REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE.
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE.

UNIVERSITE MOULOUD MAMMERI, TIZI-OUZOU
FACULTE DES SCIENCES
DEPARTEMENT MATHEMATIQUES

MÉMOIRE DE MASTER
en
MATHÉMATIQUES

Spécialité
Recherche Opérationnelle : Aide à la décision

Thème

Etude algorithmique sur les treillis et les cliques maximales

Présenté par :

Hadj Ali Bahia
Hamzi Nacira

Devant le jury d'examen composé de :

R.Kheffache	M.assist A	UMMTO	Présidente
F.Aklouche	M.assist A	UMMTO	Examinatrice
A.Belhadj	M.assist A	UMMTO	Rapporteur

soutenu : Octobre 2017

Remerciements

Avant tout propos, nous remercions «**Dieu** » le tout puissant qui nous a donné la sagesse et la santé pour faire ce modeste travail.

C'est avec un grand plaisir que nous exprimons nos gratitudes et nos sincères remerciements à notre promoteur : **Mr Belhadj Abdelaziz** pour son orientation et encadrement dans l'élaboration de ce mémoire de fin d'études, nos remerciements vont aussi à tous les membres du jury pour avoir accepter d'honorer par leur jugement notre travail, et pour leur participation au jury.

Toutes nos reconnaissances sont adressées à tous les enseignants qui nous ont suivi infatigablement durant tout notre cursus universitaire.

Nous tenons à exprimer tout au fond de nos cœurs les reconnaissances à nos familles qui nous ont offert toujours un appui sûr par leurs soutiens et leurs encouragements. Nos vifs remerciements vont également à tous ceux qui ont contribué de loin ou de prés à la réalisation de ce travail.

Dédicaces

Je tiens très respectueusement à dédier ce modeste travail de fin

D'étude à mes très chers parents, la source

D'amour et de tendresse, ainsi que pour

Leurs précieux conseils et leurs sacrifices, que

Dieu les protège et les entoure de sa bénédiction ;

Je le dédie également à mes chères sœurs : Lila, Hassina,

Fatiha, Zahia, Ghania et Fadila ainsi que mon frère Nadir.

Sans oublier tous mes amis(es), ainsi que tous ceux qui nous ont

Apporté leurs soutiens pour élaborer ce travail.

Bahia

Dédicaces

Je tiens très respectueusement à dédier ce modeste travail de fin

D'étude à mes très chers parents, la source

D'amour et de tendresse, ainsi que pour

Leurs précieux conseils et leurs sacrifices, que

Dieu les protège et les entoure de sa bénédiction ;

Je le dédie également à mes chères sœurs : Naima, Samira,

Salima, Zineb, ainsi que mes frères Naim, Said .

Sans oublier tous mes amis(es), ainsi que tous ceux qui nous ont

Apporté leurs soutiens pour élaborer ce travail.

Nacira

Table des matières

Table des figures

Liste des algorithmes

Liste des symboles

X	L'ensemble des sommets.
E	L'ensemble des arrêtes.
U	L'ensemble des arcs.
$\Gamma(x)$	L'ensemble des sommets adjacents au sommet x.
$\overline{\Gamma}(x)$	L'ensemble des sommets non adjacents au sommet x.
T	Un treillis.
P	Un ensemble ordonné ou ordre.
C	Une clique.
S^i	Une solution partielle.

Introduction générale

Depuis des années, beaucoup de gens bornent les mathématiques dans les opérations élémentaires comme l'addition, la multiplication, la soustraction et la division, mais en parallèle il y avait d'autres sciences qui se sont dérivées des mathématiques, on signale pour la théorie des graphes qui a été utilisée avant même d'avoir été structurée. Sa naissance remonte aux célèbres problèmes des ponts de Königsberg étudiés par Euler en 1736, ensuite elle a été la meilleure façon de traiter beaucoup de problèmes dans différents domaines (la chimie, la psychosociologie, l'économie, etc...).
Elle n'a pu prendre sa forme actuelle que grâce aux efforts de certains spécialistes de la Recherche opérationnelle et sous des nécessités pratiques.

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 y a une grande relation entre celle-ci et l'informatique. C'est à cause de cela qu'on est amené à un problème de la théorie des graphes qui consiste à étudier quelques algorithmes sur les treillis et d'autres sur la génération des cliques maximales et à la fin nous déduisons qu'il existe des relations entre les cliques maximales et un treillis.

Ce mémoire s'articule autour de cinq chapitres :
Le premier chapitre est constitué des définitions et orientations de base concernant les graphes et les ensembles ordonnés ainsi que l'algorithmique, afin de faciliter au lecteur la compréhension et la lecture du contenu de ce travail.

Le deuxième chapitre, résume les notions de base sur les treillis en général et quelques classes de treillis en particulier.

Quant au troisième chapitre, on fera une étude algorithmique de la génération et de la construction des treillis, en citant comme exemple deux algorithmes celui de L. Nourine et un autre de Nourine et Raynaud, en déroulant une application bien détaillée pour chacun d'eux.

Le quatrième chapitre, fera l'objet de recenser et d'analyser les algorithmes de génération des cliques maximales. Nous proposons l'algorithme de Johnson et Al qui utilise l'approche de parcours lexicographique basé sur l'ordre total des sommets, et celui de Tsukiyama qui utilise un espace polynomial. Un exemple illustratif accompagne chacun d'eux.

Enfin, dans le dernier chapitre on a présenté le langage de programmation C++ sur lequel on a appliqué l'algorithme de reconnaissance d'un treillis tout en illustrant les différentes étapes d'exécution de notre programme.

On terminera par une conclusion qui clôturera notre travail.

Chapitre 1

Définitions et concepts de base

1.1 Introduction

Les graphes sont des outils de modélisation de nombreuses situations et problèmes. C'est pourquoi leurs applications sont nombreuses et variées : en algèbre, en combinatoire, en informatique, en recherche opérationnelle, en cartographie et bien d'autre domaines. Voici quelques définitions essentielles des graphes et éléments de graphes dont on aura besoin.

1.2 Graphes

1.2.1 Définitions

- Un graphe G est un couple de deux ensembles finis et disjoints, il est de deux types :
  - **Graphe orienté** :il est défini par : Un ensemble $X=\{1, 2, 3, \ldots, n\}$ qui représente l'ensemble des sommets du graphe G. Une famille $U=\{u_1, u_2, \ldots, u_m\}$ l'ensemble des arcs de G. Un arc u est d'extrémité initiale x et d'extrémité terminale y, il sera noté $u=(x, y)$.

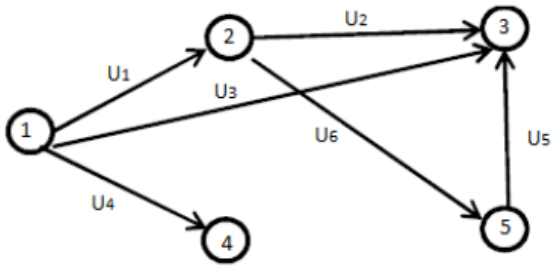


FIGURE 1.1 – Exemple d'un graphe orienté

  - **Graphe non-orienté** $G = (X, E)$ est déterminé par un ensemble $X = \{x_1, x_2, .., x_n\}$ dont les éléments sont appelés sommets et une famille $E = \{e_1, e_2, \ldots, e_m\}$ dont les éléments sont appelés les arêtes, comme le montre la figure suivante :

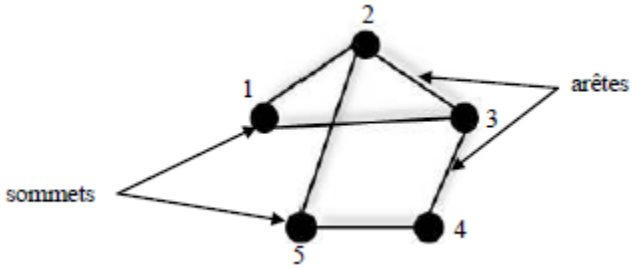


FIGURE 1.2 – Exemple de graphe non orienté d'ordre 5

- **L'ordre** d'un graphe, noté n, est le nombre de ses sommets (i.e. : $|X| = n$).

- La **taille** d'un graphe, noté m, est le nombre de ses arêtes (i.e. : $|E| = m$).

- **Sommet adjacent (ou voisin)** : Deux sommets sont adjacents s'ils sont reliés par une arête. On note par $\Gamma(x)=\{y \in X/xy \in E\}$ l'ensemble des sommets adjacents au sommet x.

- On appelle **degré** d'un sommet x de G, noté $d(x)$, le nombre de voisins que possède ce sommet (i.e :$d(x) = |\Gamma(x)|$).

- Un graphe dont tous les sommets ont le même degré est dit **régulier**.

- **Sommet pendant** : on appelle sommet pendant, un sommet qui n'est adjacent qu'à un seul sommet. (i.e :$|\Gamma(x)| = 1$).

- **Boucle** : Un arc reliant un sommet x à lui même.

- **Graphe simple** : est un graphe sans boucle et sans arêtes multiples. Tous les graphes que nous utiliserons dans ce travail sont simples.

- **Graphe complet** : un graphe $G=(X, E)$ est complet si toute paire de sommets de X est une arête de E.

- **Graphe complémentaire** : Un graphe $\overline{G} = (X, \overline{E})$ est dit complémentaire de $G = (X, E)$ s'il contient l'ensemble des sommets de G et non les arêtes de G.
(i.e :$\overline{E} = \{(i, j) \in X^2/j \notin E\}$).

- **Graphe partiel** : un graphe $G'=(X, E')/E' \subset E$ est dit graphe partiel de $G=(X, E)$ si les sommets sont les points de X et les arêtes sont ceux de E'. Autrement dit, on élimine de G les arêtes de $(E \setminus E')$.

- **Sous graphe** :Un sous graphe d'un graphe $G=(X, E)$ est un graphe $G_A=(A, E_A)$ où $A \subset X$, et $E_A=\{xy \in E/x \in A$ et $y \in A\}$.

- **Chaîne** : est une suite finie de sommets x_0,x_1,. . . ,x_p telle que $0 \leq i \leq p$, $(x_i, x_{i+1}) \in E$.

- Lorsque x_0=x_p, on dit que la chaîne est fermée et on l'appelle **cycle**.

- **Chemin** : un chemin de G=(X, U) est une séquence de sommets x_0,x_1,...,x_n telles que pour i= $0, 1, \ldots, n-1$, on ait $(x_i, x_{i+1}) \in U$.

- Si $x_0 = x_n$, on dit que le chemin est fermé et on l'appelle **Circuit**.
  La chaîne et le cycle sont des notions non orientées, le chemin et le circuit sont des notions orientées.

- **Clique** : Une clique est un sous-graphe complet (i.e : un ensemble de sommets deux-à-deux adjacents). Elle est dite maximale si elle n'est pas contenue dans une autre.

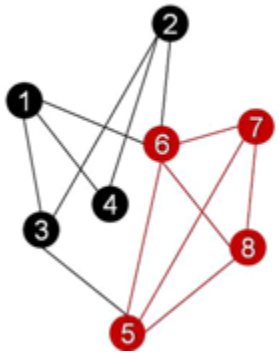


FIGURE 1.3 – Graphe $G = (X, E)$ ayant une clique maximale $C = \{5, 6, 7, 8\}$

- **Stable** : un ensemble $S \subset X$ est dit stable si deux sommets distincts de S ne sont jamais voisins, autrement dit : $\Gamma_G(S) \cap S = \emptyset$.

- **Graphe connexe** : $G = (X, E)$ est connexe si pour toute paire x, y deux sommets distincts, il existe une chaîne reliant ces deux sommets.
  $G = (X, U)$ est fortement connexe s'il existe un chemin de x à y et un chemin de y à x.

- **Arbre** : $G = (X, E)$ est un arbre s'il est connexe et sans cycle.

- **Ordre lexicographique** : Soit $\mu(G) = \{C_1, C_2, ..., C_n\}$ l'ensemble des cliques maximales d'un graphe G,On note par $\leq_{lex}$ l'ordre l'exicographique sur les cliques, C'est-à-dire $C_1 \leq_{lex} C_2 \ldots \leq_{lex} C_n$.
  On dit que C_1 est inférieure lexicographiquement à C_2 et on écrit $C_1 \leq_{lex} C_2$ si et seulement s'il existe un indice i tel que $C_1 \cap \{1, 2, \ldots, i-1\} = C_2 \cap \{1, 2, \ldots, i-1\}$ et $C_{1_i} \leq_{lex} C_{2_i}$

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

1.2.2 Représentation matricielle

Matrice d'adjacences :
On peut représenter un graphe orienté par une matrice d'adjacence. Il s'agit d'une matrice carrée $(n * n)$ où $(i; j)$ désigne l'intersection de la ligne i et de la colonne j.
Dans une matrice d'adjacence, les lignes et les colonnes représentent les sommets du graphe. Elle est définie comme suit :
Soit B une matrice carrée telle que $B = (b_{ij})\,;\, i, j = 1 \ldots n$ où

$$b_{ij} = \begin{cases} 1 & si \quad (x_i, x_j) \in U \\ 0 & sinon. \end{cases}$$

Cette matrice s'utilise également dans le cas de graphe non orienté, elle est alors symétrique.

Exemple 1.1. *Pour le graphe de la figure (1.1) la matrice d'adjacence est la suivante :*

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

Matrice d'incidence :
On peut représenter un graphe orienté par une matrice d'incidence de dimension $(n * m)$, où chaque ligne est associée à un sommet et chaque colonne à un arc. Ses éléments prennent les valeurs $(0, -1, 1)$. Celle-ci est définie comme suit :
Soit$A = (a_{ij})\,;\, i = 1 \ldots n, j = 1 \ldots m$
I : extrémité initiale de l'arc $u_{_j}$ et T : extrémité terminale de l'arc $u_{_j}$ où

$$a_{ij} = \begin{cases} +1 & si \quad x_i = I(u_j) \\ -1 & si \quad x_i = T(u_j) \\ 0 & sinon. \end{cases}$$

Exemple 1.2. *Pour le graphe de la figure (1.1) la matrice d'incidence est la suivante :*

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

1.2.3 Représentation par 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. Ce sont les listes d'adjacences.
Pour un graphe orienté, en donnant pour chacun de ses sommets la liste des sommets qu'on peut atteindre directement en suivant un arc (dans le sens de la flèche).

1.3 Ensemble ordonné

1.3.1 Définitions

- Une relation $\leq$ défini sur un ensemble X est appelée **relation d'ordre** si pour tout $x, y, z \in X$, on a les propriétés suivantes :
  1. Réflexivité :$x \leq x$;
  2. Antisymétrie :$x \leq y$ et $y \leq x \Rightarrow x = y$;
  3. Transitivité : $x \leq y$ et $y \leq z \Rightarrow x \leq z$;

- Un **ensemble ordonné** ou **ordre** P est un couple $(X, \leq_P)$ où X est un ensemble non vide et $\leq_P$ est une relation d'ordre définie sur X, (i.e réflexive, antisymétrique et transitive).

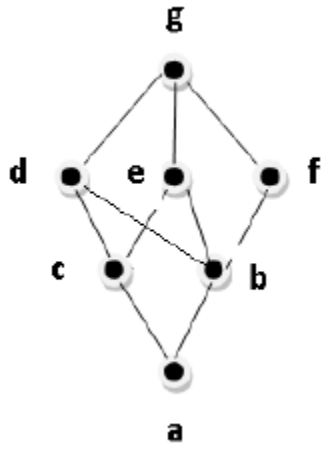


FIGURE 1.4 – Ensemble ordonné ou ordre

- Deux éléments x et y de X sont dits **comparables** si $x \leq_P y$ ou $y \leq_P x$, sinon ils sont dits **incomparables**, ils sont notés $x \parallel y$.

- Si deux éléments de X sont toujours comparables, alors on dit que la relation d'ordre est **totale**.

- Par contre si ces deux éléments sont incomparables alors cette relation d'ordre est dite **partielle**.

- Une relation « $<$ », dite d'ordre **stricte** lorsqu'elle est irréflexive et transitive.

- Une relation d'ordre est transitive ; cette propriété bien que fondamentale rend extrêmement redondante la manière de représenter ou de stocker une de ces relations ; c'est pour cela qu'on introduit la relation de couverture qu'on définit comme suit :

- Un couple $(x, y) \in X^2$ est une **couverture** si $x \leq_P y$ alors il n'existe pas z tel que $x \leq_P z \leq_P y$.

- On dit que y est **couvert** par x (y successeur immédiat de x), x **couvre** y (x successeur Immédiat de y).

- Le couple (x, y) est appelé un **saut** si x et y sont incomparables.

1.3.2 Graphe de couverture

On peut schématiser un graphe de couverture au moyen d'une représentation graphique conventionnelle. Dans celui-ci, les éléments x appartenant à X sont représentés par des points $p(x)$ du plan, de façon que la règle suivante soit respectée : $x \leq_P y$ si et seulement si $p(x)$ est au-dessous de $p(y)$ et il existe une arête reliant le point $p(x)$ et $p(y)$.

Exemple 1.3. $X = \{\{2\}, \{3\}, \{2, 4\}, \{2, 3, 6\}, \{2, 4, 3\}\}$.
Les couples de $P = \{(A, B), A \in X$ *et* $B \in X/A \subseteq B\}$.
Voici le diagramme de P :

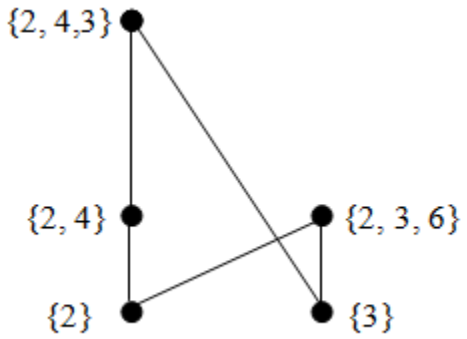


FIGURE 1.5 – Diagramme de l'ordre P

Les sauts *de P :* $(\{2\}, \{3\}), (\{3\}, \{2, 4\}), (\{2, 3, 6\}, \{2, 4, 3\}),$
$(\{2, 4\}, \{2, 3, 6\})$.
Les couvertures *de P :* $(\{2\}, \{2, 4\}), (\{2\}, \{2, 3, 6\}), (\{3\}, \{2, 3, 6\}),$
$(\{3\}, \{2, 4, 3\}), (\{2, 4\}, \{2, 4, 3\})$.

Dans un ensemble ordonné, on peut définir des éléments particuliers comme suit :

Successeur d'un élément et son ensemble des successeurs dans un ordre : Soient $P = (X, \leq_P)$ un ordre et x un élément de X, on appelle successeur de x dans P tout élément y de X, tel que l'on ait $x \leq_P y$. On note $\Gamma^+(x)$ l'ensemble des successeurs de x dans P, soit $\Gamma^+(x) = \{y : y \in X/x \leq_P y\}$.
De la même manière, on définit les prédécesseurs d'un élément.

Élément minimal : Soit $(X, \leq)$ un ensemble ordonné et F une partie non vide de X. Un élément $x \in F$ est minimal quand aucun élément de F n'est strictement plus petit que x : $\forall y \in F, y \leq x \Rightarrow y = x$.

Élément maximal : Soit $(X, \leq)$ un ensemble ordonné et F une partie non vide de X. Un élément $x \in F$ est maximal quand aucun élément de F n'est strictement plus grand que $x : \forall y \in F, x \leq y \Rightarrow y = x$.

Un élément x de X est dit **borne supérieure** de y et z si les deux propriétés suivantes sont vérifiées :

1. $y \leq_P x$ et $z \leq_P x$.
2. Pour $t \in X$, tel que $y \leq_P t$ et $z \leq_P t$ alors $x \leq_P t$.

La condition 2 exprime l'unicité de la borne supérieure, on la note $x \vee y$ ou $sup(x, y)$.

Un élément x de X est dit **borne inférieure** de y et z si les deux propriétés suivantes sont vérifiées :

1. $x \leq_P y$ et $x \leq_P z$.
2. Pour tout $t \in X$, tel que $t \leq_P y$ et $t \leq_P z$ alors $t \leq_P x$.

La condition 2 exprime l'unicité de la borne inférieure, on la note $x \wedge y$ ou $inf(x, y)$.

On peut associer un graphe orienté $G = (X, U)$ appelé **diagramme de Hasse** sur un ordre $P = (X, \leq_P)$, c'est-à-dire pour tout couple d'éléments (x, y) tels que $x \leq_P y$ et il n'existe pas de z dans X tel que $x \leq_P z \leq_P y$, on crée un arc fléché de x à y.

L'ensemble $Q = (X, \leq_Q)$ est une **extension** de l'ensemble P=$(X, \leq_P)$
si : $x \leq_P y \Rightarrow x \leq_Q y$ pour tout x, y de X.

Un ordre total $L = (X, \leq_L)$ est une **extension linéaire** d'un ordre $P = (X, \leq_P)$ si L est une extension de P.

1.4 L'algorithmique et la complexité algorithmique

1.4.1 Définitions

Un algorithme est un ensemble d'opérations élémentaires, appelées instructions ordonnées d'une manière logique et/ou chronologique. Cette suite d'instructions doit être codée dans un langage de programmation ($Pascal, C++, Matlab \ldots$) à fin d'obtenir un programme exécutable par un ordinateur. Ce dernier étant une machine programmée pour exécuter et dérouler des programmes.

1.4.2 Complexité algorithmique

La complexité d'un algorithme est une évaluation du nombre d'opérations élémentaires qu'effectue l'algorithme en fonction de la taille et de la nature des données. Il existe deux types de complexité :
Complexité temporelle : le temps nécessaire pour exécuter l'algorithme.
Complexité spatiale : l'espace mémoire nécessaire pour exécuter l'algorithme.

1.4.3 Les classes d'algorithmes

Les algorithmes usuels peuvent être classés en un certain nombre de grandes classes de complexité :

• Algorithme linéaire : si le temps de son exécution est $O(n)$.

• Algorithme exponentiel : lorsque le temps amorti nécessaire pour son exécution est $O(k^n)$, $k > 1$. Ces algorithmes sont tellement longs à l'exécution, qu'on ne les utilise presque jamais.

• Algorithme polynomial : lorsque le temps amorti nécessaire pour son exécution est $O(n^k)$ avec k un entier. Parmi les algorithmes polynomiaux, on distingue deux catégories ceux qui utilisent un espace polynomial et ceux qui utilisent un espace exponentiel.

• Algorithme avec un délai polynomial : un algorithme rencontre ce critère s'il génère des configurations une après l'autre suivant un ordre, le délai entre deux configuration consécutives doit être borné par un polynôme à la taille de l'entrée.

1.4.4 Les techniques de génération

Un algorithme de génération d'objets consiste à faire une recherche exhaustive de tous les objets d'une même famille, cette recherche s'arrête quand l'objet atteint est jugé interéssant. L'algorithmique de génération des objets combinatoires dépend essentiellement de l'aspect structurel des objets à générer. Dans ce mémoire, nous distinguons deux approches qui se comporte efficacement selon la structure des objets à générer.

L'approche retour arrière

Elle rend possible l'écriture d'algorithmes récursifs, simples à comprendre. Cette technique consiste à décomposer les objets d'une même famille en plusieurs classes. Ensuite, il s'agit de générer les objets classe par classe.
A tout algorithme récursif est associé un arbre de recherche, où les noeuds sont des appels de fonctions et les fils d'un noeud sont des instances de leur père.

Parcours de type lexicographique

Il est basé sur un ordre total des éléments. L'idée principale est d'être capable de générer d'une manière efficace un objet commençant à partir de son prédécesseur dans la suite et de savoir que le premier objet est trouvé.

Chapitre 2

Quelques notions de bases sur les treillis

2.1 Introduction

Ce chapitre a pour but d'introduire les éléments de base de la théorie des treillis en général et quelques classes de treillis en particulier tout en énumérant certaines propriétés algébriques importantes qui les identifient.

2.2 Éléments historiques

La notion de treillis définie comme une structure algébrique munie de deux opérateurs appelés borne inférieure et borne supérieure, elle a été introduite dès la fin du $19^{ème}$ siècle par Dedekind sous le terme de dualgruppe, puis oubliée. Elle a été redécouverte et développée au $20^{ème}$ siècle, sous diverses formes et terminologies entre 1928 et 1936 dans les travaux de Menger, Klein, Stone, Birkhoff, Öre ou encore Von Neumann. L'introduction d'un treillis sous forme ordinale structure ordonnée (i.e. transitive, antisymétrique et réflexive) définie par l'existence d'éléments particuliers appelés bornes supérieures et inférieures, trouve son origine dans les axiomatiques des treillis booléens (algèbre de Boole) dues par exemple à Pierce en 1880, ou à Huntington. Le terme treillis a, quant à lui, été proposé par Birkhoff lors du premier symposium sur la structure de treillis en 1938, pour être finalement repris dans son ouvrage de référence, alors que Menger utilisait le terme de Système et Von Dingen et Öre utilisaient celui de structure. S'ensuivent plusieurs conférences, ainsi que des publications, en particulier dans la revue Algebra Universalis créée en 1970, ainsi que dans la revue Order, revue du domaine créée en 1984. De nombreux ouvrages sur la théorie des treillis portent sur la définition ordinale, en particulier celui de Davey et Priestley ou encore de Grätzer.

2.3 Définitions

On trouve dans la littérature deux définitions d'un treillis : une définition algébrique et une définition ordinale.

Définition algébrique :
Un treillis ou encore une algèbre de treillis est un triplet $T = (X; \wedge; \vee)$ où $\wedge$ et $\vee$ sont deux opérateurs binaires sur l'ensemble X qui vérifient les propriétés suivantes :

- Associativité : pour tous $x; y; z \in X, (x \vee y) \vee z = x \vee (y \vee z)$ et $(x \wedge y) \wedge z = x \wedge (y \wedge z)$
- Commutativité : pour tous $x; y \in X, x \vee y = y \vee x$ et $x \wedge y = y \wedge x$
- Idempotence : pour tous $x \in X; x \vee x = x = x \wedge x$
- Loi d'absorption : pour tous $x; y \in X; x \vee (x \wedge y) = x = x \wedge (x \vee z)$

Définition ordinale :
Un treillis est une paire $T = (X; \leq)$ où $\leq$ est une relation d'ordre sur l'ensemble X, i.e. une relation binaire qui vérifie les propriétés suivantes :

- Réflexivité : pour tout $x \in X$, on a $x \leq x$
- Antisymétrie : pour tous $x; y \in X, x \leq y$ et $y \leq x \Rightarrow x = y$
- Transitivité : pour tous $x; y; z \in X, x \leq y$ et $y \leq z \Rightarrow x \leq z$

Toute paire d'éléments $x; y$ de X admet à la fois une borne inférieure et une borne supérieure :

- La borne inférieure de x et y, notée $x \wedge y$, est l'unique élément maximal (plus grand élément) de l'ensemble des prédécesseurs (ou minorants) de x et y (ensemble des éléments $z \in X$ tels que $z \leq x$ et $z \leq y$).
- La borne supérieure de x et y, notée $x \vee y$, est l'unique élément minimal (i.e. plus petit élément) de l'ensemble des successeurs (ou majorants) de x et y (ensemble des éléments $z \in X$ tels que $z \geq x$ et $z \geq y$).

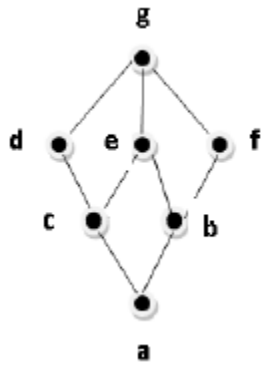


FIGURE 2.1 – Treillis

- Un treillis est dit complet lorsque tous sous ensemble d'éléments (quelque soit son cardinal) admet une borne supérieure et une borne inférieure.

- Lorsque le treillis est fini ou complet, on peut étendre ces notions de bornes supérieures ou inférieures aux sous ensemble de cardinal quelconque.

- Ainsi, les notations suivantes : $\vee Y$ et $\wedge Y$ représentent les bornes d'un ensemble $Y \subseteq X$.

- On notera $\perp = \wedge X$ son élélment minimum, appelée élèment **nul** ; et $\top = \vee X$ son élément maximum, appelé aussi élélment **universel**.

- Notons que le dual d'un treillis est aussi un treillis.

- Tout treillis sur un ensemble fini est un treillis complet.

- Un ordre partiel $P = (X, \leq)$ non vide est dit **Sup-demi-treillis** si pour tout couple d'éléments x et y de X, la borne supérieure $x \vee y$ existe et unique.

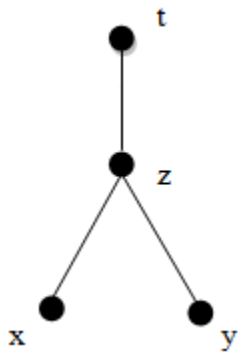


FIGURE 2.2 – Sup-demi-treillis

- Un ordre partiel $P = (X, \leq)$ non vide est dit **Inf-demi-treillis** si pour tout couple d'éléments x et y de X, la borne inférieure $x \wedge y$ existe et unique.

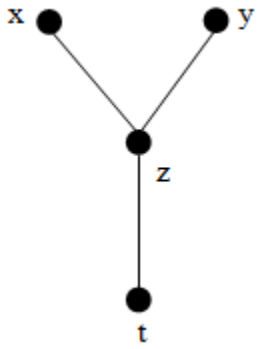


FIGURE 2.3 – Inf-demi-treillis

- Un treillis est un sup-demi-treillis ayant un seul élément minimal.
- Un treillis est un inf-demi-treillis ayant un seul élément maximal.

- Un élément x d'un treillis T est appelé **sup-irréductible** si $a \vee b{=}x$ implique $a = x$ ou $b = x$, ou encore x ne peut s'écrire comme le supremum de deux éléments.

- Un élément x d'un treillis T est appelé **inf-irréductible** si $a \wedge b{=}x$ implique $a = x$ ou $b = x$, ou encore x ne peut s'écrire comme l'infimum de deux éléments.

- Un élément x est inf-irréductible si et seulement s'il est couvert par un seul élément dans le graphe de couverture associé à T.

- L'ensemble des éléments sup-irréductible est noté par $\mathbf{J(T)}$.
- L'ensemble des éléments inf-irréductible est noté par $\mathbf{M(T)}$.

- Ces éléments irréductibles jouent un rôle essentiel dans la structure des treillis.

- Si $T = (X, \leq_T)$ est un treillis, alors $L = (Y, \leq_L)$ avec $Y \subseteq X$ est appelé **sous-treillis** engendré par Y si et seulement si pour tout x, y de Y : $x \vee_T y$ et $x \wedge_T y$ appartiennent a Y.

- Soient x et y de X, tels que $x \leq_T y$, le sous treillis engendré par $t \in T$ tel que $t \leq y$ et $t \geq x$ est appelé **Intervalle** et sera noté $[x, y]_T$.

2.4 Propriétés algébriques d'un treillis

Précédemment, on avait donc définit le treillis en général d'une manière ensembliste, sachant, qu'il existe une définition purement algébrique contenant quelques propriétés très importantes qu'on définira.

Théorème 2.1. *Un treillis T est un ensemble ordonné tel que pour tout couple d'éléments x et y il existe un plus petit majorant noté (ppM ou $x \vee y$) et un plus grand minorant noté (pGm ou $x \wedge y$). Si $\forall x, y, z \in T$ alors les opérations $\wedge$ et $\vee$ possèdent les propriétés algébriques suivantes :*

1. ***Idempotence*** *:*
   $x \vee x = x = x \wedge x$.
2. ***Commutativité*** *:*
   $x \vee y = y \vee x$ *et* $x \wedge y = y \wedge x$.
3. ***Associativité*** *:*
   $x \vee (y \vee z) = (x \vee y) \vee z$.
   $x \wedge (y \wedge z) = (x \wedge y) \wedge z$.
4. ***Absorption*** *:*
   $x \vee (x \wedge y) = x$ *et* $x \wedge (x \vee y) = x$.
5. *Et une relation d'ordre $\leq$ telle que :*
   $x \leq y \Leftrightarrow x \vee y = y \Leftrightarrow x \wedge y = x$.

2.5 Quelques classes de treillis

Les treillis quelconques en général étant définis, on passera aux différentes classes de treillis, dont le nombre est assez élevé. On citera les plus connues : bornés inférieurement, bornés supérieurement, Démantelables, Rangés, Atomistiques, Coatomistiques, Géométriques, Modulaires, Orthomodulaires, Distributifs, Semi-distributifs inférieurement, Semi-distributifs supérieurement, Localement distributifs, Complémentés, Orthocomplémentés, de Stone, de Post, Booléens, de Galois . . .etc.
Chacune de ces classes a sa particularité et possède des propriétés algébriques importantes et intéressantes.
Nous allons étudier les classes de treillis distributifs, complémentés, modulaires et booléens.

2.5.1 Treillis distributif

Un treillis T est dit **distributif** si et seulement si $\forall x, y, z \in T$ les propriétés suivantes sont vérifiées :

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$

et

$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$$

C'est à dire, qu'un treillis est distributif si et seulement si on a la distributivité de l'opération $\wedge$ par rapport à l'opération $\vee$ et inversement.

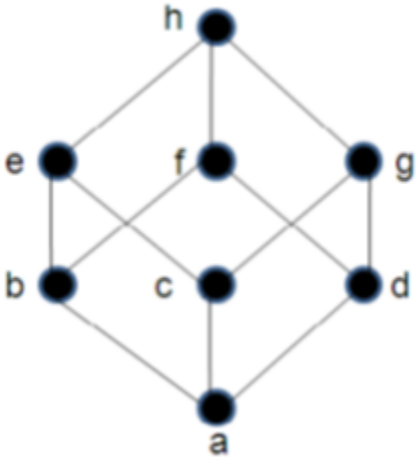


FIGURE 2.4 – Treillis distributif

2.5.2 Treillis complémenté

Qu'est ce qu'un complément ?
Dans un treillis possédant un élément plus grand que tous les autres (que l'on notera M) et un élément plus petit que tous les autres (que l'on notera m), le complément d'un élément $x \in X$ est un élément $y \in X$ tel que : $x \vee y = M$ et $x \wedge y = m$.
Un treillis dans lequel tout élément a au moins un complément est dit **Complémenté**.

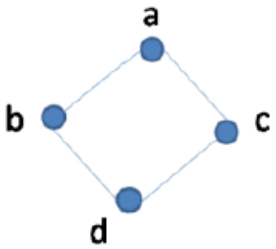


FIGURE 2.5 – Treillis complémenté

2.5.3 Treillis Modulaire

Un treillis T est dit **modulaire** si $\forall x; y; z \in T$, on a :
$x \leq z$ alors $x \vee (y \wedge z) = (x \vee y) \wedge z$.
Tout treillis distributif est modulaire, car l'égalité $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ avec $x \leq z$, donne bien la relation modulaire.

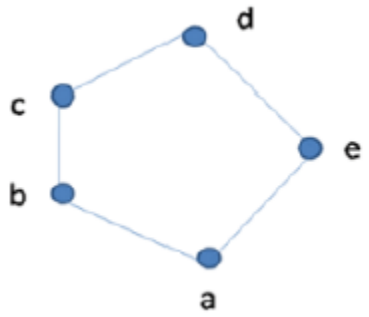


FIGURE 2.6 – Treillis modulaire

2.5.4 Treillis booléen

Un treillis est dit booléen s'il est distributif et complémenté.

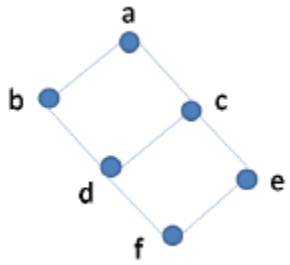


FIGURE 2.7 – Treillis booléen

2.6 Treillis de Galois ou treillis de concepts

Il existe deux écoles, qui utilisent des vocabulaires différents. Les treillis ce concepts selon l'approche allemande de Ganter et Wille (1999) et treillis de Galois selon l'approche française de Barbut et Monjardet (1970).

Un treillis de Galois est un ensemble d'objets dotés de propriétés (ou attributs) organisés en un treillis de concepts. Un concept contient l'ensemble des objets qui possèdent un même sous-ensemble de propriétés, en retenant le plus grand des sous-ensembles de propriétés communes entre les objets (Ganter B. Wille, 1999).
Le processus d'organisation débute avec la construction d'un contexte formel (nous le désignerons par T), ou plus simplement contexte, qui est une table avec les objets disposés en lignes et les propriétés en colonnes. Chaque case est marquée (par exemple avec la valeur 1) si l'objet en ligne possède la propriété en colonne ; elle n'est pas marquée (évaluée à 0) dans le cas contraire. Formellement, un contexte est un triplé (O, M, B) où O est un ensemble d'objets, M un ensemble de propriétés, et B une relation binaire entre les objets et les propriétés, i.e. $B \subseteq O \times M$.

La construction du treillis de concepts se poursuit avec la recherche des concepts. Un concept est défini par une paire de sous-ensembles : un sous-ensemble d'objets, appelé **l'extension** du concept, et un sous-ensemble de propriétés appelé **l'intension**, qui est le sous-ensemble maximal des propriétés que les objets du concept partagent. Pour un ensemble d'objets $W \subseteq O$ et un ensemble de propriétés $H \subseteq M$, on définit l'ensemble des propriétés communes aux objets de W :

$$f(W) = \{h \in H / \forall w \in W; (w, h) \in B\}$$

et l'ensemble des objets qui ont toutes leurs propriétés dans H :

$$g(H) = \{w \in W / \forall h \in H; (w, h) \in B\}$$

La paire (f, g) est une **correspondance de Galois**.
soit L l'ensemble des concepts de (O, M, B) et soit $\leq_L$, l'ordre partiel défini par : $(W_1, H_1) \leq_L (W_2, H_2) \Leftrightarrow H_1 \subseteq H_2 \Leftrightarrow W_2 \subseteq W_1$. La paire $(L, \leq_L)$ est appelée le treillis des concepts de (O, M, B). Un treillis de Galois peut être représenté par un diagramme de Hasse ; c'est un graphe orienté dont l'orientation est implicite en raison de la disposition de haut en bas. C'est un graphe dont les noeuds sont les concepts, ordonnés de haut en bas, selon leur ordre dans le treillis. Chaque concept montre son extension et son intension. Les arêtes du graphe relient les concepts qui sont dans une relation d'ordre directe, sans concept intermédiaire.

Exemple :

O $\backslash M$	a	b	c	d
1	1	0	0	0
2	0	1	1	0
3	1	0	1	0
4	1	0	1	1

Contexte T

Le tableau en dessus montre un exemple de contexte représenté par $T = (O, M, B)$ avec $O = \{1, 2, 3, 4\}$ et $M = \{a, b, c, d\}$.

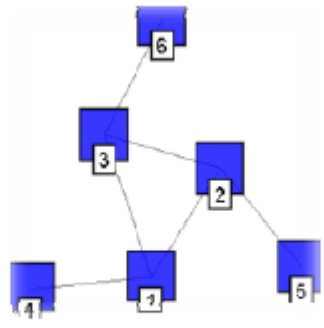
Soit : $O_1 = \{3, 4\}$ $\Rightarrow$f(O_1) = $\{a, c\}$
Et $M_1 = \{a, c\}$ $\Rightarrow$g(M_1) = $\{3, 4\}$.
Alors $(\{3, 4\}, \{a, c\})$ est un concept.

2.6.1 Treillis de Galois dérivé d'un graphe [4]

Après l'identification des concepts, le but suivant est de construire le treillis dont les éléments sont les concepts.
Soit un graphe simple non orienté $G = (X, E)$ ayant A pour matrice d'adjacence. Nous nous proposons de le transformer en un treillis de Galois $G' = (U, F)$ à U sommets et F arêtes de contexte T pour en révéler les cliques maximales et parmi celles-ci les cliques maximum. Dans ce but nous posons $T = A+I$ avec I la matrice identité de même dimension et de même rang que A. Du point de vue du graphe, cela revient à rajouter une boucle à chaque sommet. Du point de vue du treillis de Galois, on a un contexte dans lequel les propriétés sont aussi les objets. Un objet a un autre objet pour propriété si cet autre objet est lui-même ou si cet autre objet lui est adjacent dans G. Nous appelons dérivation d cette transformation.
Formellement, en considérant que X' est la recopie de X :
$d : G = (X, E) \rightarrow G' = (U, F)$ est le treillis de contexte (O, M, B) avec $O = V$, $M = X'$ et $T = A + I$.
Le figure en dessous montre un exemple de graphe à 6 sommets ; dans la première case la matrice d'adjacence du graphe, au milieu le dessin du graphe et à droite le contexte du treillis de Galois dérivé.

	1	2	3	4	5	6
1	0	1	1	1	0	0
2	1	0	1	0	1	0
3	1	1	0	0	0	1
4	1	0	0	0	0	0
5	0	1	0	0	0	0
6	0	0	1	0	0	0

Matrice d'adjacence



Graphe de la matrice d'adjacence

	a	b	c	d	e	f
1	1	1	1	1	0	0
2	1	1	1	0	1	0
3	1	1	1	0	0	1
4	1	0	0	1	0	0
5	0	1	0	0	1	0
6	0	0	1	0		1

Matrice du contexte dérivé

FIGURE 2.8 – Graphe et contexte du treillis de Galois dérivé

Le but de cette transformation est d'établir une relation entre cliques maximales et treillis de Galois pour que le premier problème bénéficie des résultats du second.

Les concepts obtenus par le contexte précédant sont :
$(\{1, 2, 3\}, \{a, b, c\})$
$(\{1, 4\}, \{a, d\})$
$(\{2, 5\}, \{b, e\})$
$(\{3, 6\}, \{c, f\})$
La figure suivante à gauche montre le diagramme de Hasse associé au treillis de Galois de ce contexte dérivé. Il a été construit avec l'outil Lattice Miner (Lahcen and Kwuida, 2010).A droite montre ce même treillis reconstruit à l'aide du logiciel In-Close pour l'identification des concepts à partir du contexte dérivé, suivi de la construction du treillis et d'un diagramme

de Hasse réalisée avec notre grapheur Molage (Crampes et al 2006) qui fait apparaître les cliques maximales comme indiqué dans la section suivante.

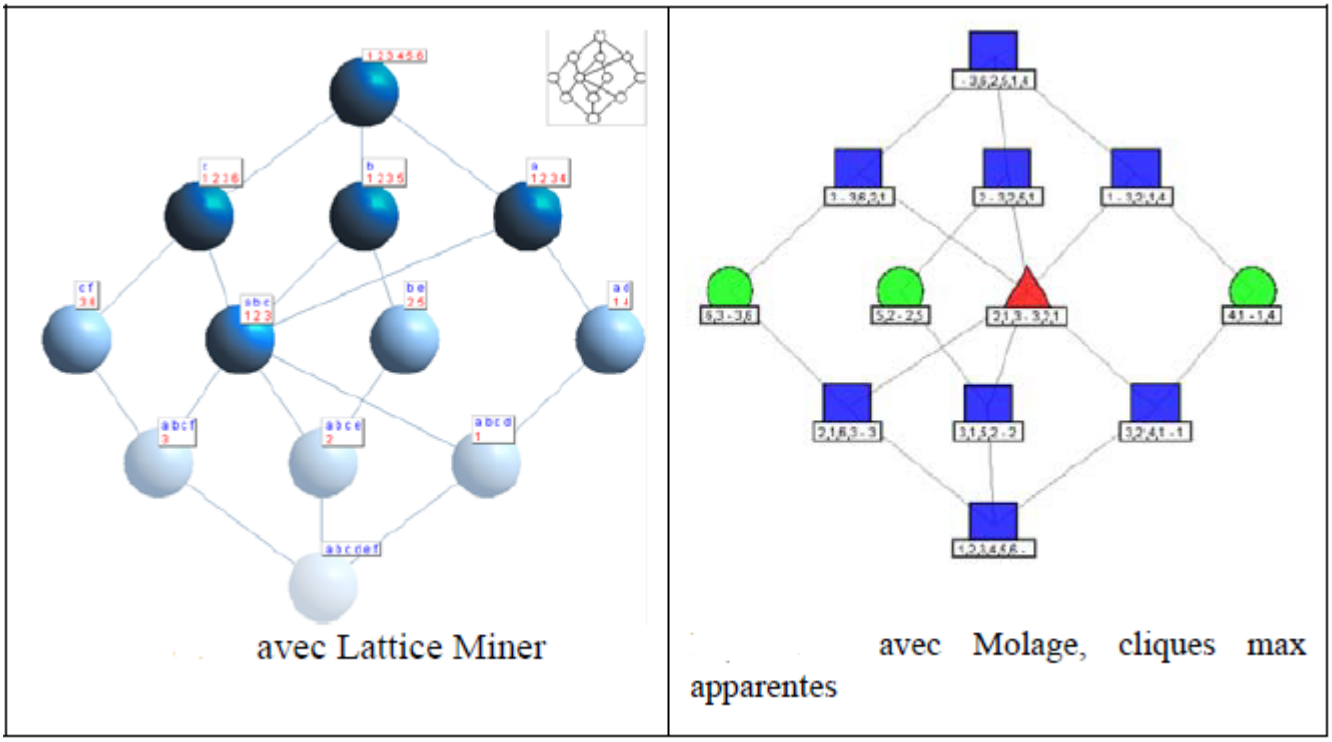


FIGURE 2.9 – Treillis de concepts associé au contexte précédant

Chapitre 3

Aspects algorithmique sur les treillis

3.1 Introduction

Il existe deux familles d'algorithme de construction ou de génération des treillis. Celle des incrémentaux et celle des non incrémentaux, les algorithmes pour la première construisent le treillis au fur et à mesure que les objets arrivent, ceux de la seconde famille construisent le treillis une fois que tous les concepts sont connus. Plusieurs travaux se sont attachés à comparer les algorithmes de construction de treillis entre eux en terme de complexité algorithmique, espace mémoire et temps de calcul.

3.2 Algorithme de reconnaissance d'un treillis de L.Nourine

3.2.1 Principe

L'idée général de l'algorithme de L.Nourine est de reconnaitre si un graphe $G = (X, E)$ est un treillis tout en testant l'existence et l'unicité de la borne supérieure pour tout couple de sommet de X et admettant un élément minimal.

On pose :
Succ(x_i) : l'ensemble de tous les successeurs de x_i dans G.
Inc(x_i) : l'ensemble de tous les successeurs de x_i dans υ_i qui lui sont incomparable dans G.
Imsucc(x_i) : l'ensemble des successeurs immédiats de x_i.
$B(a) = \{a \vee y / y \in Imsucc(x_i)\}$: ensemble des bornes supérieures.

3.2.2 Enoncé de l'algorithme

Algorithme 1 Reconnaissance d'un treillis

Données: la liste d'adjacence de prédécesseurs donne la réduction transitive d'un graphe sans circuit G.
Résultat: G est-il le graphe de couverture d'un treillis ?
Début
Calculer une extension linéaire $\upsilon = x_1, x_2, \ldots, x_n$ de G ;
$Y = (x_n)$ est un sup-demi-treillis.
pour $i = n - 1$ jusqu'à 2 **faire**
Calculer les bornes supérieures entre x_i et ses successeurs ;
pour $y \in]x_i, x_n]$ **faire**
$x_i \vee y = y$; on marque y.
Calculer les bornes supérieures entre x_i et les éléments qui lui sont incomparables.
pour $j = n$ jusqu'à $i + 1$ **faire**
si x_j est non marquer **alors**
x_j est incomparable à x_i ;
si $\omega = min(x_i \vee z)$ tel que $z \in Imsucc(x_i)$ existe **alors**
$x_i \vee x_j = \omega$
sinon
G n'est pas le graphe de couverture d'un treillis.
Fin **si**
Vérifier si x_1 est un élément minimum .
Fin **si**
Fin **pour**
Fin **pour**
Fin **pour**
Fin.

3.2.3 Etude de la complexité [2]

A chaque étape i de l'algorithme :
Etape 1 : calcule la borne supérieure et ses successeurs. Noter pour tout z successeur de x_i, $x_i \vee z = z$.
Il est clair que le temps total est borné par $O(|U|)$.
Etape 2 : calcule la borne supérieure entre x_i et les éléments qui lui sont incomparables.
L'algorithme calcule l'ensemble W des bornes supérieures entre les successeurs immédiats de x_i et z incomparables à x_i, et teste s'il possède un élément minimum.
Le calcule de W coute ($|Imsucc(x_i)|$) et la vérification qu'il possède un minimum revient à vérifier si l'élément le plus petit de W suivant l'extension linière est minimum. Puisque le test de comparabilité se fait en $O(1)$, En utilisant les bornes supérieures déjà calculées, alors la complexité total est en $O(|X|.|Imsucc(x_i)|)$,Soit $O(|X|.|U|)$.

3.2.4 Exemples

Afin de bien comprendre les différentes étapes et instructions de cet algorithme, on appliquera deux exemples sur deux ordres différents.

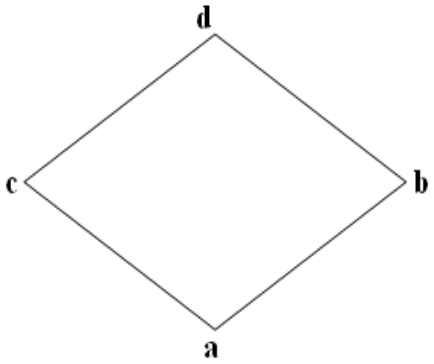


FIGURE 3.1 – Exemple 1

Soit l'extension linéaire suivante $\upsilon_1 = a, b, c, d$.

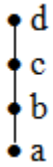


FIGURE 3.2 – Extension linéaire υ_1

L'algorithme commence à partir de l'avant dernier élément de υ_1 qui est c :

•**Pour** c :
Succ(c)=$\{d\}$;
Inc(c)= $\emptyset$;

• **Pour** b :
Succ(b)=$\{d\}$;
Inc(b)=$\{c\}$;
Imsucc(b)=$\{d\}$;
B(c) = $\{c \vee d\} = \{d\}$;
Min B(c)=d ;

Résultat : G est un treillis car la borne supérieure existe et unique ainsi que un élément minimal.

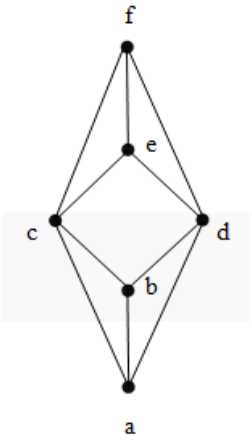


FIGURE 3.3 – Exemple 2

Soit l'extension linéaire suivante $\upsilon_2 = a, b, c, d, e, f$.

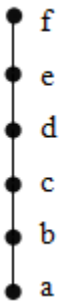


FIGURE 3.4 – Extension linéaire υ_2

L'algorithme commence à partir de l'avant dernier élément de υ_2 qui est e :

• **Pour** e :
Succ(e)= $\{f\}$;
Inc(e)=$\emptyset$;

• **Pour** d :
Succ(d)=$\{e, f\}$;
Inc(d)=$\emptyset$;

•**Pour** c :
Succ(c)=$\{e, f\}$;
Inc(c)=$\{d\}$;
Imsucc(c)=$\{e, f\}$;
B(d)=$\{d \vee e, d \vee f\} = \{e, f\}$;
Min B(d)= e ;

• **Pour** b :
Succ(b)=$\{c, d, e, f\}$;
Inc(b)= $\emptyset$;

Résultat : G n'est pas un treillis.

3.3 Algorithme de construction de graphe de couverture d'un treillis

3.4 Introduction

Cet algorithme a été conçu par Nourine et Raynaud pour la construction et le calcul du graphe de couverture d'un treillis.

3.4.1 Enoncé de l'algorithme :

Cet algorithme utilise une approche en deux étapes :
Première étape : l'algorithme calcule l'arbre lexicographique de la famille F généré par la base B.
X est un ensemble, P ordre total sur X et B une base composée par l'ensemble des parties de X.
La famille F généré par l'union des éléments de base B.

$$F = \{\bigcup b, b \in I / I \subseteq B\};$$

On définit les éléments de F par un couple $F = (f, \gamma(f))$ tel que :

$$\gamma(f) = \{b \in \ B / b \subset f\}$$

Montrons maintenant, comment construire cet arbre lexicographique à partir de la base

$$B = \{b_1, b_2, \ldots, b_m\}$$

- La racine correspond à $F^0 = \emptyset$; l'ensemble vide.
- A l'étape i, on calcule la famille F^i à partir de F^{i-1} en utilisant la formule Suivante :

$$F^i = F^{i-1} \cup \{f \cup b_i / f \in F^{i-1}\};$$

Avec :

$$B_i = \{b_1, b_2, \cdots, b_i\}$$

Algorithme 2 Calcule de l'arbre lexicographique

Données: La base B
Résultat: arbre lexicographique T_F de F.
Début
  $F = \{\emptyset\}$;
  pour $b \in B$ **faire**
    pour $f \in F$ **faire**
      $f' = f \cup b$
      si $f' \notin F$ **alors**
        $F = F \cup \{f'\}$
        $\gamma(f') = \gamma(f) \cup \{b\}$
      Fin **si**
    Fin **pour**
  Fin **pour**
Fin.

Seconde étape : cette étape consiste à calculer le graphe de couverture $G = (F, \subseteq)$ en utilisant le résultat de l'algorithme précédant (l'arbre lexicographique de la famille F générée par la base B) On définit :

$\Delta(f', f) = \gamma(f') \setminus \gamma(f)$ et on note par $\prec$ la relation de couverture ;

Soient f et $f' \in F$ tels que : $f \subset f'$
alors : $f \prec f' \Leftrightarrow \forall (b_1, b_2) \in \Delta(f', f) : b_1 \backslash f' = b_2 \backslash f$.

D'abord, il faut calculer l'ensemble des éléments de couverture noté par $Imsucc(f)$ pour chaque élément de la famille F.
$f' \in F$ est condidat si $f \subset f'$ et f' peut être calculé par $f' = f \cup b$ pour certains $b \in B \setminus \gamma(f)$.
Posons, $S(f) = \{f \cup b / b \in B \setminus \gamma(f)\}$ l'ensemble de condidats pour la couverture f, il peut avoir des éléments redondants ; l'algorithme explore cet ensemble et décide que $f' \in S$ est une couverture de f si f' est trouvé $|\Delta(f', f)|$ fois dans $S(f)$.
Nous calculons l'ensemble $S(f)$ en maintenant le nombre d'occurrence de chaque élément f' dans le compteur $count(f')$, ensuite on verifie si$|\Delta(f', f)| = count(f')$ alors f' couvre f.
L'algorithme suivant construira le graphe de couverture du treillis $G = (F, \subseteq)$.

Algorithme 3 Calcul du graphe de couverture de $G = (F, \subseteq)$

Données: arbre lexicographique de F et de $\gamma(F)$
Résultat: liste d'adjacence des $Imsucc$ du graphe de couverture du treillis $(F, \subseteq)$
Début
  Initialiser count(f) à 0 pour tout $f \in F$;
  pour $f \in F$ **faire**
    pour $b \in B \setminus \gamma(f)$ **faire**
      $f' = f \cup b$
      $Count(f') + +$
      si $|\gamma(f')| = count(f') + |\gamma(f)|$ **alors**
        $Imsucc(f) = Imsucc(f) \cup \{f'\}$
      Fin **si**
    Fin **pour**
    Réinitialiser Count
  Fin **pour**
Fin.

Algorithme 4 Construction d'un graphe de couverture

Données: Base B à n éléments
Résultat: Graphe de couverture $G = (F, \subseteq)$.
Début
  F= Tree (B) ;
  Graphe de couverture $G = (F, \subseteq)$;
Fin.

3.4.2 Complexité algorithmique [2]

Par décompte du nombre de boucles, cette étape a un coût de ;
$O((|X| + |B|)|B|.|F|)$ en temps ;
et $O((|X| + |B|)|F|)$ en espace ;
Le première étape est réalisée en : $O((|X| + |B|)|B|.|F|)$.
Le seconde étape est réalisée aussi en : $O((|X| + |B|)|B|.|F|)$.
Le coût total de l'algorithme est en : $O((|X| + |B|)|B|.|F|)$.

3.4.3 Exemple

soit $X = \{1, 2, 3, 4, 5\}$ un ensemble et B une base composée par quelques parties de X.
On désigne par F la famille générée par l'union des éléments de la base B,tel que
$F = \{\cup_{b\in I}/I \subseteq B\}$.
On définit $B = \{a = \{14\}, b = \{234\}, c = \{25\}\}$.
Appliquons la première étape qui consiste à générer la famille F représentée dans un arbre lexicographique.
On pose $\gamma(f) = \{b \in B/b \subset f\}$

1.$F = \{\emptyset\}$;(la racine de l'arbre T_F) ; $\gamma(F) = \{\emptyset\}$;

(a)**Pour** $b = \{14\}$ et $f = \{\emptyset\}$;
$f' = f \cup b = \emptyset \cup \{14\} = \{14\} \notin F$;
$F = F \cup \{f'\} = \emptyset \cup \{\{14\}\} = \{\emptyset, \{14\}\}$;
$\gamma(f') = \gamma(f) \cup \{b\} = \emptyset \cup \{14\} = \{14\} = a$;
$\gamma(f) = \{\emptyset, \{a\}\}$;

2. $F = \{\emptyset, \{14\}\}, \gamma(F) = \{\emptyset, \{a\}\}$;

(a) **Pour** $b = \{234\}$ et $f = \emptyset$;
$f' = f \cup b = \emptyset \cup \{234\} = \{234\} \notin F$;
$F = F \cup \{f'\} = \{\emptyset, \{14\}, \{234\}\}$;
$\gamma(f') = \gamma(f) \cup \{b\} = \emptyset \cup \{234\} = \{234\} = b$;
$\gamma(F) = \{\emptyset, \{a\}, \{b\}\}$;

(b) **Pour** $b = \{234\}$ et $f = \{14\}$;
$f' = f \cup b = \{14\} \cup \{234\} = \{1234\} \notin F$;
$F = F \cup \{f'\} = \{\emptyset, \{14\}, \{234\}, \{1234\}\}$;
$\gamma(f') = \gamma(\{1234\}) = ab$;
$\gamma(F) = \{\emptyset, \{a\}, \{b\}, \{ab\}\}$;

3. $F = \{\emptyset, \{14\}, \{234\}, \{1234\}\}, \gamma(F) = \{\emptyset, \{a\}, \{b\}, \{ab\}\}$;

(a) **Pour** $b = \{25\}$ et $f = \emptyset$;
$f' = f \cup b = \{\emptyset\} \cup \{25\} = \{25\} \notin F$;
$F = F \cup \{f'\} = \{\emptyset, \{14\}, \{234\}, \{1234\}, \{25\}\}$;
$\gamma(f') = \gamma(\{25\}) = c$;
$\gamma(F) = \{\emptyset, \{a\}, \{b\}, \{ab\}, \{c\}\}$;

(b) **Pour** $b = \{25\}$ et $f = \{14\}$;
$f' = f \cup b = \{14\} \cup \{25\} = \{1245\} \notin F$;
$F = F \cup \{f'\} = \{\emptyset, \{14\}, \{234\}, \{1234\}, \{25\}, \{1245\}\}$;
$\gamma(f') = \gamma(\{1245\}) = ac$;
$\gamma(F) = \{\emptyset, \{a\}, \{b\}, \{ab\}, \{c\}, \{ac\}\}$;

(c) **Pour** $b = \{25\}$ et $f = \{234\}$;
$f' = f \cup b = \{234\} \cup \{25\} = \{2345\} \notin F$;
$F = F \cup \{f'\} = \{\emptyset, \{14\}, \{234\}, \{1234\}, \{25\}, \{1245\}, \{2345\}\}$;
$\gamma(f') = \gamma(\{2345\}) = bc$;
$\gamma(F) = \{\emptyset, \{a\}, \{b\}, \{ab\}, \{c\}, \{ac\}, \{bc\}\}$;

(d) **Pour** $b = \{25\}$ et $f = \{1234\}$;
$f\,' = f \cup b = \{1234\} \cup \{25\} = \{12345\} \notin F$;
$F = F \cup \{f'\} = \{\emptyset, \{14\}, \{234\}, \{1234\}, \{25\}, \{1245\}, \{2345\}, \{12345\}\}$;
$\gamma(f') = \gamma(\{12345\}) = abc$;
$\gamma(F) = \{\emptyset, \{a\}, \{b\}, \{ab\}, \{c\}, \{ac\}, \{bc\}, \{abc\}\}$;

Finalement, à partir de la base ; $B = \{a = \{14\}, b = \{234\}, c = \{25\}\}$;
On obtient : $F = \{\emptyset, \{14\}, \{234\}, \{1234\}, \{25\}, \{1245\}, \{2345\}, \{12345\}\}$;
et $\gamma(F) = \{\emptyset, \{a\}, \{b\}, \{ab\}, \{c\}, \{ac\}, \{bc\}, \{abc\}\}$;

Déssinons l'arbre lexicographique de F.

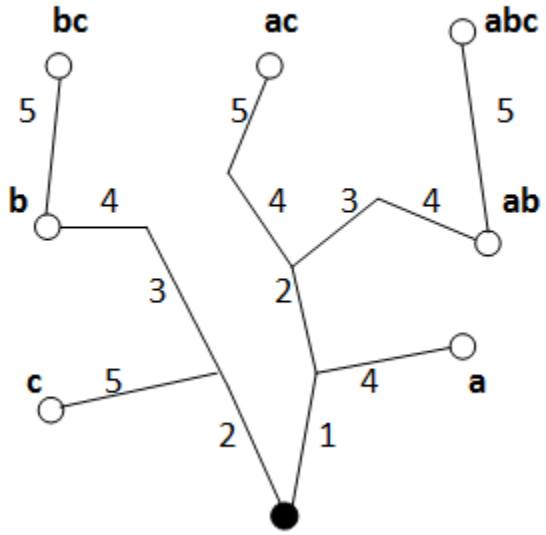


FIGURE 3.5 – Arbre lexicographique de la famille F

Appliquons la deuxième étape qui calcule le graphe de couverture à partir de l'arbre lexicographique des familles :
$F = \{\emptyset, \{14\}, \{234\}, \{1234\}, \{25\}, \{1245\}, \{2345\}, \{12345\}\}$;
et $\gamma(F) = \{\emptyset, \{a\}, \{b\}, \{ab\}, \{c\}, \{ac\}, \{bc\}, \{abc\}\}$;
générées par la base $B = \{a = \{14\}, b = \{234\}, c = \{25\}\}$;

Il donne les listes d'adjacence des successeurs immédiats du graphe de couverture du treillis $(F, \subseteq)$.

On démarre l'algorithme 2 par ;
count(f)=0 ;pour tout $f \in F$,on calcule :
$S(f) = \{f \cup b / b \notin f, b \in B, f \in F\}$;

1. $f = \emptyset$; *S(f)* $= \{\{14\}, \{234\}, \{25\}\}$

(a) **Pour** *f'*=$\{14\}$, *count(f')* = *count*$(\{14\}) = 1$;
$\Delta(f', f) = \Gamma(f')\backslash\Gamma(f) = \{a\}\backslash\emptyset = \{a\}$;
$|\Delta(f', f)| = 1 \Longrightarrow |\Delta(f', f)| =$ *count(f')* $= 1$;
(f, f') $= (\emptyset, \{14\})$ est une couverture ;
ImSucc(f)=ImSucc($\emptyset$*)* $= \{\{14\}\}$;

(b) **Pour** *f'*=$\{234\}$, *count(f')* = *count*$(\{234\}) = 1$;
$\Delta(f', f) = \Gamma(f')\backslash\Gamma(f) = \{b\}\backslash\emptyset = \{b\}$;
$|\Delta(f', f)| = 1 \Longrightarrow |\Delta(f', f)| =$ *count(f')* $= 1$;
(f, f') $= (\emptyset, \{234\})$ est une couverture ;
ImSucc(f)=ImSucc($\emptyset$*)* $= \{\{14\}, \{234\}\}$;

(c) **Pour** *f'*=$\{25\}$, *count(f')* = *count*$(\{25\}) = 1$;
$\Delta(f', f) = \Gamma(f')\backslash\Gamma(f) = \{c\}\backslash\emptyset = \{c\}$;
$|\Delta(f', f)| = 1 \Longrightarrow |\Delta(f', f)| =$ *count(f')* $= 1$;
(f, f') $= (\emptyset, \{25\})$ est une couverture ;
ImSucc(f)=ImSucc($\emptyset$*)* $= \{\{14\}, \{234\}, \{25\}\}$;
count(f')=0 ;

2. $f = \{14\}$; *S(f)* $= \{\{1234\}, \{1245\}\}$

(a) **Pour** *f'*=$\{1234\}$, *count(f')* = *count*$(\{1234\}) = 1$;
$\Delta(f', f) = \Gamma(f')\backslash\Gamma(f) = \{ab\}\backslash\{a\} = \{b\}$;
$|\Delta(f', f)| = 1 \Longrightarrow |\Delta(f', f)| =$ *count(f')* $= 1$;
(f, f') $= (\{14\}, \{1234\})$ est une couverture ;
ImSucc(f)=ImSucc($\{14\}$*)* $= \{\{14\}, \{1234\}\}$;

(b) **Pour** *f'*=$\{1245\}$, *count(f')* = *count*$(\{1245\}) = 1$;
$\Delta(f', f) = \Gamma(f')\backslash\Gamma(f) = \{ac\}\backslash\{a\{= \{c\}$;
$|\Delta(f', f)| = 1 \Longrightarrow |\Delta(f', f)| =$ *count(f')* $= 1$;
(f, f') $= (\{14\}, \{1245\})$ est une couverture ;
ImSucc(f)=ImSucc($\{14\}$*)* $= \{\{14\}, \{1245\}\}$;
count(f')=0 ;

3. $f = \{234\}$; *S(f)* $= \{\{1234\}, \{2345\}\}$

(a) **Pour** *f'*=$\{1234\}$, *count(f')* = *count*$(\{1234\}) = 1$;
$\Delta(f', f) = \Gamma(f')\backslash\Gamma(f) = \{ab\}\backslash\{b\} = \{a\}$;
$|\Delta(f', f)| = 1 \Longrightarrow |\Delta(f', f)| =$ *count(f')* $= 1$;
(f, f') $= (\{234\}, \{1234\})$ est une couverture ;
ImSucc(f)=ImSucc($\{234\}$*)* $= \{1234\}\}$;

(b) **Pour** *f'*=$\{2345\}$, *count(f')* = *count*($\{2345\}$) = 1;
$\Delta(f', f) = \Gamma(f')\backslash\Gamma(f) = \{bc\}\backslash\{b\{= \{c\}$;
$|\Delta(f', f)| = 1 \Longrightarrow |\Delta(f', f)|$ = *count(f')* = 1;
(f, f') = $(\{234\}, \{2345\})$ est une couverture ;
ImSucc(f)=ImSucc($\{234\}$) = $\{\{2345\}\}$;
count(f')=0 ;

4. *f*= $\{1234\}$; *S(f)* = $\{\{12345\}\}$

(a) **Pour** *f'*=$\{12345\}$, *count(f')* = *count*($\{12345\}$) = 1;
$\Delta(f', f) = \Gamma(f')\backslash\Gamma(f) = \{abc\}\backslash\{ab\} = \{c\}$;
$|\Delta(f', f)| = 1 \Longrightarrow |\Delta(f', f)|$ = *count(f')* = 1;
(f, f') = $(\{1234\}, \{12345\})$ est une couverture ;
ImSucc(f)=ImSucc($\{1234\}$) = $\{\{12345\}\}$;
count(f')=0 ;

5. *f*= $\{25\}$; *S(f)* = $\{\{1245\}, \{2345\}\}$

(a) **Pour** *f'*=$\{1245\}$, *count(f')* = *count*($\{1245\}$) = 1;
$\Delta(f', f) = \Gamma(f')\backslash\Gamma(f) = \{ac\}\backslash\{c\} = \{a\}$;
$|\Delta(f', f)| = 1 \Longrightarrow |\Delta(f', f)|$ = *count(f')* = 1;
(f, f') = $(\{25\}, \{1245\})$ est une couverture ;
ImSucc(f)=ImSucc($\{25\}$) = $\{1245\}\}$;

(b) **Pour** *f'*=$\{2345\}$, *count(f')* = *count*($\{2345\}$) = 1;
$\Delta(f', f) = \Gamma(f')\backslash\Gamma(f) = \{bc\}\backslash\{c\} = \{b\}$;
$|\Delta(f', f)| = 1 \Longrightarrow |\Delta(f', f)|$ = *count(f')* = 1;
(f, f') = $(\{25\}, \{2345\})$ est une couverture ;
ImSucc(f)=ImSucc($\{25\}$) = $\{\{2345\}\}$;
count(f')=0 ;

6. *f* = $\{1245\}$; *S(f)* = $\{\{12345\}\}$

(a) **Pour** *f'*=$\{12345\}$, *count(f')* = *count*($\{12345\}$) = 1;
$\Delta(f', f) = \Gamma(f')\backslash\Gamma(f) = \{abc\}\backslash\{ac\} = \{b\}$;
$|\Delta(f', f)| = 1 \Longrightarrow |\Delta(f', f)|$ = *count(f')* = 1;
(f, f') = $(\{1245\}, \{12345\})$ est une couverture ;
ImSucc(f)=ImSucc($\{1245\}$) = $\{\{12345\}\}$;
count(f')=0 ;

7. *f* = $\{2345\}$; *S(f)* = $\{\{12345\}\}$

(a) **Pour** *f'*=$\{12345\}$, *count(f')* = *count*($\{12345\}$) = 1;
$\Delta(f', f) = \Gamma(f')\backslash\Gamma(f) = \{abc\}\backslash\{bc\} = \{a\}$;
$|\Delta(f', f)| = 1 \Longrightarrow |\Delta(f', f)|$ = *count(f')* = 1;
(f, f') = $(\{2345\}, \{12345\})$ est une couverture ;
ImSucc(f)=ImSucc($\{2345\}$) = $\{\{12345\}\}$;
count(f')=0 ;

8. $f = \{12345\}$; $S(f) = \{\emptyset\}$

Enfin,Les couvertures (f', f) ou successeurs immédiats sont les suivants :

1. $(\emptyset, \{14\})$ est une couverture ;
$ImSucc(\emptyset) = \{\{14\}\}$;

2. $(\emptyset, \{234\})$ est une couverture ;
$ImSucc(\emptyset) = \{\{234\}\}$;

3. $(\emptyset, \{25\})$ est une couverture ;
$ImSucc(\emptyset) = \{\{25\}\}$;
4. $(\{14\}, \{1234\})$ est une couverture ;
$ImSucc(\{14\}) = \{\{1234\}\}$;

5. $(\{14\}, \{1245\})$ est une couverture ;
$ImSucc(\{14\}) = \{\{1245\}\}$;

6. $(\{234\}, \{1234\})$ est une couverture ;
$ImSucc(\{234\}) = \{\{1234\}\}$;

7. $(\{234\}, \{2345\})$ est une couverture ;
$ImSucc(\{234\}) = \{\{2345\}\}$;

8. $(\{1234\}, \{12345\})$ est une couverture ;
$ImSucc(\{1234\}) = \{\{12345\}\}$;

9. $(\{25\}, \{1245\})$ est une couverture ;
$ImSucc(\{25\}) = \{\{1245\}\}$;

10. $(\{25\}, \{2345\})$ est une couverture ;
$ImSucc(\{25\}) = \{\{2345\}\}$;

11. $(\{1245\}, \{12345\})$ est une couverture ;
$ImSucc(\{1245\}) = \{\{12345\}\}$;

12. $(\{2345\}, \{12345\})$ est une couverture ;
$ImSucc(\{2345\}) = \{\{12345\}\}$;

On doit récupèrer aussi F et $\gamma(F)$, tels que

$F = \{\emptyset, \{14\}, \{234\}, \{1234\}, \{25\}, \{1245\}, \{2345\}, \{12345\}\}$;
$et \gamma(F) = \{\emptyset, \{a\}, \{b\}, \{ab\}, \{c\}, \{ac\}, \{bc\}, \{abc\}\}$;

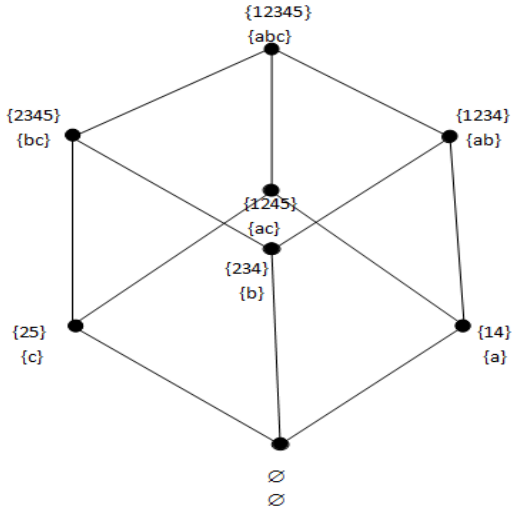


FIGURE 3.6 – Graphe de couverture $G = (F, \subseteq)$

A la fin, on peut résumer tous les trois algorithmes dans le tableau suivant :

Algorithme	But	Critères	Complexité
L.Nourine	Reconnaissance d'un treillis	Graphe orienté et sans circuit	$O(n.m)$
L.Nourine et O.Raynaud	Génère la famille F représentée dans un arbre lexicographique	- Une base B	$O((\|X\| + \|B\|)\|B\|.\|F\|)$
L.Nourine et O.Raynaud	Calcule les relations de couverture des éléments de F	- Arbre lexicographique de la famille F	$O((\|X\| + \|B\|)\|B\|.\|F\|)$

Chapitre 4

Génération des cliques maximales

4.1 Inroduction

Dans ce chapitre, on introduit deux algorithmes de génération des cliques maximales qui sont : l'algorithme de Johnson et Al et l'algorithme de Tsukiyama et Al. Nous allons découvrir le mode d'opérations de chaque algorithme suivant les critères qui sont : l'espace mémoire, le temps d'exécution et la génération des cliques suivant l'ordre lexicographique.
Avant de commencer à étudier ces algorithmes nous voulons passer par l'algorithme qui respecte le critère de l'ordre lexicographique qui est l'algorithme Lex-BFS.

4.2 Algorithme Lex-BFS

4.2.1 Introduction

Le parcours en largeur lexicographique (Lex-BFS) a été proposé par Rose, Tarjan et Lueker en 1976 [RTL76]. L'algorithme Lex-BFS (Lexicographic Breadth First Search) est un algorithme très utilisé, pour des problèmes divers. Il fournit un ordre $\{\sigma(1), \sigma(2), \ldots, \sigma(n)\}$ des sommets du graphe donné en entrée.

4.2.2 Principe

Pour donner la description de l'algorithme, supposons que chaque sommet a une étiquette, qui est une suite décroissante d'entiers (au début les étiquettes sont considérées vides). Les sommets sont pris l'un après l'autre, on applique un critère de comparaison : on préfère les sommets dont les voisins déjà visités l'ont été très tôt. Pour exprimer cela, on attribue à chaque sommet une sorte de poids (d'autant plus grand que le sommet a été visité tôt) et on met dans l'étiquette de chaque sommet la liste (decroissante) des poids de ses voisins déjà visités. Ensuite, on choisit à chaque étape le sommet dont l'étiquette est maximum par ordre lexicographique (cet ordre est l'ordre du dictionnaire, où les lettres sont remplacées par des chiffres). On répète l'opération tant qu'il existe des sommets non visités.
Nous présentons ci-dessous l'algorithme en détails.

4.2.3 Algorithme

Notons que $G = (X, E)$ un graphe simple avec $|X| = n$; $|E| = m$ et $\Gamma(x)$ l'ensemble des sommets adjacents de x.

Algorithme 5 (Lex-BFS, parcours en largeur lexicographique).

Données: un graphe $G = (X, E)$ et un sommet source s
Résultat: un ordre total σ de X.
Début
  Affecter l'étiquette $\emptyset$ à chaque sommet.
  $label(s) := \{n\}$
  pour i allant de n à 1 **faire**
    Choisir un sommet x d'étiquette lexicographique maximale.
    $\sigma(i) := x$
    pour chaque sommet non numéroté w de $\Gamma(x)$ **faire**
      $label(w) := label(w) \cup \{i\}$ ▷ Concaténation de i à la fin de l'étiquette de w
    Fin **pour**
  Fin **pour**
Fin.

4.2.4 Complexité

Étiquetter ou "lire" l'étiquette d'un sommet ou d'une arête se fait en $O(1)$. Chaque sommet est étiquetté deux fois se fait en $O(n)$ et l'opération Adjacents(x) est appelée une fois pour chaque sommet x, donc la complexité en temps de l'algorithme Lex-BFS est $O(m+n)$.

4.2.5 Exemple d'application

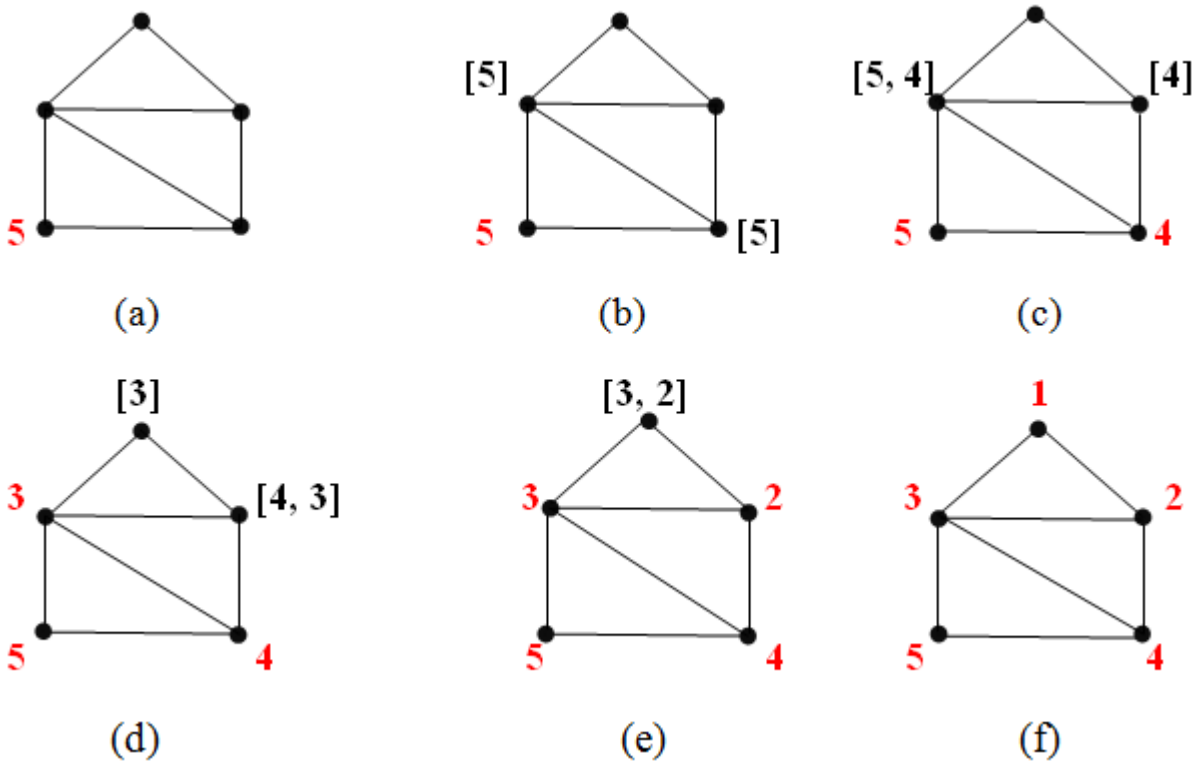


FIGURE 4.1 – Déroulement de l'algorithme Lex-BFS sur un exemple

4.3 Algorithme de Johnson et Al

4.3.1 Introduction

L'algorithme de Johnson et Al est utilisé dans les problèmes d'ordonnancement ; notamment les files d'attente et la génération des cliques. 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 pouvoir générer efficacement une clique maximale à partir de sa prédécesseur dans l'ordre lexicographique prédéfini, sachant que la première clique est donnée.
Notons qu'il est facile de générer la première clique maximale dans l'ordre lexicographique, avec une complexité $O(n + m)$.

4.3.2 Algorithme

Algorithme 6 Jonson et Al (première clique)

Données: un graphe $G = (X, E)$.
Résultat: La première clique maximale dans l'ordre lexicographique.
Début
  $C = \{1\}$
  pour $i = 2$ à n **faire**
    si $\Gamma(i) \cap C = C$ **alors**
      $C = C + \{i\}$
    Fin **si**
  Fin **pour**
Fin.

4.3.3 Principe

On cherche à déterminer toutes les cliques maximales du graphe G, l'organigramme ci-dessous montre le déroulement de cet algorithme.

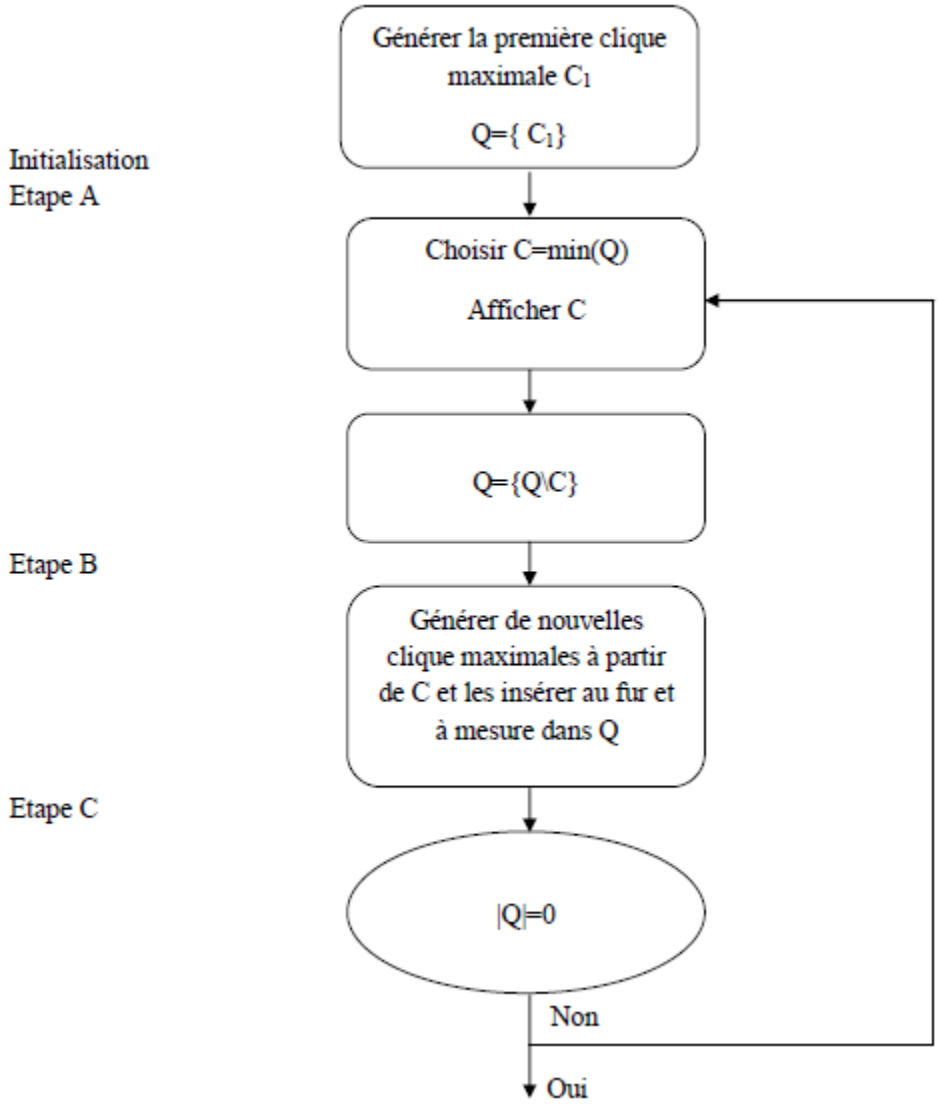


FIGURE 4.2 – Organigramme représentant le déroulement de l'algorithme de Johnson et Al.

Au début, à l'étape A, on prend la clique la plus petite dans Q suivant l'ordre lexicographique ; comme Q est un ensemble vide au départ, la première clique, obtenue à travers l'algorithme 6, est la première clique maximale que l'algorithme affiche.
A l'étape B, l'algorithme génère à partir de $C = min(Q)$ un ensemble de cliques maximales plus grandes, suivant l'ordre lexicographique, que C ; il les compare par la suite aux cliques dans Q et insère celles qui n'existent pas déjà.
Pour générer les cliques, l'algorithme procède de la manière suivante :
Pour chaque sommet $i \in C$, et pour chaque sommet j, tel que $j > i$ et $j \notin \Gamma(i)$, si $(C \cap \{1, \ldots, j\} \cap \Gamma(j)) \cup \{j\}$ est une clique maximale du graphe induit par les sommets

$\{1, \ldots, j\}$ alors on l'insère dans l'ensemble Q.
Lorsque Q devient vide ($Q = \emptyset$), toutes les cliques maximales du graphe ont été générées dans l'ordre lexicographique.

4.3.4 Algorithme

Algorithme 7 Johnson et Al

Données: un graphe $G = (X, E)$ non orienté
Résultat: génère toutes les cliques maximales dans G.

```
Début
Soit C_1 la première clique maximale de G ;
insérer C_1 dans Q ;
tant que Q ≠ ∅ faire
  début
  C := minQ ;
  afficher (C)
  pour chaque sommet i ∈ C faire
    début
    pour tout sommet j de G non adjacent à i tels que i < j faire
      début
      si (C ∩ {1,...,j} ∩ Γ(j)) ∪ {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 ∩ {1,...,j} ∩ Γ(j)) ∪ {j}
        si C' ∉ Q alors
          insérer C' dans Q
        Fin si
        fin
      Fin si
      fin
    Fin pour
    fin
  Fin pour
  fin
Fin tant que
Fin.
```

4.3.5 Complexité

Le temps demandé pour chaque extraction de clique maximale de l'ensemble Q est $O(n \log(|C(G)|))$ et pour chaque nouvelle clique maximale trouvée $O(n + m)$. Ajoutons à cela le temps d'insertion d'une clique maximale dans Q après comparaison aux cliques existantes dans Q est $O(n \log(|C(G)|))$.

On obtient donc le délai total qui est $O(n\log(|C(G)|) + n + m + n\log(|C(G)|)) = O(n^3)$ qui est polynomial.

Malheureusement, cet algorithme bien qu'il respecte les deux critères du temps polynomial est de la génération par ordre lexicographique ne respecte pas le critère de l'espace polynomial, qui est exponentiel. Ceci est dû au fait que toutes les cliques maximales sont sauvegardées dans l'ensemble Q pendant l'exécution et elles sont toutes comparées à chaque fois à une nouvelle clique maximale.

4.3.6 Exemple d'application

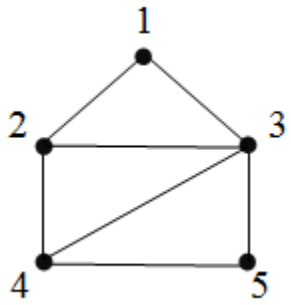


FIGURE 4.3 – Graphe de l'exemple d'application

- **Etape A** : Trouver C_1 la première clique maximale dans l'ordre lexicographique en appliquant l'algorithme (6).
  $C = 1$
  Pour $i = 2$ à 5 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 $C = \{1,2\} + \{3\} = \{1,2,3\}$
  - $i = 4 : \Gamma(4) \cap \{1,2,3\} = \{2,3,5\} \cap \{1,2,3\} = \{2,3\} \neq \{1,2,3\}$
  - $i = 5 : \Gamma(5) \cap \{1,2,3\} = \{3,4\} \cap \{1,2,3\} = \{3\} \neq \{1,2,3\}$

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

- **Etape B** : Générer d'autres cliques maximales à partir de C_1
  $C_1 = \{1,2,3\}$
  $Q = \{1,2,3\}$
  $C = \min Q \Rightarrow C = \{1,2,3\}$, afficher (C).

Pour $i = 1$:
$\overline{\Gamma}(1) = \{4, 5\}$ tels que : $1 < 4$ et $1 < 5$.

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

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

Pour $i = 2$:
$\overline{\Gamma}(2) = \{5\}$ tel que : $2 < 5$.

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

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

Pour $i = 3$:
$\overline{\Gamma}(3) = \emptyset$

$Q = Q - \{1, 2, 3\} = \{2, 3, 4\}$
$C = \min Q \Rightarrow C = \{2, 3, 4\}$, afficher (C).

Pour $i = 2$:
$\overline{\Gamma}(2) = \{5\}$ tel que : $2 < 5$.

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

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

Pour $i = 3$:
$\overline{\Gamma}(3) = \emptyset$

Pour $i = 4$:
$\overline{\Gamma}(4) = \{1\}$ mais $4 > 1$
$Q = \{3, 4, 5\}$

$Q = Q - \{3, 4, 5\} = \emptyset$.
Q est vide donc fin de l'exécution.
Toutes les cliques maximales sont générées suivant l'ordre lexicographique, qui sont :$\{1, 2, 3\}$, $\{2, 3, 4\}$ et $\{3, 4, 5\}$.

4.4 Algorithme de Tsukiyama et Al

4.4.1 Introduction

Pour générer les cliques maximales, cet algorithme se base sur la technique du retour arrière 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 ne vérifie pas le troisième critère car il génère les cliques maximales du graphe suivant un ordre quelconque.

4.4.2 Principe

A chaque itération $i = \{1, \ldots, n\}$, l'algorithme génère toutes les cliques maximales du sous graphe induit par les sommets $\{1, \ldots, i\}$ en leurs ajoutant tous les sommets supérieurs à i. A l'itération $i = n$, toutes les cliques maximales du graphe seront générées.
Une solution partielle S^i est noté par le couple (C^i, X^i), où C^i est une clique maximale du sous-graphe induit par les sommets $\{1, , i\}$ et $X^i = \{i+1, \ldots, n\}$.
A la profondeur $k = 0$, nous obtenons qu'une seule solution partielle qui est $S^0 = (\emptyset, X)$ et à la profondeur $k = 1$, $S^1 = (1, \{2, \ldots, n\}) = X$ l'ensemble des sommets du graphe.
Maintenant, nous allons montrer comment construire S^{i+1} à partir de S^i.
Soit $S^i = (C^i, X^i)$ une solution partielle. Déterminons les solutions partielles qui seront générer par S^i, pour ceci on distingue deux cas :

1. Si $i+1$ est adjacent à tous les sommets S^i, c'est à dire, $C^i \cap \Gamma(i+1) = C^i$ alors $C^i \cup \{i+1\}$ est une clique maximale du graphe induit par les sommets $\{1, \ldots, i+1\}$. Par suite $S^{i+1} = (C^i \cup \{i+1\}, X^{i+1})$ est une solution partielle de la profondeur $i+1$, déduite de S^i.
2. S'il existe au moins un sommet de C^i qui n'est pas adjacent à $i+1$ ($C^i \cap \Gamma(i+1) \neq C^i$), alors C^i est une clique maximale du graphe induit par les sommets $\{1, \ldots, i+1\}$, par suite $S^{i+1} = (C^{i+1}, X^{i+1})$ est une solution partielle à la profondeur $i+1$, avec $C^{i+1} = C^i$.

Si $((C^i \cap \Gamma(i+1)) \cup \{i+1\})$ est une clique maximale du graphe induit par les $i+1$ premiers sommets alors nous aurons une deuxième solution partielle issue de S^i ; c'est $S^{i+1} = (C^{i+1}, X^{i+1})$, avec $C^{i+1} = (C^i \cap \Gamma(i+1)) \cup \{i+1\})$.
Notons bien que ce traitement se fait sur toutes les solutions partielles générées à l'itération i , pour déduire celles de l'itération $i+1$. De cette façon l'algorithme générera toutes les solutions partielles à l'itération $i+1$, par celles qui étaient à l'itération i.

Dans la figure qui suit, nous représentons l'évolution des solutions partielles de l'algorithme de Tsukiyama et Al.

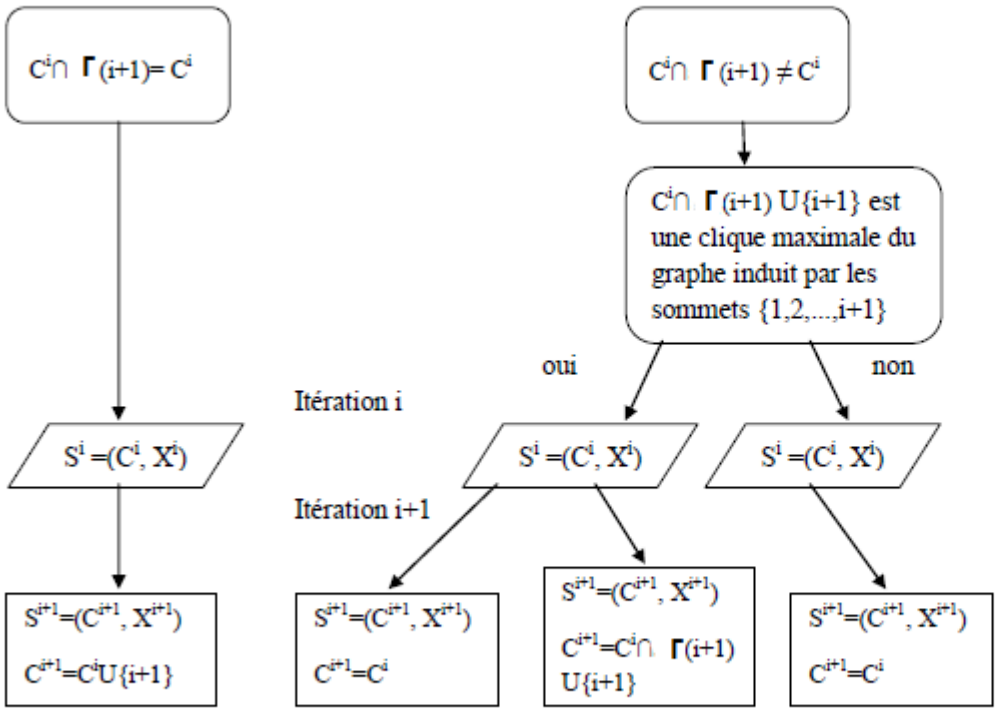


FIGURE 4.4 – Schéma représentant l'algorithme de Tsukiyama

Remarque 4.1.

1. *Pour $i = n$, $X^n = \emptyset$, et dans ce cas toutes les cliques maximales du graphe seront générées.*
2. *Cette méthode nous permet d'avoir un arbre de n itérations dont les feuilles (sommets pendant) sont les cliques maximales du graphe G.*

4.4.3 Exemple d'application

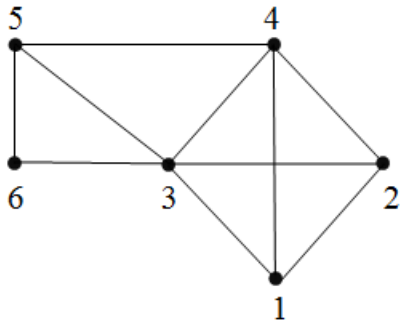


FIGURE 4.5 – Graphe de l'exemple d'application

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

$S^i = (1234, (5, 6))$ avec $C^i = \{1234\}$ et $X^i = \{5, 6\}$.

$1234 \cap \Gamma(5) = 1234 \cap 346 = 34 \neq 1234$
$\Rightarrow \{1234\}$ est une clique maximale du graphe induit par les sommets $\{1, 2, 3, 4, 5\}$.
donc $S^{i+1} = (1234, (6)) = 12346$

$1234 \cap \Gamma(5) \cup \{5\} = 1234 \cap 346 \cup \{5\} = 345$
$\Rightarrow \{345\}$ est une clique maximale du graphe induit par les sommets $\{1, 2, 3, 4, 5\}$.

$1234 \cap \Gamma(6) = 1234 \cap 35 = 3 \neq 1234$
$\Rightarrow \{1234\}$ est une clique maximale du graphe induit par les sommets $\{1, 2, 3, 4, 5, 6\}$.
donc $S^{i+1} = (1234, \emptyset) = 1234$

$1234 \cap \Gamma(6) \cup \{6\} = 1234 \cap 35 \cup \{6\} = 36$
$\Rightarrow \{36\}$ n'est pas une clique maximale.

$S^i = (345, (6))$ avec $C^i = \{345\}$ et $X^i = \{6\}$.

$345 \cap \Gamma(6) = 345 \cap 35 = 35 \neq 345$
$\Rightarrow \{345\}$ est une clique maximale du graphe induit par les sommets $\{1, 2, 3, 4, 5, 6\}$.
donc $S^{i+1} = (345, \emptyset) = 345$

$345 \cap \Gamma(6) \cup \{6\} = 345 \cap 35 \cup \{6\} = 356$
$\Rightarrow \{356\}$ est une clique maximale du graphe induit par les sommets $\{1, 2, 3, 4, 5, 6\}$.

$S^i = (356, \emptyset)$
Fin de l'exécution car $X^i = \emptyset$ et $i = 6$.

Donc toutes les cliques maximales du graphe sont générées, et qui sont : $\{1234\}$; $\{345\}$ et $\{356\}$.

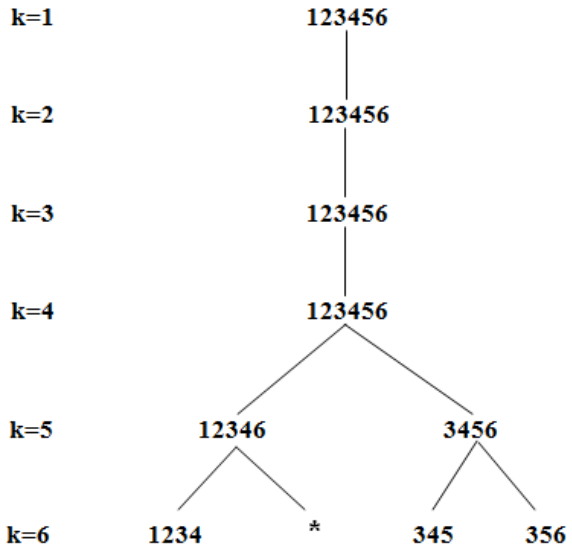


FIGURE 4.6 – Arbre obtenue par le déroulement de l'algorithme de Tsukiyama et Al

$*$ signifie que la condition de la clique maximale n'est pas vérifiée.

Le tableau suivant résume les différents algorithmes étudiés dans ce chapitre :

Algorithme	But	Critères	Complexité
Lex-BFS	Ordonnancement des sommets d'un graphe	- Ordre lexicographique	$O(m+n)$
Johnson et al	Génération de la première clique maximale C_1	- Ordre lexicographique - Temps polynomial	$O(m+n)$
Johnson et al	Génération d'autres cliques maximale à partir de C_1	- Ordre lexicographique - Temps polynomial	$O(n^3)$
Tsukiyama et al	Génération des cliques maximales d'un graphe G	- Espace polynomial - Délai polynomial	$O(n.m)$

4.5 Relations entre treillis et cliques maximales

Après avoir étudier l'algorithme de construction d'un graphe de couverture d'un treillis et les deux algorithmes qui génèrent les cliques maximales dans un graphe on obtient les relations entre un treillis et clique maximale suivantes :

1. Les couvertures d'un treillis sont des cliques maximales.
2. Les concepts d'un treillis de Galois sont des cliques maximales.

Chapitre 5

Implémentation

Ce chapitre traite du langage de programmation utilisé pour coder l'algorithme de L.Nourine étudié dans le cadre de ce travail.

5.1 Logiciel d'application

Un programme est un ensemble de blocs de codage qui décrit un ensemble d'instructions pour résoudre un problème donné dans un langage de programmation. on utilise DEV-C++ comme langage de programmation.
Le C++ est un langage de programmation permettant la programmation sous de multiples paradigmes comme la programmation procédurale, la programmation orientée objet et la programmation générique.
C++ est actuellement le 3^{e} langage le plus utilisé au monde. Le langage C++ n'appartient à personne et par conséquent n'importe qui peut l'utiliser sans besoin d'une autorisation ou obligation de payer pour avoir le droit d'utilisation.

5.2 Mon premier programme

```cpp
#include <iostream>
using namespace std;

int main()
{
    cout << "BONJOUR" << endl;

    return 0;
}
```

FIGURE 5.1 – Mon premier programme C++

Après avoir taper le code de la figure 5.1 dans l'éditeur du texte, on compile et on exécute pour avoir le résultat suivant :

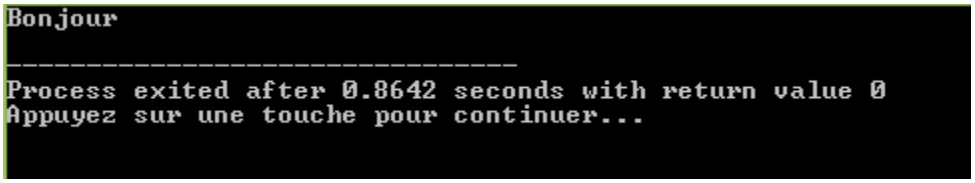


FIGURE 5.2 – Execution de mon premier programme C++

5.2.1 Explications

La directive `#include` :

La directive de compilation `#include <iostream>` permet d'inclure les prototypes des différentes classes contenues dans la bibliothèque standard iostream. Cette bibliothèque contient la définition de `cout` qui permet entre autre d'afficher des messages à l'écran.

L'espace de nommage standard :

Un espace de nommage peut être vu comme un ensemble d'identifiants C++ (types, classes, variables etc.). **cout** fait partie de l'espace de nommage **std**. Pour parler de l'objet prédéfini cout de l'espace de nommage std, on peut écrire std : :cout. Cette notation est assez lourde car elle parsème le code de std : :. Pour cela on a écrit **using namespace std** qui précise que, par défaut, la recherche s'effectuera aussi dans l'espace de nommage std. On pourra donc alors écrire tout simplement cout pour parler de std : :cout. On dit que std est un espace de nom (namespace en anglais).

La fonction main :

Tout programme en C++ commence par l'exécution de la fonction main. Il se termine lorsque la fonction main est terminée. La fonction main peut être vue comme le point d'entrée de tout programme en C++. Cette fonction renvoie un entier, très souvent 0, qui permet d'indiquer au système d'exploitation que l'application s'est terminée normalement.

L'objet cout :

Il permet d'envoyer des caractères vers le flux de sortie standard du programme, c'est-à-dire l'écran pour ce programme (sa fenêtre console). Il permet donc d'afficher des messages à l'écran. En utilisant l'opérateur `<<`, on peut écrire une chaîne de caractères à l'écran. L'instruction `cout << "BONJOUR";` affiche donc le message **BONJOUR** à l'écran.

return 0 :

Cette instruction (facultative ici) indique que la fonction main est terminée et que tout s'est bien passé. Nous verrons plus loin ce que veut dire exactement l'instruction `return`, mais même si elle est facultative il est fortement recommandé de la mettre, que ce soit par simple souci de conformité ou du fait que votre programme est sensé renvoyer une valeur à la fin de son exécution.

Nous avons essayé de programmer l'algorithme de L.Nourine pour la reconnaissance d'un treillis mais avant de présenter les différentes étapes d'exécution nous allons définir d'abord la liste des structures de données utilisées dans la représentation d'un graphe.

5.3 Structure de données

1. **std : :vector**
   std : :vector <**type**> où type est le type de données qui sera utilisé dans le vecteur.
   std : :vector contient plusieurs méthodes de manipulation, parmi ces méthodes :

   push-back(val) : pour insérer une valeur à la fin du vecteur.
   $erase(du, au)$: pour supprimer à partir de (du) jusqu'à (au).
   pop-back() : pour supprimer le dernier élément

   Voici un exemple d'utilisation d'un vecteur :

```
#include<vector>
#include<iostream>


main::main()
{
    std::vector<int> vect;  //definer un vector de 10 elements
    for(int i=0;i<10;i++)   //inserer 10 entier dans le vecteur
        vect.push_back(i);  //


    std::vector<int>::iterator it; //un iterateur pour parcourir le vecteur

    for(it=vect.begin();it!=vect.end();++it)
    {
        std::cout<<*(it);   //afficher le contenu de l'iterateur
    }


}
```

FIGURE 5.3 – Utilisation d'un vecteur en c++

2. **std : :list**
   std : :list est comme **std : :vector** , juste **std : :list** contient plus de méthodes comme :
   push-front(val) : pour insérer une valeur de l'avant.
   pop-front() : pour supprimer le premier élément.

3. **std : :queue (La queue)**
   std : :queue est comme **std : :liste**, mais ne contient pas d'itérations, elle contient seulement les méthodes
   push(val) : pour insérer une valeur à la fin.
   pop() : pour supprimer le premier élément de la queue.

4. **std : :stack (La pile)**
   std : :stack est comme **std : :queue**, ne contient pas d'itérations, et contient plusieurs méthodes de manipulation comme :
   $push(val)$: pour insérer l'élément a la fin de la pile.
   $pop()$ pour supprimer le dernier élément de la pile.

5. **std : :unordered-map**
   std : :unordered-map est un conteneur qui sauvegarde des éléments par la commination des (key,Value) ou (Clé,Valeur).

5.4 Exemple d'application

Dans notre programme on va introduire les données suivantes :

1. L'ordre du graphe.
2. Nombre de noeuds.
3. Les valeurs des noeuds.
4. Le nombre d'arcs.
5. La direction des arcs.

Soit le graphe d'application suivant :

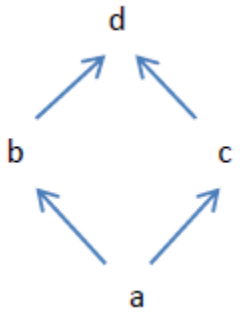


FIGURE 5.4 – Exemple d'application

Ce graphe sera représenté par sa liste d'adjacence, ensuite on calculera l'ensemble de tous les successeurs de chaque noeud.
L'extension linéaire est choisie par l'utilisateur.

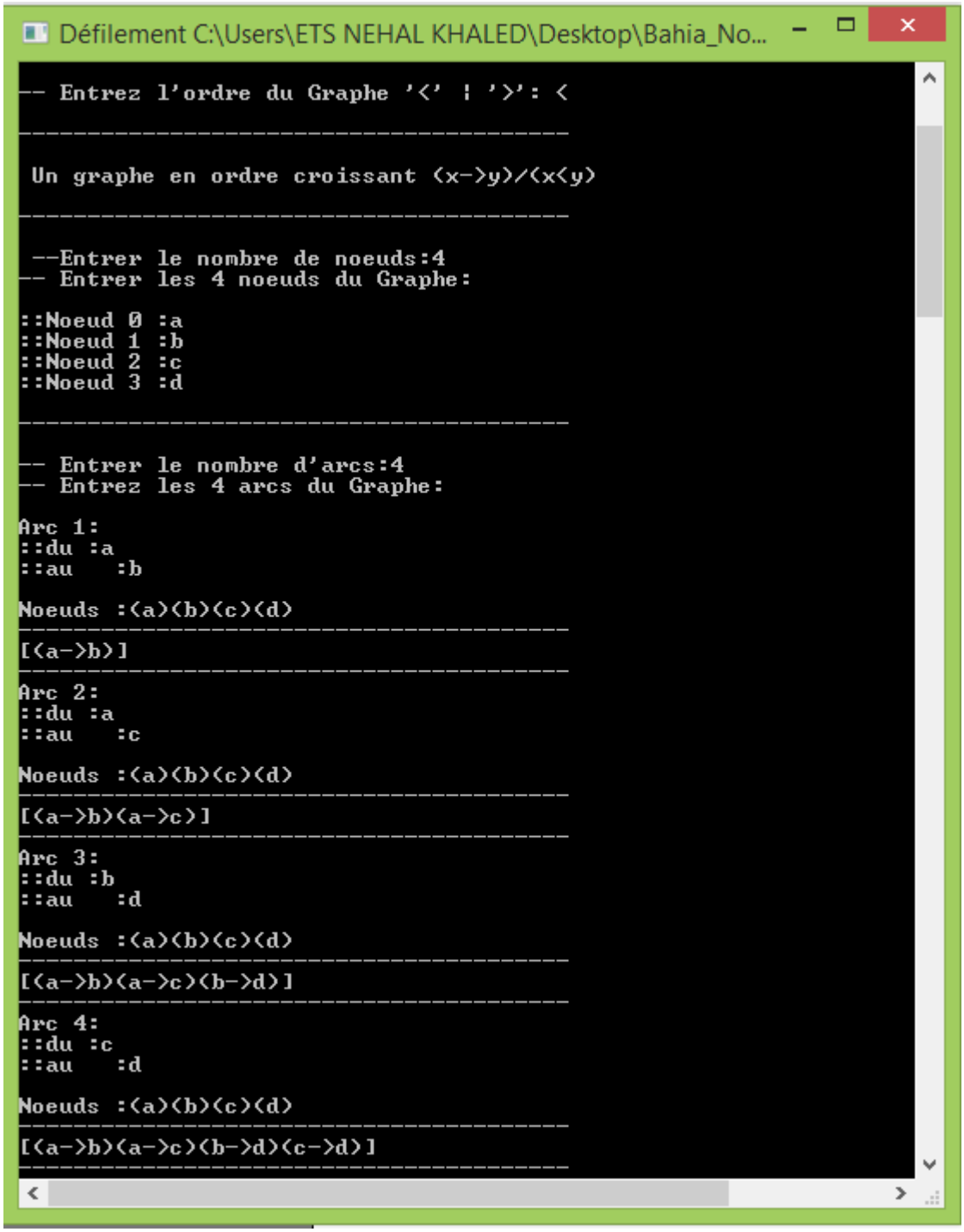
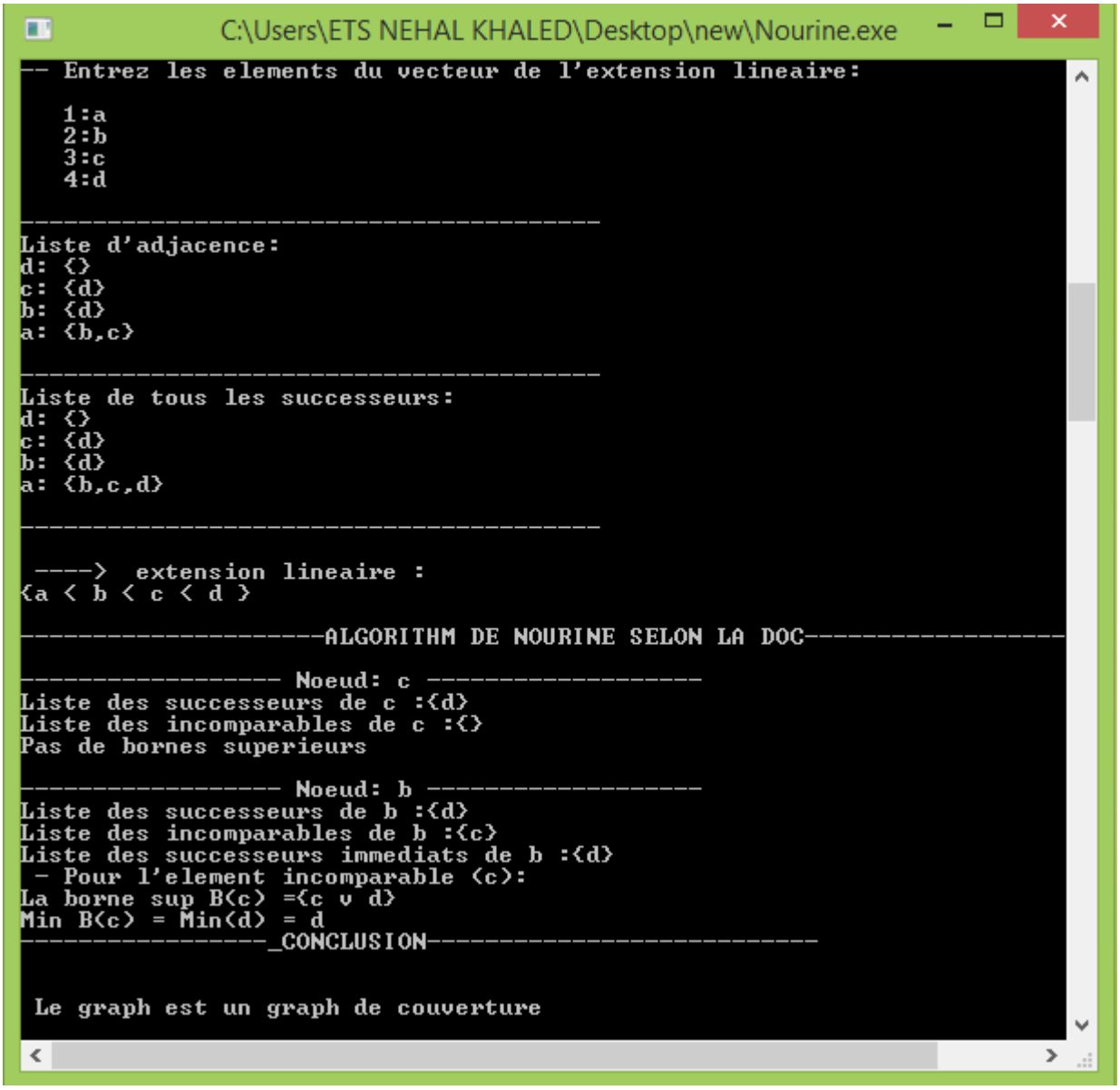


FIGURE 5.5 – Les données du graphe

Et maintenant, on affichera l'exécution de notre programme.



Conclusion générale et perspectives

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.

L'objet principal de notre travail est de faire une étude algorithmique sur les treillis et les cliques maximales et de déduire les relations entre eux.
Pour les algorithmes de génération des cliques maximales, nous avons vu qu'aucun des deux n'a pu respecter les trois critères qui sont le délai polynomiale, l'espace mémoire polynomiale et la génération des cliques suivant un ordre lexicographique.
Concernant les algorithmes liés aux treillis, nous avons présenté aussi deux algorithmes celui de L.Nourine qui reconnait un treillis et celui de L.Nourine et O.Raynaud qui construit un graphe de couverture à partir d'une base composée par des éléments de X.
Après cette étude algorithmique, nous avons pu déduire que les couvertures d'un treillis sont des cliques maximales ainsi que les concepts d'un treillis de Galois en particulier.

Comme perspectives, il serait souhaitable d'implémenter un algorithme qui génère et construit un treillis d'une classe particulière en énumérant ses cliques maximales.

Bibliographie

[1] Claude Berge. Livre : Graphes et Hypergraphes ed.Dunod,Paris.

[2] Belhadj Abdelaziz. Thèse : Génération de treillis et propriétés algébriques, UMMTO 03/11/2011.

[3] Karell Bertet. Mémoire : Structure de treillis, Université de La Rochelle 14/06/2011.

[4] Michel Cramps-Michel Plantié-Julien Bidault. Article : Cliques maximales d'un graphe et treillis de Galois, Laboratoire de Génie Informatique et d'Ingénierie de Production (LGI2P) 2011.

[5] Michel Habib. Cours d'algorithmique des graphes du MPRI Algorithme LexBFS, 23/01/2009.

[6] Louadj Kahina. Thèse : Génération des cliques maximales dans un graphe non orienté, UMMTO 08/06/2005.

[7] Lynda Oubakouk.Thèse : Génération des cliques maximales : Etude du graphe de transition, UMMTO.

[8] Lhouari Nourine, Olivier Raynaud. Article : A fast algorithm for building lattices, Information Processing Letters 71 (1999) 199–204.

[9] Abbadja Dahbia, Ammouche Melha. Mémoire : Représentation des sous-treillis polyedraux et généraux et sous-treillis des produits d'espaces, UMMTO 30/09/2015.

[10] Medjouti Norredine, Zeher Yacine. Mémoire : Sous-treillis de produit des espaces :Enveloppants, représentations et dénombrement, UMMTO 2015.