} \cap \{3,4,6\}) \cup \{5\}$
$=\{3,4\} \cup \{5\} =$ **{3,4,5}** est une clique maximale du sous-graphe induit par les sommets
$\{1,2,3,4,5\}$, donc **{3,4,5}**

j = 6 : $(\{1,2,3,4\} \cap \{1,2,3,4,5,6\} \cap \Gamma(6) \cup \{6\}$
$= (\{1,2,3,4\} \cap \{1,2,3,4,5,6\} \cap \{3,5\}) \cup \{6\}$
$= \{6\} \cup \{6\} = \{3,6\}$ n'est pas une clique maximale du sous-graphe induit par les sommets $\{1,2,3,4,5,6\}$

- pour i = 3 : les $j \notin \Gamma(i)$ et $i < j$:
$\Gamma(3) = \{1,2,3,4,5,6\}$ donc $j = \varnothing$

- pour i = 4 : les $j \notin \Gamma(i)$ et $i < j$:
$\Gamma(4) = \{1,2,3,5\}$ donc $j = 6$

j = 6 :$(\{1,2,3,4\} \cap \{1,2,3,4,5,6\} \cap \Gamma(6)) \cup \{6\}$
$= (\{1,2,3,4\} \cap \{1,2,3,4,5,6\} \cap \{3,5\}) \cup \{6\}$
$= \{3\} \cup \{6\} = \{3,6\}$ n'est pas une clique maximale du sous-graphe induit par les sommets $\{1,2,3,4,5,6\}$
supprimer la clique $\{1,2,3,4\}$
Donc $Q =$ **{3,4,5}**
$C := minQ \Rightarrow C := \{3,4,5\}$
afficher (C)

- pour i = 3 : les $j \notin \Gamma(i)$ et $i < j$:
$\Gamma(3) = \{1,2,4,5,6\}$ donc $j = \varnothing$

- pour i = 4 : les $j \notin \Gamma(i)$ et $i < j$:
$\Gamma(4) = \{1,2,3,5\}$ donc $j = 6$

j = 6 : $(\{3,4,5\} \cap \{1,2,3,4,5,6\} \cap \Gamma(6)) \cup \{6\}$
$= (\{3,4,5\} \cap \{1,2,3,4,5,6\} \cap \{3,5\}) \cup \{6\}$
$=\{3,5\} \cup \{\mathbf{6}\}$ est une clique maximale du sous-graphe induit par les sommets $\{1,2,3,4,5,6\}$
donc {**3,5,6**} $\in Q$

- pour i = 5 : les $j \notin \Gamma(i)$ et $i < j$:
  $\Gamma(5) = \{3,4,6\}$ donc $j = \varnothing$
  supprimer la clique $\{3,4,5\}$
  donc $Q =$ {**3,5,6**}
  $C := Q \Rightarrow C := \{3,5,6\}$
  afficher(C), $Q = \varnothing$
  Donc toutes les cliques maximales sont générées suivant l'ordre lexicographique, qui sont : {**1,2,3,4**}, {**3,4,5**}, {**3,5,6**}.

2.5 Algorithme Allclique

2.5.1 Principe

Cet algorithme utilise l'approche du retour arrière qui permet une écriture simple de l'algorithme avec l'utilisation de fonctions et procédures récursives. Pour chaque sommet, l'algorithme génère d'abord les ensembles de sommets qui le contiennent ou partage avec ce sommet une propriété ; ensuite, il génère les ensembles qui ne contiennent pas ce sommet ou alors les ensembles qui ne partage pas avec le sommet une propriété. Après cela, il génère à chaque profondeur d'empilement une ou plusieurs solutions partielles qui seront exploitées a fin de trouver une clique maximale. Contrairement à l'algorithme précédent, l'algorithme All Clique utilise un espace mémoire polynomial et prend un délai d'exécution exponentiel.

Formulation :

Soit S^i une solution partielle, donnée à la profondeur k. F_{k+1} sera donner par : $F_{k+1} = \{x \in X / S^i : xy \in E$, Pour $y \in S^i et x > i\}$ avec $F_1 = X$.
Nous pouvons l'écrire aussi sous la forme suivante :
$F_{k+1} = \{x \in F_{k+1}/i; x_i \in E$ et $x > i\}$.
Ceci signifie que les sommets qu'on peut ajouter sont les sommets qui était possible à l'itération d'avant, sauf i qui est déjà dans la solution partielle courante mais ces sommets doivent être adjacent et supérieure strictement à i. Par suite : $F_{k+1} = F_k \cap adj(x)$. Enfin pour déterminer si la solution courante s'agit d'une clique maximale du graphe, nous définissons l'ensemble :$N_{k+1} = N_{k+1} \cap adj(x)$ avec $N_1 = X$, alors la solution partielle S^i est une clique si et seulement si $N_{k+1} = \varnothing$.

2.5.2 L'algorithme

Algorithme 4 Algorithme Allclique et al.

Entrée : un graphe G = (X , E) non orienté.
Sortie : les cliques maximales de G.

```
Début
  C := ∅;
  Allclique (1,1) ;
fin Début
Allclique (i,j)
Début
  si i ⩽ n alors
    si i = 1 alors
   Début
    N_i = X;
    F_i = X;
   fin Début
    sinon
     Début
      N_i = N_{i-1} ∩ Γ(j);
      F_i = F_{i-1} ∩ Γ(j);
     fin Début
fin Début
   si N_i = ∅ alors  afficher(C) ;
   pour chaquej ∈ F_i ( dans l'ordre croissant )faire
   Début
     siC ∪ {j} clique max du sous graphe induit par les sommets {1,...j}alors
      Début
      C = C ∪ {j};
      Allclique (i+1,j)
      C = C − {j}
      fin Début
    fin Début
   fin pour
  fin Début
fin Début
```

2.5.3 Complexité

La complexité de cet algorithme est $O(n^{Log_2(n)+2})$, qui est exponentielle.

2.5.4 Exemple

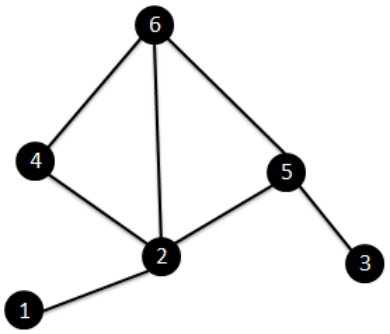


FIGURE 2.5 – Graphe de l'exemple 3

Pour $k = 1$, S^0 *et* $F_1 = \{1,2,3,4,5,6\}$.
Pour chaque $i \in F_1$, nous allons traiter la solution partielle $S^i = \{i\}$.
$F_{k+1} = \{x \in F_k / \{i\},\ xi \in E \text{ et } x > i\}$.

$S^1 = \{1\}$
$\{1\} \cap \Gamma(2) = \{1\} \cap \{1,4,5,6\} = \{1\}$ Donc $\{1\} \cup \{2\} = \{1,2\}$ est une clique maximale du sous-graphe induit par les sommets $\{1,2\}$ et $S^2 = \{1,2\}$ nouvelle solution partielle .
$\{1\} \cap (3) = \{1\} \cap \{5\} = \varnothing \neq \{1\}$ Donc $\{1\} \cup \{3\} = \{1,3\}$ n'est ni clique maximale du sous-graphe induit par les sommets $\{1,2,3\}$ ni nouvelle solution partielle .
$\{1\} \cap \Gamma(4) = \{1\} \cap \{2,6\} = \varnothing \neq \{1\}$ Donc $\{1\} \cup \{4\} = \{1,4\}$ n'est ni clique maximale du sous-graphe induit par les sommets $\{1,2,3,4\}$ ni solution partielle .
$\{1\} \cap \Gamma(5) = \{1\} \cap \{2,3,6\} = \varnothing \neq \{1\}$ Donc $\{1\} \cup \{5\} = \{1,5\}$ n'est ni clique maximale du sous-graphe induit par les sommets $\{1,2,3,4,5\}$ ni solution partielle .
$\{1\} \cap \Gamma(6) = \{1\} \cap \{2,4,5\} = \varnothing \neq \{1\}$ Donc $\{1\} \cup \{6\} = \{1,6\}$ n'est ni clique maximale du sous-graphe induit par les sommets $\{1,2,3,4,5,6\}$ ni solution partielle.

$S^2 = \{1,2\}$
$\{1,2\} \cap \Gamma(3) = \{1,2\} \cap \{5\} = \varnothing \neq \{1,2\}$ Donc $\{1,2\} \cup \{3\} = \{1,2,3\}$ n'est ni clique maximale induit par les sommets $\{1,2,3\}$, ni solution partielle .
$\{1,2\} = \cap \Gamma(4) = \{1,2\} \cap \{2,6\} = \{2\} \neq \{1,2\}$ Donc $\{1,2\} \cup \{4\} = \{1,2,4\}$ n'est ni clique maximale du sous-graphe induit par les sommets $\{1,2,3,4\}$, ni solution partielle.
$\{1,2\} \cap \Gamma(5) = \{1,2\} \cap \{2,3,6\} = \{2\} \neq \{1,2\}$ Donc $\{1,2\} \cup \{5\} = \{1,2,5\}$ n'est ni clique maximale du sous-graphe induit par les sommets $\{1,2,3,4,5\}$,ni solution partielle.
$\{1,2\} \cap \Gamma(6) = \{1,2\} \cap \{2,4,5\} = \{2\} \neq \{1,2\}$ Donc $\{1,2\} \cup \{6\} = \{1,2,6\}$ n'est ni clique

maximale du sous-graphe induit par les sommets $\{1,2,3,4,5,6\}$,ni solution partielle . Alors {**1,2**}est une clique maximale du graphe G.

$S^2 = \{2\}$
$\{2\} \cap \Gamma(3) = \{2\} \cap \{5\} = \varnothing \neq \{2\}$ Donc $\{2\} \cup \{3\} = \{2,3\}$ n'est ni clique maximale du sous-graphe induit par les sommets $\{1,2,3\}$, ni solution partielle.
$\{2\} \cap \Gamma(4) = \{2\} \cap \{2,6\} = \{2\}$ Donc $\{2\} \cup \{4\} = \{2,4\}$ est une clique maximale du sous-graphe induit par les sommets $\{1,2,3,4\}$ et $S^4 = \{2,4\}$ solution partielle.
$\{2\} \cap \Gamma(5) = \{2\} \cap \{2,3,6\} = \{2\}$ Donc $\{2\} \cup \{5\} = \{2,5\}$ est une clique maximale du sous-graphe induit par les sommets $\{1,2,3,4,5\}$ et $S^5 = \{2,5\}$ solution partielle.
$\{2\} \cap \Gamma(6) = \{2\} \cap \{2,4,5\} = \{2\}$ Donc $\{2\} \cup \{6\} = \{2,6\}$ mais $\{2,6\}$ n'est ni clique maximale du sous-graphe induit par les sommets $\{1,2,3,4,5,6\}$, ni solution partielle.

$S^4 = \{2,4\}$
$\{2,4\} \cap \Gamma(5) = \{2,4\} \cap \{2,3,6\} = \{2\} \neq \{2,4\}$ Donc $\{2,4\} \cup \{5\} = \{2,4,5\}$ n'est ni clique maximale du sous-graphe induit par les sommets $\{1,2,3,4,5\}$, ni solution partielle.
$\{2,4\} \cap \Gamma(6) = \{2,4\} \cap \{2,4,5\} = \{2,4\}$ Donc $\{2,4\} \cup \{6\} = \{2,4,6\}$ est une clique maximale du sous-graphe induit par les sommets $\{1,2,3,4,5,6\}$, et $S^6 = \{2,4,6\}$ solution partielle .

$S^5 = \{2,5\}$
$\{2,5\} \cap \Gamma(6) = \{2,5\} \cap \{2,4,5\} = \{2,5\}$ Donc $\{2,5\} \cup \{6\} = \{2,5,6\}$ est une clique maximale du sous-graphe induit par les sommets $\{1,2,3,4,5,6\}$ et $S^6 = \{2,5,6\}$ solu -tion partielle .

$S^6 = \{2,4,6\}$
{**2,4,6**} est une clique maximale du graphe G car il n'existe pas un sommet de $F_1/\{2,4,6\}$ qui est supérieur à 2, à 4 et 6, donc $F_2 = \varnothing$.

$S^6 = \{2,5,6\}$
{**2,5,6**} est une clique maximale du graphe G car il n'existe pas un sommet de $F_1/\{2,5,6\}$ qui est supérieur à 2, à 5 et 6, donc $F_2 = \varnothing$.

$S^3 = \{3\}$
$\{3\} \cap \Gamma(4) = \{3\} \cap \{2,6\} = \varnothing \neq \{3\}$ Donc $\{3\} \cup \{4\} = \{3,4\}$ n'est ni clique maximale du sous graphe induit par les sommets $\{1,2,3,4\}$, ni solution partielle .
$\{3\} \cap \Gamma(5) = \{3\} \cap \{2,3,6\} = \{3\}$ Donc $\{3\} \cup \{5\} = \{3,5\}$ est une clique maximale du sous-graphe induit par les sommets $\{1,2,3,4,5\}$ et $S^5 = \{3,5\}$ solution partielle.

$\{3\} \cap \Gamma(6) = \{3\} \cap \{2,4,5\} = \varnothing \neq \{3\}$ Donc $\{3\} \cup \{6\} = \{3,6\}$ n'est ni clique maximale du sous-graphe induit par les sommets $\{1,2,3,4,5,6\}$, ni solution partielle.

$S^5 = \{3,5\}$
$\{3,5\} \cap \Gamma(6) = \{3,5\} \cap \{2,4,5\} = \{5\} \neq \{3,5\}$ Donc $\{3,5\} \cup \{6\} = \{3,5,6\}$ n'est ni clique maximale du sous-graphe induit par les sommets $\{1,2,3,4,5,6\}$, ni solution partielle.
Alors **{3,5}** est une clique maximale du graphe G.

$S^4 = \{4\}$
$\{4\} \cap \Gamma(5) = \{4\} \cap \{2,3,6\} = \varnothing \neq \{4\}$ Donc $\{4\} \cup \{5\} = \{4,5\}$ n'est ni clique maximale du sous-graphe induit par les sommets $\{1,2,3,4,5\}$, ni solution partielle.

$S^5 = \{5\}$
$\{5\} \cap \Gamma(6) = \{4\} \cap \{2,4,5\} = \{4\}$ Donc $\{4\} \cup \{6\} = \{4,6\}$ mais $\{4,6\}$ n'est ni clique maximale du sous-graphe induit par les sommets $\{1,2,3,4,5,6\}$, ni solution partielle .
Donc l'ensemble des cliques maximales de G suivant l'ordre lexicographique sont :
{1,2}, **{2,4,6}**, **{2,5,6}**, **{3,5}**.

2.6 Algorithme de Tsukiyama et al

2.6.1 Principe

C'est un algorithme qui se base aussi sur l'approche du retour arrière pour générer les cliques maximales en utilisant un espace mémoire linéaire O(n+m) et ayant une complexité en O(n.m) par clique. Ceci signifie qu'il est de délai polynomial, utilise un espace polynomial mais malheureusement , il génère les cliques maximales du graphe suivant un ordre quelconque.
A chaque itération $i = \{1,...,n\}$, l'algorithme génère toutes les cliques maximales du sous graphe induit par les sommets $\{1,....,i\}$ en leurs ajoutant tous les sommets supérieurs à i. A l'itération i = n, toutes les cliques maximales du graphe seront générées. Pendant l'exécution, il génère à chaque profondeur d'empilement une ou plusieurs solutions partielles qui seront exploitées a fin de trouver une clique maximale. Une solution partielle S_i est notée par le couple (C^i, X^i), où C^i est une clique maximale du sous-graphe induit par les sommets $\{1,....,i\}$ et $X^i = \{i+1,...,n\}$.
A la profondeur k = 0, nous obtenons qu'une seule solution partielle qui est $S^0 = (\varnothing, X)$ et à k=1, $S^1 = (1,2,...,n) = X$. Pour $\{1,....,n\}$, Si est construite à partir de S^{i-1} de la manière suivante :
- si i est adjacent à tous les sommets S_{i-1}, alors $\{C_{i-1} \cup i\}$ est une clique maximale du graphe

induit par les sommets $\{1,....,i\}$. Ainsi on obtient $S^i = (C^{i-1} \cup i, X^i)$, qui est une solution partielle de la profondeur k=i, à partir de S^{i-1}

- s'il existe au moins un sommet de Ci qui n'est pas adjacent à i+1, alors Ci est une clique maximale du graphe induit par les les sommets $\{1,....,i\}$. Ainsi on obtient $S^i = (C^i, X^i)$, qui est une solution partielle de la profondeur k = i, à partir de S^{i-1}, avec . Pour trouvée la prochaine solution partielle, il est important que chaque solution partielle trouvée à chaque itération i-1 soit traitée de sorte qu'on puisse avoir d'autres solutions partielles dans chaque profondeur.

- si $((C^{i-1} \cap adj(i)) \cup i)$ est une clique maximale du graphe induit par les premiers i sommets alors on obtient une deuxième solution partielle issue de $\{S_{i-1}\}$ qui est $S^i = (C^i, X^i)$ avec $((C^{i-1} \cap adj(i)) \cup i)$.

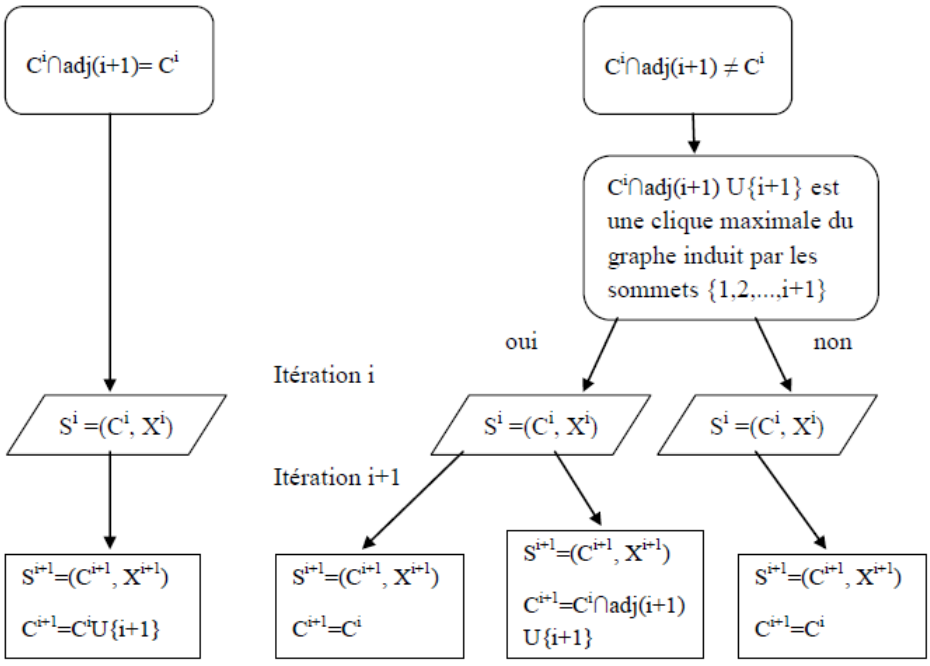


FIGURE 2.6 – Schéma représentant l'évolution des solutions partielles

2.6.2 Complexité

Le temps d'exécution de cet algorithme est O(n.m.|C(G)|). Le délai d'exécution n'est donc pas exponentiel et il utilise un espace mémoire de O(n+m). L'inconvénient de cet algorithme est le fait qu'il ne génère pas les cliques suivant l'ordre lexicographique mais il respecte tout de même deux des critères (l'espace et le délai polynomial).

2.6.3 Exemple

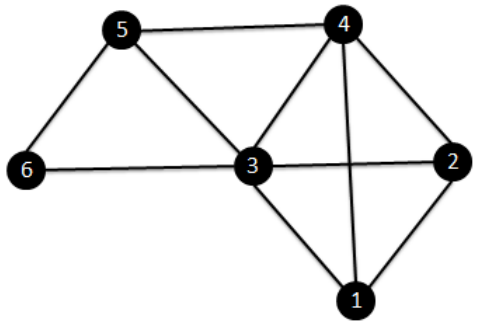


FIGURE 2.7 – Graphe de l'exemple 4

$S^i = (C^i, X^i)$
A la profondeur 0, on a : $S^0 = (\varnothing, X)$
$S^1 = (1, (2,3,4,5,6))$
$S^2 = (12, (3,4,5,6))$
$S^3 = (123, (4,5,6))$
$S^4 = (1234, (5,6))$

$C^i = 1234$ et $X^i = 5,6$

$1234 \cap \Gamma(5) = 1234 \cap 346 = 34 \neq 1234$
$1234 \cap \Gamma(5) = 1234 \cap 346 = 34$
$34 \cup 5 = 345$ est une clique maximale du graphe induit par les sommets $\{12345\}$
donc $S^{n+1} = (1234, 6) = 12346$
$1234 \cap \Gamma(6) = 1234 \cap 35 = 3 \neq 1234$
donc 1234 est une clique maximale du graphe induit par les sommets $\{123456\}$
$1234 \cap \Gamma(6) = 1234 \cap 35 = 3$
$3 \cup 6 = 36$ n'est pas une clique maximale

$C^i = 345$ et $X^i = 6$

$345 \cap \Gamma(6) = 345 \cap 35 = 35 \neq 1234$
$345 \cap \Gamma(6) = 345 \cap 35 = 35$
$35 \cup 6 = 356$ est une clique maximale du sous-graphe induit par les sommets $\{3456\}$

donc $S^{n+1} = (35, 6) = 356$

Donc les cliques maximales du graphe sont :{**1234**}, {**345**}, {**356**}.

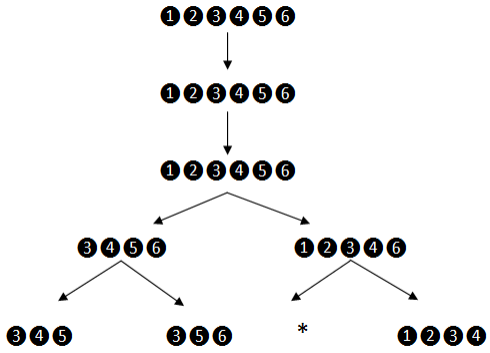


FIGURE 2.8 – Arbre obtenue aprés le déroulement de l'algorithme de Tsukiyama et al sur le graphe 3

2.7 Algorithme de Dahlhaus

2.7.1 Principe

C'est un algorithme qui utilise une décomposition de graphe en sous-graphe pour générer les cliques maximales du graphe G.

Dans l'étape (1), on décompose notre graphe G=(X,E) en deux sous-graphe G_1 et G_2 tel que $G_1 = \{x_1, ..., x_{[n/2]}\}$ et $G_2 = \{x_{[n/2]}, .., x_n\}$.
Dans l'étape (2), on définit U et W tel que
$U = \{$cliques maximales de $G_1\}$ et $W = \{$cliques maximales de $G_2\}$.
Dans l'étape (3), pour tout $u \in U, v \in W$, on cherchera $D_{u,v}$ tel que $D_{u,v} := \{C \subseteq u \cup v$; C clique maximale restreinte à $u \cup v\}$
En dernière étape, On définit $E_{u,v}$ qui est la réunion de $D_{u,v} \forall u \in U, \forall v \in W$ est l'ensemble de toutes les cliques maximales de G

La procédure $D_{u,v}$:
$D_{u,v}$ est l'ensemble des sous-graphes complets et maximaux de G restreint à $u \cup v$.
Les ensembles complets u et v sont disjoints. Tout sous-graphe complet maximal de G restreint à $u \cup v$ correspond à un sous-graphe biparti complet maximale d'un graphe biparti $(u \cup v, E^{'})$ où $E^{'}$ est l'ensemble des *arêtes*de E qui relie tout sommet de u à un sommet de v.

Soit A un sous-ensemble de U, Alors

$$A_2 := \{x \in A; \forall y \in v \{y, x\} \in E\}$$
$$A_1 := \{y \in v; \forall x \in A_2 \{y, x\} \in \text{É}\}$$

Remarque 2.2 [4] *$A_1 \cup A_2$ forme un sous-graphe biparti complet maximale et tous les sous graphes bipartis complets sont de cette forme.*

2.7.2 L'algorithme

Algorithme 5 Algorithme de Dahlhaus

Entrée : G=(X,E), $X = (x_1, ..., x_n)$.
Sortie : CLIQUE G=(X,E).
Procédure CLIQUE(X, E), (ensemble de cliques maximales de G=(X,E))
Si $|X| = 1$ **alors** $CLIQUE(X, E) := (X)$**Sinon**
début
Construire G_1 un sous-graphe de G induit par $x_1, ..., x_{[n/2]}$
Construire G_2 un sous-graphe de G induit par $x_{[n/2]}, ..., x_{[n]}$
U := CLIQUE (G_1)(= ensemble de cliques max de G_1)
W := CLIQUE (G_2)(= ensemble de cliques max de G_2)
Pour tout $u \in U$,$v \in W$ **faire**
début
Procédure COM-MAX$(D_{u,v})$
$D_{u,v} := \{C \subseteq u \cup v$;C est complète et maximale dans $G/u \cup v\}$
$E_{u,v}$:=$C \in D_{u,v} : C$ est une clique maximale dans G
fin
CLIQUE(C) :=$\cup_{u \in U v \in W} E_{u,v}$
fin
Procédure CLIQUE

2.7.3 Complexité

Cet algorithme de génération de cliques maximales s'exécute en $O((n.m)|C|)$ et espace mémoire $O(n + m)$.

2.7.4 Exemple

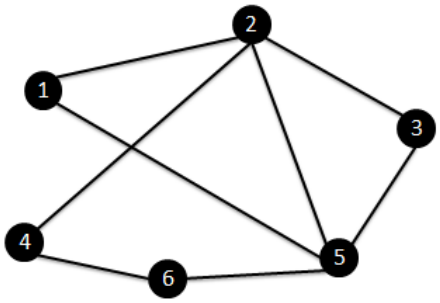


FIGURE 2.9 – Le graphe 5

$X = \{1,2,3,4,5,6\}$
$G_1 = \{4,5,6\}$
$G_2 = \{1,2,3\}$

U = { clique maximale de G_1} = $\{46,56\}$

FIGURE 2.10 – Le sous-graphe G_1

W = { clique maximale de G_2} = $\{12,23\}$

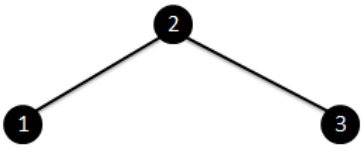


FIGURE 2.11 – Le sous graphe G_2

A sous-ensemble de U

$A_2 = \{x \in A, \forall y \in v : \{y,x\} \in E\}$

$A_1 = \{y \in v, \forall x \in A_2 : \{y,x\} \in E^{'}\}$

$A = \{4,6\}, v = \{1,2\}$

$x = 4 : \{14\} \notin E$ et $\{24\} \in E$

$x = 6 : \{16\} \notin E$ et $\{26\} \notin E$

donc $A_2 = \{\varnothing\}$

$A_2 = \{\varnothing\}$, $v = \{1,2\}$

$y = \varnothing$

donc $A_1 = \{\varnothing\}$

$A = \{4,6\}, v = \{2,3\}$

$x = 4 : \{24\} \in E$ et $\{34\} \notin E$

$x = 6 : \{26\} \notin E$ et $\{36\} \notin E$

donc $A_2 = \{\varnothing\}$

$A_2 = \{\varnothing\}, v = \{2,3\}$

$y = \varnothing$

donc $A_1 = \{\varnothing\}$

$A = \{5,6\}, v = \{1,2\}$

$x = 5 : \{15\} \in E$ et $\{25\} \in E$ donc $\{5\} \in A_2$

$x = 6 : \{16\} \notin E$ et $\{26\} \notin E$

donc $A_2 = \{5\}$

$A_2 = \{5\}, v = \{1,2\}$

$y = 1 : \{15\} \in E^{'}$ et $\{1\} \in E^{'}$ donc $\{1\} \in A_1$

$y = 2 : \{25\} \in E^{'}$ et $\{2\} \in E^{'}$ donc $\{2\} \in A_1$

donc $A_1 = \{1,2\}$

$A_1 \cup A_2 = \{\mathbf{1,2,5}\}$ clique maximale du graphe G .

$A = \{5,6\}, v = \{2,3\}$

$x = 5 : \{25\} \in E$ et $\{35\} \in E$ donc $\{5\} \in A_2$

$x = 6 : \{26\} \notin E$ et $\{36\} \notin E$

donc $A_2 = \{5\}$

$A_2 = \{5\}, v = \{2,3\}$

$y = 2 : \{25\} \in E^{'}$ et $\{2\} \in A_1$ donc $\{2\} \in A_1$

$y = 3 : \{35\} \in E^{'}$ et $\{3\} \in A_1$ donc $\{3\} \in A_1$

donc $A_1 = \{2,3\}$

$A_1 \cup A_2 = \{\mathbf{2,3,5}\}$ clique maximale du graphe G

les cliques maximales du graphe G sont $\{\mathbf{1,2,5}\}$ et $\{\mathbf{2,3,5}\}$

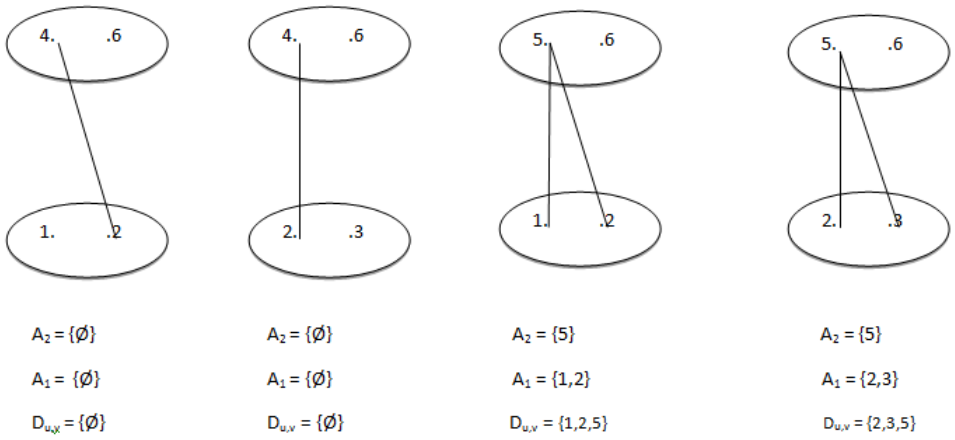


FIGURE 2.12 –

Remarques 2.1 [4] *Le but de cet algorithme est de générer toutes les cliques maximales du graphe G. Malheureusement, on trouve certains problèmes qu'on va citer comme suit :*

- *La façon de partager le graphe G en deux sous graphe G_1 et G_2, ne génère pas toutes les cliques maximales du graphe de départ, existe-il une manière de construire les deux sous graphes pour générer toutes les cliques maximales de G ?*

- *Si la taille n est grande, quand on construit les sous-graphe G_1 et G_2 pour générer toutes les cliques maximales de G, est-ce que les cliques maximales de G_1 et G_2 sont les cliques maximales de G ?*

2.8 Algorithme D'énumération

Cet algorithme combine entre l'algorithme de johnson et al pour déterminer la première clique maximale suivant l'ordre lexicographique, et la procédure All children pour générer toutes les cliques maximales du graphe G suivant l'ordre lexicographique.

2.8.1 Définition

Soit C une clique, et $\mathbf{C}(C)$ la clique qui contient la clique C.
C_1 la première clique maximale suivant l'ordre lexicographique. pour une clique maximale $C(\neq C_1)$, on définit un parent de C, $P(C)$ par : $\mathbf{C}(C_{\leq i-1})$, tel que i l'indice maximum qui satisfait : $\mathbf{C}(C_{\leq i-1}) \neq C$.
L'indice i est appelé l'indice parent, dénoté par $i(C)$.

Notons que : $C \neq \mathbf{C}(C_{\leq 0})$ vérifie par $C \neq C_1$, et comme $P(C)$ est grand lexicographiquement que C, la relation binaire parent-fils sur les cliques maximales est acyclique, et crée un arbre de racine C_1.

Lemme 2.1.[4] La relation parent-fils construit un arbre de racine C_1.

On appelle cet arbre, un arbre d'énumération de cliques maximales.

pour une clique maximale C et un indice i, on définit $C[i] = C((C_i \cap adj(x_i)) \cup \{x_i\})$.

Lemme 2.2.[4] Soient C et $C^{'}$ des cliques maximales dans G . Alors $C^{'}$ est le fils de C si seulement si $C^{'} = C[i]$, vérifie pour un certain i tel que :

a. $x_i \notin C$.

b. $i > i(C)$.

c. $C[i]_{\leq i-1} = C_{\leq i} adj(x_i)$.

d. $C_{\leq i} = \mathbf{C}(C_i \cap (x_i))_i$.

Si un indice i satisfait $\mathbf{a} \rightsquigarrow \mathbf{d}$, alors i est l'indice parent de $C[i]$.

Lemme2.3.[4] Soit C une clique maximale dans G.

Alors l'indice i satisfait (**c**) du **lemme 2.2** si seulement s'il n'existe pas un indice j qui satisfait les conditions suivantes :

c-1. $j < i$.

c-2. $x_j \notin C_{\leq i} \cap adj(x_i)$.

c-3. x_j est adjacents à tous les sommets dans $C_{\leq i} \cap adj(x_i) \cup \{x_i\}$.

Lemme 2.4.[4] Soit C une clique maximale dans G. Soit C une clique maximale dans G.
Soit C une clique maximale dans G.

d-1. $j < i$.

d-2. $x_i \notin C$.

d-3. x_j est adjacent à tous les sommets dans $C_{\leq i}$.

d-4. x_j est adjacent à tous les sommets dans $C_{\leq i} \cap adj(x_i)$.

2.8.2 Principe

Il s'agit d'un algorithme qui utilise l'approche de parcours lexicographique, basé sur l'ordre total des sommets. L'idée principal est de générer une clique maximale en appliquant All children sachant que la première clique est donnée.

Dans **l'étape (1**), on prend C_1 la première clique suivant l'ordre lexicographique en appliquant l'algorithme (2) de johnson. Cette clique est une clique maximale de G.

L'étape (2), consiste à générer à partir de C := min Q un ensemble de cliques maximales supérieurs à C, si ces cliques n'appartiennent pas déjà à Q, on les insère.

La génération se fait de la manière suivante : On applique la procédure All children de C, tel que C est une clique maximale de G.
On calcule d'abord l'ensemble des indices I vérifiant les conditions $(a) \rightsquigarrow (b)$ du lemme 2.2 pour chaque $i \in I$, si $C[i] = C(C_{\leq i} \cap adj(i) \cup \{i\})$ est une clique maximale du graphe induit par les sommets $1, ..., j$, on la complète par les sommets supérieurs à j pour obtenir la première clique maximale du graphe G qui la contient. Alors, lorsque $Q = \varnothing$ toutes les cliques maximales ont été généré.

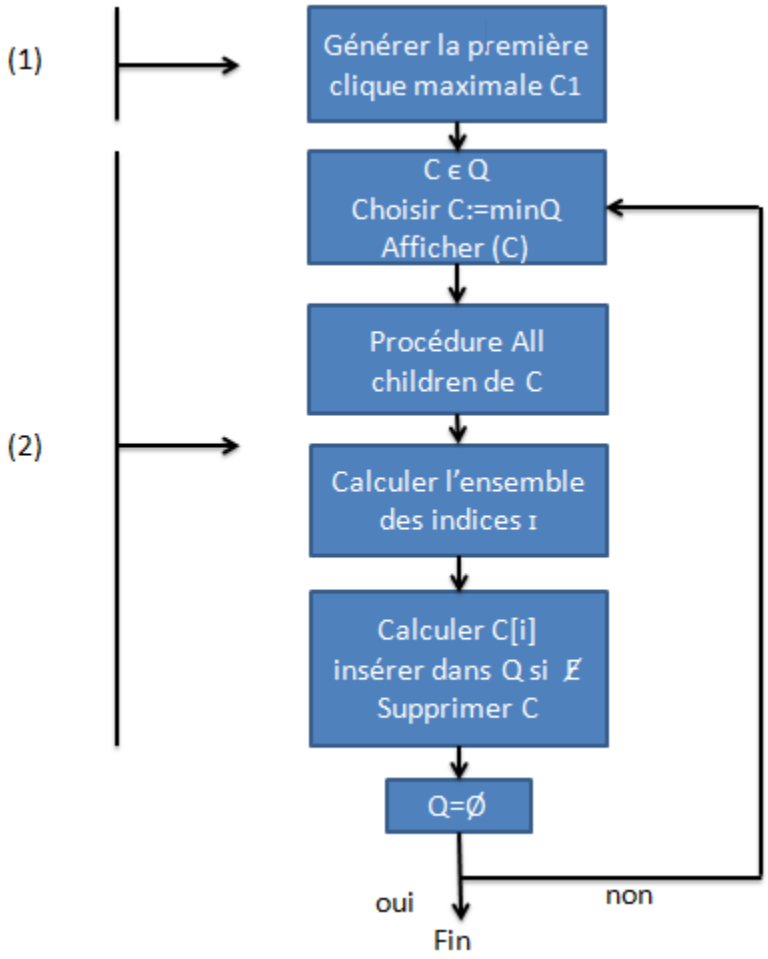


FIGURE 2.13 – Schéma représentant les étapes de l'algorithme

2.8.3 L'algorithme

Algorithme 6 Algorithme

Entrée : Un graphe G = (X,E) non orienté.
Sortie : Génère toutes les cliques maximales de G .

Début
Soit C_1 la première clique de G.
insérer C_1 dans Q
tant que $Q \neq \emptyset$ **faire**
début
$C_1 := minQ$
Afficher (C_1)
fin
Procédure All children C_1/C_1 est une clique maximale de G
début
Calculer l'ensemble I de tous les indices vérifiant les conditions de $(a) \rightsquigarrow (d)$ du lemme 2.2
début
pour tout $i \in I$ **faire**
Calculer C[i]
fin
début
Soit $C^{'}$ une clique maximale
Si $C^{'} \notin Q$ alors insérer $C^{'}$ dans Q et supprimer C_1
fin
fin
fin //de la procédure All chidren
fin // tant que
Fin

2.8.4 Exemple

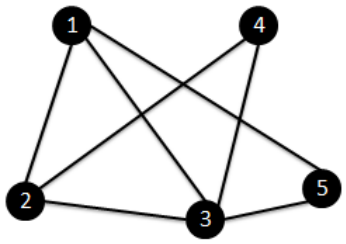


FIGURE 2.14 – Le graphe 6

L'étape 1 : Trouver la première clique maximale suivant l'ordre lexicographique avec l'algorithme 2 de johnson et al

$C = \{1\}$

pour i = 2 à 6 faire

i = 2 : $\Gamma(2) \cap \{1\} = \{1,3,4\} \cap \{1\} = \{1\}$

Donc $C = \{1\} + \{2\} = \{1,2\}$

i = 3 : $\Gamma(3) \cap \{1,2\} = \{1,2,4,5\} \cap \{1,2\} = \{1,2\}$

Donc $C = \{1,2\} + \{3\} = \{1,2,3\}$

i = 4 : $\Gamma(4) \cap \{1,2,3\} = \{2,3\} \cap \{1,2,3\} = \{2,3\} \neq \{1,2,3\}$

i = 5 : $\Gamma(5) \cap \{1,2,3\} = \{1,3\} \cap \{1,2,3\} = \{1,3\} \neq \{1,2,3\}$

Donc la première clique maximale dans l'ordre lexicographique est $C_1 = \{\mathbf{1,2,3}\}$

$Q = \{1,2,3\}$

$C_1 := minQ \Rightarrow C_1 = \{1,2,3\}$, Afficher (C_1).

L'étape 2 : La procédure All children de C_1.

On va calculer l'ensemble des indices I tel que $I = I_a \cap I_b \cap I_c \cap I_d$.

a. $x_i \notin C_1$.

I_a l'ensemble des indices vérifiant la condition a. du lemme 2.2 . Donc $I_a = \{\mathbf{4,5}\}$

b. $i > i(C)$ $\quad \mathbf{C}(C_{i-1}) \neq C_1$

i = 5 : $C_{1\leq 4} = \{1,2,3\} \cap \{1,2,3,4,5\} = \{1,2,3\}$

$\mathbf{C}(C_{1\leq 4}) = \{1,2,3\}$

i = 4 : $C_{1\leq 3} = \{1,2,3\} \cap \{1,2,3\}\} = \{1,2,3\}$

$\mathbf{C}(C_{1\leq 3}) = \{1,2,3\}$

i = 3 : $C_{1\leq 2} = \{1,2,3\} \cap \{1,2\} = \{1,2\}$

$\mathbf{C}(C_{1\leq 2}) = \{1,2,3\}$

i = 2 : $C_{1\leq 1} = \{1,2,3\} \cap \{1\} = \{1\}$

$\mathbf{C}(C_{1\leq 1}) = \{1,3,5\} \neq C_1$

Donc l'indice parent $i(C_1) = \mathbf{2}$

Et $I_b = \{\mathbf{3,4,5}\}$

c. $I_c = \{$ i tel qu'il n'existe pas un j qui vérifie $(c-1) \wedge (c-2) \wedge (c-3)\}$

c-1. $j < i$.

$$Q_{(c-1)} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

c-2. $x_j \notin C_{1\leq i} \cap \Gamma(x_i)$

i $= 1 : C_{1\leq 1} \cap \Gamma(1) = \{1,2,3\} \cap \{1\} \cap \{2,3,5\} = \varnothing$

i $= 2 : C_{1\leq 2} \cap \Gamma(2) = \{1,2,3\} \cap \{1,2\}\{1,3,4\} = \{1\}$

i $= 3 : C_{1\leq 3} \cap \Gamma(3) = \{1,2,3\} \cap \{1,2,3\} \cap \{1,2,4,5\} = \{1,2\}$

i $= 4 : C_{1\leq 4} \cap \Gamma(4) = \{1,2,3\} \cap \{1,2,3,4\} \cap \{2,3\} = \{2,3\}$

i $= 5 : C_{1\leq 5} \cap \Gamma(5) = \{1,2,3\} \cap \{1,2,3,4,5\} \cap \{1,3\} = \{1,3\}$

$$Q_{(c-2)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

c-3. x_j est adjacents à tous les sommets dans $C_{\leq i} \cap \Gamma(x_i) \cup \{x_i\}$

Soit Q la matrice où la $i^{éme}$ colonne de est $x((C_{\leq i} \cap \Gamma(x_i) \cup \{x_i\})$

i $= 1 : C_{1\leq 1} \cap \Gamma(1) \cup \{1\} = \{1\}$

i $= 2 : C_{1\leq 2} \cap \Gamma(2) \cup \{2\} = \{1,2\}$

i $= 3 : C_{1\leq 3} \cap \Gamma(3) \cup \{3\} = \{1,2,3\}$

i $= 4 : C_{1\leq 4} \cap \Gamma(4) \cup \{4\} = \{2,3,4\}$

i $= 5 : C_{1\leq 5} \cap \Gamma(5) \cup \{5\} = \{1,3,5\}$

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

A la matrice d'adjacence de G.

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

$$Q \cdot A = \begin{pmatrix} 3 & 2 & 3 & 2 & 2 \\ 2 & 2 & 2 & 2 & 1 \\ 2 & 2 & 2 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

A partir des matrices Q_{c-1}, Q_{c-2} et Q_{c-3} on aura :

i = 1 : $j_{(c-1)} = \{\varnothing\}$

$j_{(c-2)} = \{1,2,3,4,5\}$

$j_{(c-3)} = \{\varnothing\}$

Donc il n'existe pas j qui vérifie $(c-1) \wedge (c-2) \wedge (c-3) \Rightarrow i = 1$ vérifie c.

i = 2 : $j_{(c-1)} = \{1\}$

$j_{(c-2)} = \{2,3,4,5\}$

$j_{(c-3)} = \{1,2,3,4,5\}$

Donc n'existe un j qui vérifie $(c-1) \wedge (c-2) \wedge (c-3) \Rightarrow i = 2$ vérifie c.

i = 3 : $j_{(c-1)} = \{1,2\}$

$j_{(c-2)} = \{3,4,5\}$

$j_{(c-3)} = \{1,2,3,4,5\}$

Donc n'existe un j qui vérifie $(c-1) \wedge (c-2) \wedge (c-3) \Rightarrow i = 3$ vérifie c.

i = 4 : $j_{(c-1)} = \{1,2,3\}$

$j_{(c-2)} = \{1,4,5\}$

$j_{(c-3)} = \{\varnothing\}$

Donc n'existe un j qui vérifie $(c-1) \wedge (c-2) \wedge (c-3) \Rightarrow i = 4$ vérifie c.

i = 5 : $j_{(c-1)} = \{1,2,3,4\}$

$j_{(c-2)} = \{2,4,5\}$

$j_{(c-3)} = \{1,3\}$

Donc n'existe un j qui vérifie $(c-1) \wedge (c-2) \wedge (c-3) \Rightarrow i = 5$ vérifie c.

$I_c = \{\mathbf{1,2,3,4,5}\}$

d.

d-1. $j < i$ On a $Q_{d-1} = Q_{c-1}$

d-2. $x_j \notin C_1$ $j_{(d-2)} = \{4,5\}$

d-3. Soit Q la matrice où la $i^{éme}$ colonne est $x(C_{\leq i})$.

On va calculer Q_{d-3}

i $= 1 : C_{1\leq 1} = \{1,2,3\} \cap \{1\} = \{1\}$

i $= 2 : C_{1\leq 2} = \{1,2,3\} \cap \{1,2\} = \{1,2\}$

i $= 3 : C_{1\leq 3} = \{1,2,3\} \cap \{1,2,3\} = \{1,2,3\}$

i $= 4 : C_{1\leq 4} = \{1,2,3\} \cap \{1,2,3,4\} = \{1,2,3\}$

i $= 5 : C_{1\leq 5} = \{1,2,3\} \cap \{1,2,3,4,5\} = \{1,2,3\}$

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

$$Q \cdot A = \begin{pmatrix} 3 & 3 & 4 & 2 & 2 \\ 3 & 2 & 3 & 2 & 1 \\ 2 & 2 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

d-4. Soit Q la matrice où $i^{éme}$ colonne est $x(C_{\leq i} \cap \Gamma(x_i))$

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

$$Q \cdot A = \begin{pmatrix} 3 & 1 & 2 & 2 & 1 \\ 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

i $= 1 : j_{(d-1)} = \{\varnothing\}$

$j_{(d-2)} = \{4,5\}$

$j_{(d-3)} = \{\varnothing\}$

$j_{(d-4)} = \{\varnothing\}$

Donc n'existe un j qui vérifie $(d-1) \wedge (d-2) \wedge (d-3) \wedge (d-4) \Rightarrow i = 1$ vérifie c.

i = 2 : $j_{(d-1)} = \{1\}$

$j_{(d-2)} = \{4,5\}$

$j_{(d-3)} = \{2,4\}$

$j_{(d-4)} = \{1,3,4,5\}$

Donc n'existe un j qui vérifie $(d-1) \wedge (d-2) \wedge (d-3) \wedge (d-4) \Rightarrow i = 2$ vérifie c.

i = 3 : $j_{(d-1)} = \{1,2\}$

$j_{(d-2)} = \{4,5\}$

$j_{(d-3)} = \{1,2,3,4,5\}$

$j_{(d-4)} = \{1,2,3\}$

Donc n'existe un j qui vérifie $(d-1) \wedge (d-2) \wedge (d-3) \wedge (d-4) \Rightarrow i = 3$ vérifie c.

i = 4 : $j_{(d-1)} = \{1,2,3\}$

$j_{(d-2)} = \{4,5\}$

$j_{(d-3)} = \{\varnothing\}$

$j_{(d-4)} = \{\varnothing\}$

Donc n'existe un j qui vérifie $(d-1) \wedge (d-2) \wedge (d-3) \wedge (d-4) \Rightarrow i = 4$ vérifie c.

i = 5 : $j_{(d-1)} = \{1,2,3,4\}$

$j_{(d-2)} = \{4,5\}$

$j_{(d-3)} = \{\varnothing\}$

$j_{(d-4)} = \{\varnothing\}$

Donc n'existe un j qui vérifie $(d-1) \wedge (d-2) \wedge (d-3) \wedge (d-4) \Rightarrow i = 5$ vérifie c.

$I_d = \{\mathbf{1,2,3,4,5}\}$

Par conséquent $I = I_a \cap I_b \cap I_c \cap I_d = \{4,5\} \cap \{3,4,5\} \cap \{1,2,3,4,5\} \cap \{1,2,3,4,5\} = \{\ \mathbf{4,5}\}$

$C[i] = \mathbf{C}((C_{\leq i} \cap \Gamma(x_i)) \cup \{x_i\})$

$C[4] = \mathbf{C}((C_{1\leq 4} \cap \Gamma(4)) \cup \{4\}) = \{\mathbf{2,3,4}\}$

$C[5] = \mathbf{C}((C_{1\leq 5} \cap \Gamma(5)) \cup \{5\}) = \{\mathbf{1,3,5}\}$

$Q = \{2,3,4\}, \{1,3,5\}$

$C_1 := minQ = \{1,3,5\}$

a. $x_i \notin C_1 \quad I_a = \{2,4\}$

b. $i > i(C) \quad \mathbf{C}(C_{1\leq i-1}) \neq C_1$

i = 5 :$C_{1\leq 4} = \{1,3,5\} \cap \{1,2,3,4\} = \{1,3\}$

$\mathbf{C}(C_{1\leq 4}) = \{1,2,3\} \neq \{1,3,5\}$

Donc l'indice parent $i(C_1) = 5$ et $I_b = \{\varnothing\}$, par suit $I_a \cap I_b = \{\varnothing\}$

$Q = \{2,3,4\}$, $C_1 := minQ = \{2,3,4\}$

a. $x_i \notin C_1 \quad I_a = \{1,5\}$

b. $i > i(C) \quad \mathbf{C}(C_{1\leq i-1}) \neq C_1$
$I_a \cap I_b = \{\varnothing\}$

$Q = \{\varnothing\}$ qui est la condition d'arrêt.

En conséquent, toutes les cliques maximales sont générées et suivant l'ordre lexicographique : **C = {1,2,3}** , **{1,3,5}** , **{2,3,4}**

FIGURE 2.15 – Arbre de cliques

Tableau comparatif

L'algorithme	**Complexité**	**Délai**(temps)	**Espace mémoire**	**Technique utilisée**
Johnson et al	$O(n^3)$	polynomial	exponentiel	parcours lexicographique
Allclique	$O(n^{Log_2(n)+2})$	exponentiel	linéaire	retour arrière
Tsukiyama et al	$O(n \cdot m)$	polynomial	linéaire	retour arrière
Dahlhaus	$O((n \cdot m)\|C\|)$	polynomial	linéaire	/
Enumération	/	Polynomial	linéaire	parcours lexicographique

Description

1. L'algorithme Johnson génère toutes les cliques maximales suivant l'ordre lexicographique, et avec un délai polynomial. Mais malheureusement le troisième critère l'espace mémoire n'est pas vérifié qui est exponentiel, car à chaque itération toutes les cliques maximales insérées dans la file Q sont comparables à une nouvelle clique maximale, ce qui demande un espace mémoire très grand.
2. L'algorithme Allclique utilise la technique du retour arrière pour générer toutes les cliques maximales et avec un délai polynomial et espace mémoire linéaire. L'ordre lexicographique sera vérifié si on respecte l'ordre total défini sur les sommets.

3. L'algorithme Tsukiyama génère toutes les cliques maximales avec un ordre quelque conque, malgré qu'il vérifie le critère d'espace mémoire qui est linéaire, et un délai polynomial.
4. l'algorithme Dahlhaus ne génère pas toutes les cliques maximales du graphe G,cela dépend de la partition des sous-graphes.
5. Le dernier algorithme combine entre l'algorithme johnson pour déterminer la première clique maximale, et la procédure All children pour générer toutes les cliques maximales suivant l'ordre lexicographique.

Chapitre 3

Application

3.1 Introduction

Dans ce chapitre, nous nous intéressons à programmer l'algorithme de johnson qui génère toutes les cliques maximales d'un graphe non orienté.
Dans cette perspective nous cherchons à trouver un codage qui permet de générer ces dernières. Sans utiliser des méthodes combinatoires, ce problème sera résolu et programmable au sens informatique, en utilisant comme outil un langage C++.

3.2 Définitions de base

Langage de programmation C++
L'ordinateur est une machine étonnante et complexe, à la base ne comprend qu'un langage très simple constitué de 0 et 1. Tous les langages de programmation ont le même but, ils permettent de parler à l'ordinateur plus simplement qu'en binaire. Voici comment cela fonctionne :

1. Écrire des instructions pour l'ordinateur dans un langage de programmation(par exemple C++).
2. Traduire les instructions en binaire grâce à un compilateur.
3. Lire le binaire et faire ce qui est demandé.

FIGURE 3.1 – schéma représentant les étapes de compilation

Langage de haut et de bas niveau

Il existe des centaines de langage de programmation, certains sont populaires que d'autres, par exemple le java et C++ occupent régulièrement le haut du classement. Et on peut classer ces langage en deux niveaux :

- **Langage de haut niveau** : est un langage assez éloigné du binaire(et donc du fonctionnement de la machine), il permet généralement de développer de façon plus souple et rapide.
- **Langage de bas niveau** : est un langage plus proche du fonctionnement de la machine, il demande généralement un peu plus d'effort, mais il donne plus de contrôle sur le problème. Java est haut niveau par rapport au C++, et le C++ est haut niveau par rapport à l'assembleur.

Le C++ est un langage de programmation très répandu et rapide. C'est une évolution du langage C car il offre en particulier la possibilité de programmer en orienté objet, qui est une technique de programmation puissante.

Programmes

Permettent de réaliser toute sorte d'action sur un ordinateur : navigation sur le web, rédaction de texte, manipulation des fichiers.

Code source

Ensemble d'instructions écrites dans un ordinateur avec un langage de programmation, qui réalise un programme. Et ce code doit être traduit en binaire par un outil appelé compilateur pour qu'il soit possible de lancer le programme. L'ordinateur ne comprend en effet que le binaire.

Code binaire

La machine ne comprend que le code binaire, qui est constitué de 0 et 1.

Editeur de texte

il permet d'écrire le code source du programme en langage C++. En théorie un logiciel comme le Bloc-note sous Windows, ou "vi" sous linux. L'idéal c'est d'avoir un éditeur de texte intelligent qui colore tout seul le code, ce qui vous permet de vous repérer dedans plus facilement. C'est pour ça les programmeurs n'utilisent pas le Bloc-note.

Compilateur
Il transforme("compile") le code source en code binaire.
Débugger
Il aide à traquer les erreurs dans un programme.
Objet
C'est là qu'intervient la programmation orienté objet(en abrégé P.O.O), fondée justement sur le concept d'objet. Un projet est constitué de plusieurs fichiers de code source : des fichiers.cpp, .h,les images du programme, etc. Qui forme une association des données et des procédures (qu'on appelle méthodes) agissant sur ces données.

Méthodes + Données = Objet

IDE(EDI)
En français, "Environnement de développement intégré", il rassemble tous les fichiers d'un projet au sein d'une même interface, pour avoir l'accès à tous les éléments de programme à portée de clic. Il existe plusieurs IDE en version gratuite et payante, parmi les IDE plus connus :

- **Code : :blocks** : il est gratuit et fonctionne sur la plupart des systèmes d'exploitation : Windows, Mac et Linux.
- **Visual C++** : il existe à la base en version payante, mais heureusement il existe une version gratuite intitulée **Visual C++ Express**, il y a peu de différence avec la version payante. Il est très complet et possède puissant module de correction des erreurs (débuggage), il fonctionne sous Windows uniquement.
- **XCode** : généralement fourni sur un CD d'installation Mac OS X. C'est IDE très apprécié par tous ceux qui font de la programmation sur le Mac, il fonctionne sous Mac OS X uniquement.

3.3 Présentation de langage

Très tôt, les concepts de la programmation orienté objet, ont donné naissance à de nouveaux langages dits "orienté objet" tels que Smalltalk, Simula, Eiffel ou, plus récemment, java. Le langage C++, quant à lui, a été conçu suivant une démarche hybride. En effet, Bjarne Stroustrup, son créateur, a cherché à adjoindre à un langage structuré existant(C), un certain nombre de spécificités lui permettant d'appliquer les concepts de P.O.O. Dans une certain mesure, il a permis à des programmeurs C d'effectuer une transition en douceur de la programmation structurée vers la P.O.O. De sa conception jusqu'à sa normalisation, le langage C++ a quelque peu évolué. Initialement, un certain nombre de publications de ATT ont servi de référence du langage. Les dernières en date sont : la version 2.0 en 1989, les

versions 2.1 et 3 en 1991. C'est cette dernière qui a servi de base au travail du comité ANSI qui, sans la remettre en cause, l'a enrichie de quelques extensions et surtout de composants standard originaux se présentant sous forme de fonctions et de classes génériques qu'on désigne souvent par le sigle S.T.L(Standard Template Library). La norme définitive de C++ a été publiée par L'ANSI en juillet 1998.

Les avantages de C++

- Il est **très répandu**, il fait partie des langages de programmation les plus utilisés sur la planète. On trouve donc beaucoup de documentation sur internet et on peut facilement avoir de l'aide sur les forums.
- Il est **rapide**, ce qui en fait un langage de choix pour les applications critiques qui ont besoin de performances. C'est en particulier le cas des jeux vidéo, mais aussi des outils financiers ou de certains programmes militaires qui doivent fonctionner en temps réel.
- Il est **portable**, un même code source peut théoriquement être transformer sans problème en exécutable sous Windows, Mac OS et Linux, n'a pas besoin de réécrire le code pour d'autres plate-formes.
- Il existe de **nombreuses bibliothèques** pour le C++, sont des extensions pour le langage, en combinant le C++ avec de bonnes bibliothèques, on peut créer des programmes 3D, réseaux, audio, etc.

Les outils nécessaire pour programmer

- **Editeur de texte** : pour écrire le code source du programme.
- **Compilateur** : pour traduire le code source en code binaire.
- **Débugger** : pour corriger les erreurs.

3.4 Exemple d'application

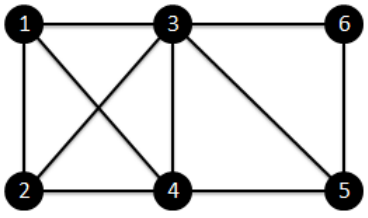


FIGURE 3.2 – Graphe d'application

Étape 1 : On fait la représentation du graphe avec sa matrice d'adjacence, si un sommet i est relié à un autre sommet j par une arrête, on tape 1 sinon 0 .
La matrice d'adjacence associée apparait dans l'image suivante :

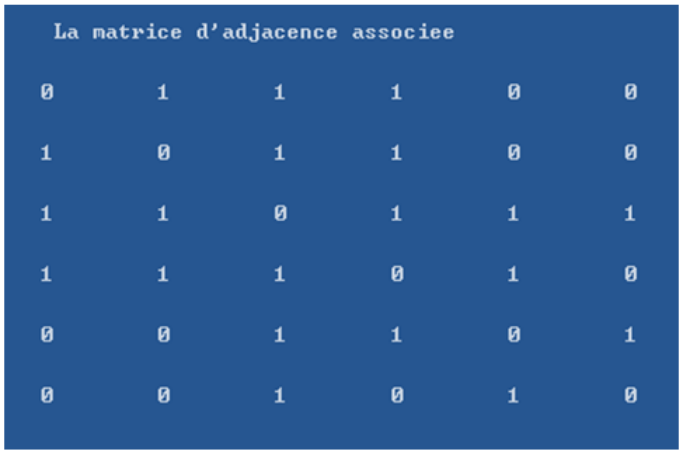


FIGURE 3.3 – Matrice d'adjacence

Étape 2 : La première partie de l'algorithme consiste à trouver la première clique maximale suivant l'ordre lexicographique, qui est {**1,2,3,4**}. Le résultat de cette dernière apparait dans l'image suivante :

FIGURE 3.4 – La première clique maximale

La deuxième partie c'est pour générer toutes les cliques maximales du graphe G.

FIGURE 3.5 – Les cliques maximales

Toutes les cliques maximales sont classées suivant l'ordre lexicographique :

{1,2,3,4} , **{3,4,5}** , **{3,5,6}**

Conclusion générale

De nos jours, de nombreux laboratoires de recherches, à travers le monde, travaillent sur l'amélioration de ces algorithmes présentés dans ce travail. Certains cherchent à trouver ou emploient des heuristiques afin d'essayer d'améliorer la complexité et le rendement de ces algorithmes ; d'autres, cherchent des algorithmes avec de meilleures complexités.
L'objectif de cette étude est de faire connaitre d'une manière générale le concept des cliques maximales dans les graphes non orientés tout en mettant en évidence leurs principaux avantages dans la recherche des cliques maximales.
Ce travail nous a permis de dégager plusieurs perspectives qui nous semblent intéressantes à explorer dans les travaux futures. Parmi ces perspectives, nous citons :

- Étude et analyse des cliques maximales pour les différentes classes de graphes.
- Synthèse et étude des applications des cliques maximales dans différents domaines.

Bibliographie

[1] Jaques Carlier, Livre "Recherche Opérationnelle : Optimisation Combinatoire".

[2] A.Gherboudj, Thèse "Méthodes de résolutions de problèmes difficiles académique", université de CONSTANTINE 2 , 2013.

[3] O.Hammoutene, Mémoire "La génération des cliques maximales dans un graphe non orienté", UMMTO le Mai 2014.

[4] Louadj Kahina, Thèse "Génération des cliques maximales dans un graphe non orienté", UMMTO le 08/06/2005

[5] Oubakouk Lynda, Thèse "Génération de cliques maximales : Étude du graphe de transition", UMMTO en 2015.

[6] B.Sadi, B.Oukacha, K.Louadj, Article "Génération de cliques maximales d'un graphe", UMMTO

[7] C.Berge, Livre "graphes et hypergraphes"

[8] Philippe Gambette, Article "Cours d'algorithmique des graphes du MPRI", 23 janvier 2009.

[9] Claude Delannoy, Livre "Apprendre le C++".

[10] Mathieu Nebra et Matthieu Schaller, Livre " Programmez avec le langage C++".