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

Université : Mouloud Mammeri, Tizi-Ouzou

Faculté des sciences

Département de Mathématiques

Mémoire de Magistère

Spécialité

Mathématiques

Option

Recherche Opérationnelle et optimisation

Thème

Génération de treillis et propriétés algèbriques

Présenté par

Mr BELHADJ Abdelaziz

Soutenu devant le jury composé de :
Mr **AIDENE Mohamed**, Professeur, UMMTO. Président
Mr **OUKACHA Brahim**, Maître de conférence (A). UMMTO, Rapporteur
Mr **SADI Bachir**, Maître de conférence (A). UMMTO, Examinateur
Mr **AIDER Meziane**, Professeur, USTHB., Examinateur
Mr **BERRACHEDI Abdelhafidh**, Professeur, USTHB. Examinateur

Soutenu le : 03/11/2011

A mes parents que j'aime très fort.
A ma femme et mes enfants que j'adore.
A mes frères et sœurs.
A mes beaux parents que j'estime et que je respecte beaucoup.
A la mémoire de mes grands parents.
A toute ma famille.

C'est le thème du topos qui est ce lit où viennent s'épouser la géométrie et l'algèbre, la topologie et l'arithmétique, la logique mathématique et la théorie des catégories, le monde du continu et celui des structures discontinues ou discrètes . Il est ce que j'ai conçu de plus vaste, pour saisir avec finesse, par un même langage riche en résonnances géométriques, une essence commune à des situations des plus éloignées les unes des autres.

A. Grothendieck

Remerciements

Mes toutes premières expressions de gratitude vont à l'encontre de mon directeur de mémoire **Mr B. OUKACHA**, qui m'a dirigé avec rigueur et application ; il a su manifester intérêt et confiance tout en me redonnant le goût des études que j'aime tant. Ainsi, je le remercie pour son soutien moral, sa disponibilité, ses qualités personnelles et son amabilité.

Je remercie Messieurs **M. AIDENE, M. AIDER, B. SADI et A. BERRACHEDI** d'avoir accepté de prendre part au jury qui évaluera mon travail. Leur renommée dans le milieu des mathématiques en général et de la Recherche Opérationnelle en particulier, confère à leurs positions d'examinateurs une grande qualité, ainsi, leur présence m'honore beaucoup.

Je remercie de tout coeur mes très chers enseignants de m'avoir donner le courage, le savoir, la science et la volonté de poursuivre mes études malgré tous les obstacles et les difficultés rencontrés.

Je suis particulièrement reconnaissant à tous mes enseignants qui ont participé à mon cursus scolaire et à mon éducation depuis l'école primaire jusqu'à présent, ils ont su m'inculquer le savoir, la science, la discipline, la rigueur, la réflexion et les bonnes manières.

Un grand merci à tous mes amis et collègues de travail , qui m'ont aidé et soutenu, en particulier A.Kahlouche, O.Lamrous, B.Smail, D.Talem, M.Hamidani, H.Abed, A.Mendes, M.Messaoudi, M.Aomari, A.Djaou, H.Bouzar, O.Elkechai, M.Hamizi, S.Mansouri,

R.Hamdad, A.Sadani, M.Taflis, sans oublier le grand Monsieur HACID que j'éstime tellement.

Je salue affectueusement et chaleureusement mes respectables et respectueux parents et beaux parents, ma brave femme et mes adorables enfants **(Camélia et Mehdi)**, mes frères, beaux frères, soeurs et belles soeurs qui m'ont toujours fait confiance et soutenu, et enfin, à tout ce beau monde je dedie ce modeste travail.

Abdelaziz Belhadj

Table des matières

Introduction générale

Depuis des centaines d'années, l'homme a cherché à résoudre de nombreux problèmes ; mais la conclusion de ces recherches était, la majorité du temps, infructueuse et les objectifs très durs, voir impossible à atteindre sans méthode rigoureuse.
L'être humain, a ainsi creé et utilisé, des outils l'aidant à calculer, et à automatiser diverses tâches afin d'aboutir à une solution, ou de montrer que celle-ci, n'est pas atteignable.

Ainsi, au dix-huitième siècle, la logique a fait son apparition grâce aux travaux de plusieurs chercheurs comme Boole, Peirce, Schroder, Dedekind, Skolem, Menger etc..., la théorie des treillis a longtemps été négligée, avant de prendre son essor à partir des années 30, grâce à des auteurs et chercheurs comme Klein, Birkhoff, Ore, Stone, Kurosh , Tarski , Glivenko, von Neumann, etc. . ., la transformant en branche fertile de l'algèbre.

C'est à cette époque qu'on découvre que la structure de treillis se retrouve dans de nombreux domaines mathématiques tels que : l'algèbre, la géométrie, la logique, l'analyse fonctionnelle, etc. . . ; mais ce n'est que vers les années 60 que l'aspect combinatoire des treillis va se développer, en liaison avec la naissance de l'informatique et donc l'accroissement des besoins en informatique, algorithmique, programmes et combinatoire.

Malgré les avancées technologiques et l'augmentation permanente de puissance des machines, certains problèmes nécessitent toujours un temps de calcul qui n'est pas envisageable en terme de traitement pratique.

Un des facteurs prépondérants est l'augmentation incessante de la taille des données à traiter ; C'est l'explosion combinatoire, car ; il suffit de considérer un ensemble de données de cardinalité n pour constater que l'ensemble des parties comporte 2^n éléments ce qui représente un nombre exponentiel de configurations possibles et donc un ensemble très large de possibilité de réponses et de solutions à tester.

De ce fait, de nombreux problèmes de recherche restent encore à ce jour ouverts, c'est à dire ne possèdent pas d'algorithmes efficaces permettant d'arriver aux solutions souhaitées.

Dans ce mémoire, nous nous intéressons particulièrement aux algorithmes de génération de quelques classes de treillis ainsi qu'à leurs propriétés algèbriques. Celles ci existent en nombre important, pour cela, nous ne pouvons les étudier toutes vu non seulement leur diversité, leur complexité mais aussi leur étude et conception algorithmiques.

Ce document est organisé de la manière suivante :
Le premier chapitre est constitué des définitions et des concepts de base ; à ce niveau, nous rappelons les notions et outils de base de la théorie des graphes et des ensembles ordonnés, nous présentons les concepts et le vocabulaire permettant de définir et représenter un graphe et un ensemble ordonné en proposant pour chacun un exemple illustratif.

Le second chapitre consiste à définir et à étudier les treillis en général et quelques classes de treillis en particulier tout en énumérant certaines propriétés algèbriques importantes qui les identifient et les caractérisent.

Le dernier chapitre fera l'objet de l'étude algorithmique de la génération et de la construction des treillis , en citant comme exemple deux algorithmes de L.Nourine et un autre de Genall qui consiste à améliorer celui de L.Nourine sur la construction et la génération du treillis de concept.

On déroulera une application sur l'un des algorithmes afin de pouvoir discuter sa complexité ; puis on procédera à une évaluation en terme de complexité spatiale et temporelle pour enfin désigner et selectionner le meilleur et le plus performant, tout en portant les critiques nécéssaires.

En perspectives, et en tenant compte des études et recherches actuelles, on essaiera de cerner pratiquement toutes les propriétés algèbriques existantes qui sont très nombreuses, puis ; proposer un algorithme de génération et de construction de treillis ou d'une certaine classe de treillis plus consistant dont la complexité est meilleure, pour pouvoir enrichir et rapprocher au mieux ces treillis du domaine informatique pour une utilisation rationnelle dont les domaines d'application sont sans aucun doute très nombreux. Il est très important, et primordial de savoir que l'algorithmique des treillis est actuellement à l'état embryonnaire.

Chapitre 1

Définitions sur les graphes et ordres

Afin de faciliter au lecteur la compréhension et la lecture du contenu de ce travail, nous avons jugé utile de donner quelques définitions et orientations de base concernant les graphes et les ensembles ordonnés.

1.1 Graphes

Soit X un ensemble fini. Nous désignons par $|X|$ le cardinal de X, de plus $P(X)$ désigne l'ensemble des parties de X , et, pour $0 \leq k \leq |X|$, $P_k(X)$ désigne l'ensemble des parties de X ayant k éléments.

- Il est bien connu que $|P(X)| = 2^n$ si $|X| = n$, dans ce cas on a évidemment $P_0(X) = \emptyset$ et $P_n(X) = \{X\}$.
- Les éléments de $P_1(X)$ sont appelés des boucles et les éléments de $P_2(X)$ des arêtes.

- Un graphe G, est défini par un ensemble fini non-vide X, appelé ensemble des sommets de G et un ensemble E de paires de sommets, appelé ensemble des arêtes de G.
- On a donc $E \subseteq P_2(X)$; et on écrit $G = (X, E)$.
- Lorsque $e = (x, y)$, on dit que e est l'arête de G d'extrémités x et y ; ou que e joint x et y ; ou que e passe par x et y.
- Les sommets x et y sont alors **adjacents** dans G et on dit qu'ils sont **incidents** à l'arête e.
- Un graphe simple ne possède ni arêtes multiples ni boucles.
- Le nombre de sommets $|X|$ du graphe simple $G = (X, E)$ s'appelle **ordre** de G. On le note $|G|$ ou ord G. Le nombre d'arêtes du graphe est noté par $|E|$.
- On représente le graphe simple G par une figure dans le plan. Les sommets de G sont représentés par des points du plan et ses arêtes par des traits entre ces sommets.
- Un graphe simple $G = (X, E)$ où $X = \{x, y, z, u, v\}$ et $E = \{(x, y), (x, u), (y, z), (y, u)\}$ peut se représenter par la figure suivante :

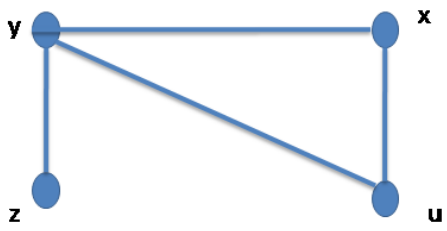

FIG. 1.1 – Exemple de Graphe

- Dans un graphe simple $G = (X, E)$, une chaîne C est une séquence finie de sommets $x_0, x_1, ..., x_p$ telle que
  $\forall\ 0 \leq i \leq p$, $(x_i, x_{i+1}) \in E$; on écrit $C = [x_0, x_1, ..., x_p]$.
- $p = l(C)$: longueur de la chaîne.
- x_0 et x_p sont les extrémités respectivement initiale et terminale de la chaîne C.
- Lorsque $x_0 = x_p$, on dit que la chaîne est fermée et on l'appelle **cycle**.
- Une chaîne de longueur ≥ 1 dont toutes les arêtes (x_i, x_{i+1}) sont distinctes est dite **simple**.
- Une chaîne dont tous les sommets x_i sont distincts est dite **élémentaire**.
- Un graphe G est **connexe** si $\forall x, y \in X$, il existe une chaîne de x à y.
- Sur l'ensemble X des sommets du graphe simple G, on définit la relation d'équivalence suivante :
  $x \Re y \Leftrightarrow$ il existe une chaîne de x à y dans G.
- Soient $X_1, X_2, ..., X_k$ les classes d'équivalence de $\Re$, pour $1 \leq i \leq k$, $G_i = G_{X_i}$ le sous-graphe de G engendré par X_i.
- Les graphes simples G_i sont appelés composantes connexes de G.
- Un graphe orienté $G{=}(X, U)$, est un graphe formé d'un ensemble fini non vide X de sommets et d'un ensemble $U \subseteq X * X$ d'arcs.

- Si $u = (x, y) \in U$, alors on dit que x est l'extrémité initiale de u et y l'extrémité finale de u. On dit aussi que u est un arc de x à y.
- Si $u = (x, x)$, alors u est une **boucle** en x .
- Etant donné un graphe simple $G{=}(X, E)$, on lui associe un graphe orienté (X, U) où $(x, y) \in U \Leftrightarrow (x, y) \in E$; où chaque arête $(x, y) \in E$ donne deux arcs (x, y) et (y, x) dans U.
- Inversement , à tout graphe orienté $G{=}(X, U)$, on associe un graphe simple $G = (X, E)$, où $(x, y) \in E \Leftrightarrow x \neq y$ et $(x, y) \in U$ ou $(y, x) \in U$.
- Un chemin de $G{=}(X, U)$ est une séquence de sommets $C = [x_0, x_1, ..., x_n]$, telle que pour $i = 0, 1, ..., n - 1$, on ait $(x_i, x_{i+1}) \in U$.
- On dit que x_0 est l'extrémité initiale, x_n est l'extrémité terminale et n la longueur du chemin C.
- Si $x_0 = x_n$, on dit que le chemin est fermé et on l'appelle **circuit**.
- Le chemin C est dit **élémentaire** si les sommets x_i sont distincts.
- Il est dit **simple** si les arcs u_i sont distincts.
- G est **fortement connexe** si : $\forall x, y \in X$ il existe un **chemin** de x à y et un **chemin** de y à x

- Soit $G = (X, U)$ un graphe orienté sans circuit. On dit que (x, y) est un **arc de transitivité** s'il existe un sommet z tel que (x, z) et (z, y) sont des arcs de G.
- A tout graphe, on peut associer de façon unique un **graphe transitif**, $\hat{G} = (X, \hat{U})$, appelé **fermeture transitive** de $G = (X, U)$, où $\hat{U}$ est définie par la relation d'appartenance suivante : $(x, y) \in \hat{U} \Leftrightarrow \exists$ dans G un chemin de x à y.
- On appelle **fermeture transitive** de G, le graphe obtenu en ajoutant à G tous les arcs de transitivité.
- On appelle **réduction transitive de** G, le graphe obtenu en retirant à G tous les arcs de transitivité.
- Soit $G = (X, U)$ un graphe orienté, dont les sommets et les arcs sont numérotés comme suit : $i = \overline{1, n}$ et $j = \overline{1, m}$.
- **La matrice d'adjacence** associée au graphe G, est une matrice carrée.

$$A = (a_{ij})\ ;\ i, j = \overline{1, n},\ \text{où}$$

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

- **La matrice d'incidence** associée au graphe G, est une matrice $M = (m_{ij})$, où $i = \overline{1, n}$, $j = \overline{1, m}$.

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

1.2 Ensembles ordonnés

- Une relation $\Re$ définie sur un ensemble X est appelée **relation d'ordre** si pour tout $x, y, z \in T$, on a les propriétés suivantes :
  1. Réflexivité : $x \Re x$;
  2. Antisymétrie : $x \Re y$ et $y \Re x \Rightarrow x = y$;
  3. Transitivité : $x \Re y$ et $y \Re z \Rightarrow x \Re z$.
- Un **ensemble ordonné** ou **ordre** 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).
- On note $P = (X, \leq_p)$ ou P, l'ensemble ordonné ou ordre.
- Deux éléments x et y de X sont dits **comparables** si $x \leq_P y$ ou $y \leq_P x$ avec l'intérprétation usuelle, sinon ils sont dits **incomparables**, ils sont alors notés $x \| y$.
- Les relations d'ordres telles qu'elles sont définies dans leur généralité, ne sont ni propices à une représentation en machine ni optimisées pour en extraire les caractéristiques.
- 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 $(x, y) \in X \times X$ 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 y **couvre** x (y est **successeur immédiat** de x), x **couvert** par y (x est **predecesseur immédiat** de y).
– Un couple $(x, y) \in X \times X$ est un **saut** si x et y sont incomparables.

– **Graphe de couverture :**
On peut schèmatiser 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 1.1. $X = \{\{2\}, \{3\}, \{2,4\}, \{2,3,6\}, \{2,4,3\}\}$.
Les couples de $P = \{(A, B), A \in XetB \in X/A \subseteq B\}$.
Voici le diagramme de P :
Les sauts de P :$(\{2\}, \{3\}), (\{3\}, \{2,4\}), (\{2,3,6\}, \{2,3,4\})$ $(\{2,4\}, \{2,3,6\})$.
Les couvertures de P : $(\{2\}, \{2,4\}), (\{2\}, \{2,3,6\})$, $(\{3\}, \{2,3,6\}), (\{3\}, \{2,4,3\}), (\{2,4\}, \{2,3,4\})$.

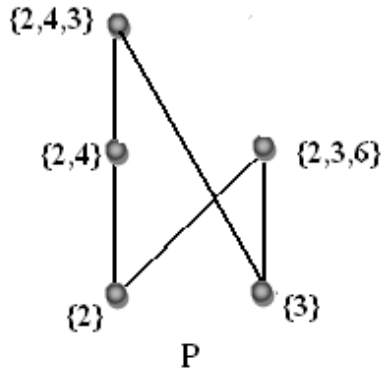

FIG. 1.2 – Diagramme de P

- L'ensemble des éléments n'ayant aucun prédécesseur (resp. succésseur) est noté $Min(P)$(resp.$Max(P)$).
- A tout ordre $P = (X, \leq_p)$, on peut associer un graphe orienté $G = (X, U)$ avec $U = \{(x, y)/x <_P y\}$.
- La réduction transitive du graphe associé à P est appelée **graphe de couverture de** P ou **diagramme de Hasse**.
- Une **chaîne** de P est un sous-ensemble d'éléments de X deux à deux comparables.
- Elle a pour **longueur** le nombre de ses éléments.
- On note $d(P)$ la longueur de la plus grande chaîne de P.
- La **hauteur** de P est la quantité $h(P) = d(P) - 1$, c'est la longueur d'une chaine de cardinal maximum.
- Une **antichaîne** est un sous-ensemble d'éléments de X deux à deux incomparables.

- Une chaîne (resp. antichaîne) de P est dite **maximale** si elle n'est pas strictement incluse dans une autre chaîne (resp. antichaîne).
- La **largeur** de P est le **cardinal maximum** d'une antichaine ; On la note $W(\mathrm{P})$.
- Un ordre **total** ou **chaîne** est un ensemble d'éléments deux à deux comparables.
- $Q = (X, \leq_Q)$ est une **extension** de $P = (X, \leq_P)$ si $x \leq_P y \Longrightarrow x \leq_Q y$ 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.
- On note alors $L(P)$ l'ensemble des extensions linéaires de l'ordre P.
- On dit qu'un ordre $Q = (Y, \leq_Q)$ est un **sous-ordre** de P si et seulement si $Y \subseteq X$ et pour tout x,y de Y $x \leq_Q y$ si et seulement si $x \leq_P y$.
- Un **Idéal** I d'un ordre P, est une **partie héréditaire inférieure**, i.e. $I \subseteq X$ tel que si $x \in I$ et $y \in X$ vérifient $y \leq_P x$ alors $y \in I$.
- On note par $I(P)$ l'ensemble des idéaux de P.
- Un Idéal est dit **Principal** s'il existe un élément $a \in X$ tel que $I=\{x \in X / x \leq_P a\}$, on note alors $I =\downarrow a$.
- Les notions de **filtre** et de filtre **principal** ou **partie héréditaire supérieure** peuvent se définir dualement.

Considérons un ordre $P = (X, \leq_P)$:

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

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

– 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. $x <_P y$ et $x <_P z$.
2. Pour tout $t \in X$, tel que $t <_P y$ et $t <_P z$ alors $t <_P x$.

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

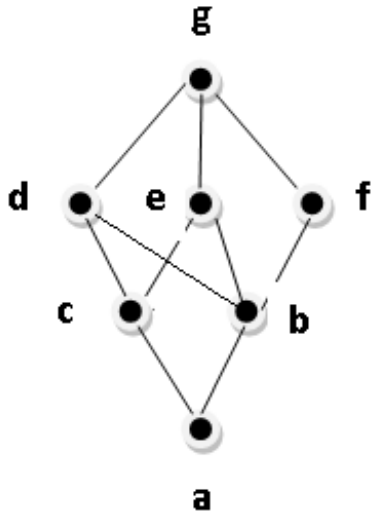

FIG. 1.3 – Ensemble ordonné ou Ordre

Chapitre 2

Une classe d'ordres partiels : les treillis

Dans ce chapitre ; nous allons nous intéresser spécialement au cas d'une classe d'ordre **(les treillis)**, qui a particulièrement intéressé et occupé les informaticiens sur les aspects et les retombées algorithmiques ; ainsi que sur les différents domaines d'application.

2.1 Treillis

Un ordre partiel $P = (X, \leq_p)$ non vide est dit **treillis** si pour tout couple d'éléments x et y de X, la borne supérieure $x \vee y$ et la borne inférieure $x \wedge y$ existent dans P.

- Un treillis est dit **complet** lorsque tout 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-ensembles 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 élément minimum et $\top = \vee X$ son élément maximum, appelé aussi élément **universel**.
- Notons que le dual d'un treillis est aussi un treillis.
- Tout treillis sur un ensemble fini est un treillis complet.

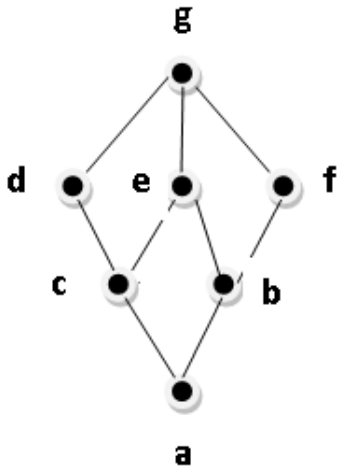

FIG. 2.1 – Treillis

- Un ordre partiel $P = (X, \leq_p)$ 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.

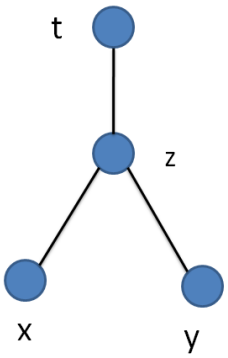

FIG. 2.2 – Sup-demi-treillis

– Un ordre partiel $P = (X, \leq_p)$ 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.

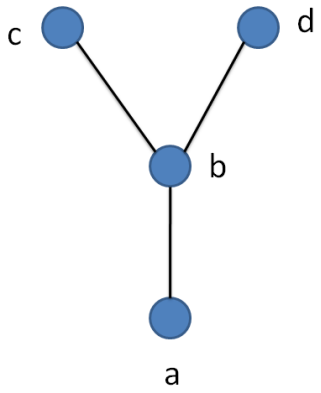

FIG. 2.3 – Inf-demi-treillis

– 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 treillis est un ensemble partiellement ordonné qui est à la fois un inf-demi-treillis et un sup-demi-treillis.
- 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 est un sup-irréductible si et seulement si il couvre un seul élément dans le graphe de couverture associé à T.
- Par convention $\perp$ qui est l'élément minimum noté par $\perp = \wedge X$ et $\top$ qui est l'élément maximum noté par $\top = \vee X$; appelé aussi élément **universel**.
- 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 un inf-irréductible si et seulement si il est couvert par un seul élément dans le graphe de couverture associé à T.
- L'ensemble des éléments sup-irréductibles est noté par $\mathbf{J(T)}$.
- L'ensemble des éléments inf-irréductibles 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 à 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 $[x, y]_T$.

Quelques exemples de treillis :

– L'ensemble des parties d'un ensemble muni de l'inclusion forme un treillis où la borne supérieure est l'union et la borne inférieure est l'intersection.
– L'ensemble des ouverts d'un espace topologique muni de l'inclusion forme un treillis.
– L'ensemble des entiers naturels muni de son ordre usuel est un treillis.
– L'ensemble des entiers naturels muni de la relation "divise" forme un treillis, où la borne supérieure est le **PPCM** et la borne inférieure le **PGCD**.
C'est un treillis borné **(l'élément minimum est 1, et l'élément maximum est 0)**.
– Notons que $I(P)$ l'ensemble des idéaux de P et $A(P)$ L'ensemble des antichaînes maximales de P sont tous deux des treillis.

2.2 Différentes classes de treillis

Les treillis quelconques en général étant définis, on passera aux différentes classes de treillis tirées de [2,3,11,17], 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, Pseudo-complémentés, Brouwériens, de Stone, de Post, Booléens, de Galois ....etc. Chacune de ces classes a sa particularité et possède des propriétés algèbriques importantes et interessantes ; notre étude sera basée sur un nombre restreint de classes.

2.3 Propriétés algèbriques des treillis

Que ce soit pour le treillis quelconque, ou pour les différentes classes de treillis, il existe un nombre important de propriétés algèbriques, qui les caractérisent et les définissent. Il serait interessant aussi, de citer les différents liens existants entre certaines de ces classes, ainsi que leur domaines d'application.

2.3.1 Propriétés algèbriques d'un treillis

Précédemment, on avait donc définit le treillis en général d'une manière ensembliste, [2,3,30,33] sachant, qu'il existe une définition purement algèbrique contenant quelques propriétés très importantes qu'on définira et qu'on démontrera par la suite :

Théorème 1 :[16] 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 $\vee$ et $\wedge$ possèdent les propriétés algèbriques suivantes :

1. L'**idempotence** :
   $x \vee x = x.$ ou $x \wedge x = x.$
2. La **commutativité** :
   $x \vee y = y \vee x.$ ou $x \wedge y = y \wedge x.$
3. L'**associativité** :
   $x \vee (y \vee z) = (x \vee y) \vee z$.
   $x \wedge (y \wedge z) = (x \wedge y) \wedge z.$
4. L'**absorption** :
   $x \vee (x \wedge y) = x.$ ou $x \wedge (x \vee y) = x.$
5. Et une relation d'ordre $\leq$ telle que :
   $x \leq y \Leftrightarrow x \vee y = y \Leftrightarrow x \wedge y = x.$

Démonstration : Soient $x, y, z \in T$

1. Idempotence :

   $x \vee x = x = x \wedge x$

   Cette propriété découle de la définition de la relation d'ordre, le plus petit des majorants (ppM) de x est x et le plus grand des minorants (pGm) de x est x.

2. Commutativité :

   $x \vee y = y \vee x$ et $x \wedge y = y \wedge x$.

   Par hypothèse $x \vee y$ est le plus petit des majorants (ppM) du sous-ensemble (x, y) et $x \wedge y$ est le plus petit des majorants (ppM) du sous-ensemble (y, x) or ces deux ensembles sont identiques comme ayant les mêmes éléments.

3. Associativité :

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

   Cette démonstration est analogue à la précédente.

4. Absorption :

   $x \vee (x \wedge y) = x$ et $x \wedge (x \vee y) = x$.

   La relation d'ordre étant réflexive, on aura $x \leq x$, de plus $x \wedge y$ étant le (pGm), on peut écrire $(x \wedge y) \leq x$; ceci prouve que x est un majorant de l'ensemble $(x, x \wedge y)$, par ailleurs si z est un majorant de $(x, x \wedge y)$ on a nécessairement $x \leq z$. Il résulte de tout ceci que x est le (ppM) de $(x, x \wedge y)$ c'est à dire $x \vee (x \wedge y) = x$.

   Le théorème précédent admet une importante réciproque.

Théorème 2 :[45] Si un ensemble T est muni de deux opérations notées $\perp$ et $*$ qui ont les propriétés suivantes :
$x\perp y = y\perp x$ et $x * y = y * x$
$x\perp(y\perp z) = (x\perp y)\perp z$ et $x * (y * z) = (x * y) * z$
$x\perp x = x$ et $x * x = x$
$x\perp(x * y) = x$ et $x * (x\perp y) = x$
Alors, il existe sur T une seule relation d'ordre qui fait de celui-ci un treillis dans lequel les opérations ($\perp$ et $*$) donnent respectivement le plus petit des majorants **(ppM)** et le plus grand des minorants (pGm).
Pour démontrer ce théorème, nous devons énoncer les lemmes suivants :

Lemme 1 : [**45**] On a l'équivalence suivante : $x\perp y = y \Leftrightarrow x * y = x$.

Démonstration : Si nous supposons que $x\perp y = y$, alors on peut en déduire que $x * y = x * (x\perp y) = x$.
La démonstration de la réciproque est analogue.

Lemme 2 : [**45**] Si on définit sur T une relation binaire notée $\leq$ par $x \leq y$ si et seulement si $x\perp y = y$, alors $\leq$ est une relation d'ordre, de plus celle ci fait de T un treillis et les opérations $\wedge$ et $\vee$ de ce treillis sont respectivement les opérations $*$ et $\perp$.

Démonstration : Pour montrer que $\leq$ est une relation d'ordre, il faut montrer qu'elle est réflexive, antisymétrique et transitive.

1. Réflexive : On a bien $(x \leq x)$, c'est à dire $(x \perp x) = x$
   à cause de l'idempotence.
2. Antisymétrique : $x \leq y \Rightarrow x \perp y = y$ et $y \leq x \Rightarrow y \perp x = x$.
   comme $x \perp y = y \perp x$, on a $x \leq y$ et $y \leq x$
   ceci veut dire que $x = y$.
3. Transitive : Montrons que $x \leq y$ et $y \leq z \Rightarrow x \leq z$
   $x \perp y = y$ et $y \perp z = z$
   $x \perp z = x \perp (y \perp z) = (x \perp y) \perp z$

La relation $\leq$ est bien une relation d'ordre, il reste à démontrer que tout couple (x, y) possède un ppM $x \perp y$ et un pGm $x * y$
$x \perp y$ est un majorant de x ;
en effet : $x \leq x \perp y \Leftrightarrow x \perp (x \perp y) = x \perp y$;
or, on a bien ; $x \perp (x \perp y) = (x \perp x) \perp y = x \perp y$
on montre de la même manière que $x \perp y$ est un majorant de y.
Pour montrer que $x \perp y$ est le ppM de (x, y), nous allons supposer qu'il existe un majorant z de (x, y) , on aura alors ;
$x \leq z$ et $y \leq z$ soit $x \perp z = z$ et $y \perp z = z$.
Ces deux relations impliquent $x \perp y \leq z$;
en effet : $(x \perp y) \perp z = x \perp (y \perp z) = x \perp z = z$.
On a donc bien $(x \perp y) \leq z$.
Idem, on montrerait que $x * y$ est le pGm de (x, y).

Lemme 3 : [**45**] La relation $\leq$ définie ci dessus est la seule relation d'ordre sur T, qui fasse de T un treillis telles que les opérations $\perp$ et $*$ définissent le ppM et le pGm.

Démonstration : Soit $\Re$ une relation d'ordre sur T différente de $\leq$, tel que T soit un treillis dans lequel $\perp$ et $*$ donnent respectivement le ppM et le pGm.
Si $x\Re y$ alors y est le ppM de (x, y) ;
comme $\perp$ donne par hypothèse le ppM, alors $x \perp y = y$.
Il résulte alors du lemme précédent que $x \leq y$.
On montrerait de la même manière que la relation $\leq$ est la seule pour laquelle $x * y$ est le pGm de $(x,\ y)$.
Si $x\Re y$ alors x est le pGm de (x, y) ;
comme $*$ donne par hypothèse le pGm, alors $x * y = x$.
Il résulte alors du lemme précédent que $x \leq y$.

2.3.2 Propriétés algèbriques de quelques classes de treillis

Nous allons étudier les classes de treillis distributifs, complémentés, modulaires et booléens. Ce choix n'est pas arbitraire, mais ; bien réflechi, car celles-ci sont les plus importantes, les plus intéressantes et les plus particulières dans le monde des treillis. On donnera aussi quelques propriétés et on les représentera à l'aide de schèmas illustratifs.

2.3.2.1.Treillis Modulaires :[15,30,32,33,34]

– Dans tout treillis T, on sait qu'on a, pour trois éléments quelconques x, y, z :

$$(1)\ x \vee (y \wedge z) \leq (x \vee y) \wedge (x \vee z)$$

On en déduit immédiatement que :

$$(2)\ x \leq z \text{ entraine } x \vee (y \wedge z) \leq (x \vee y) \wedge z,$$

l'égalité ayant lieu pour $x = z$ d'après la loi d'absorption.

– Un treillis T est dit **modulaire** si $\forall x, y, z \in T$,
on a :

$$(3)\ x \leq z \text{ alors } x \vee (y \wedge z) = (x \vee y) \wedge z.$$

– 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 (3).

Remarque 1 : La propriété de modularité se conserve par dualité, car la propriété duale

$$x \geq z \text{ entraine } x \wedge (y \vee z) = (x \wedge y) \vee z$$

c'est à dire

$$x \leq z \text{ entraine } z \vee (x \wedge y) = (z \vee y) \wedge x$$

n'est autre que (3) où x et z ont été échangés.

– Tout sous-treillis d'un treillis modulaire est modulaire.
– Pour qu'un treillis soit modulaire, il suffit de montrer que, si $x \leq z$, alors

$$x \vee (y \wedge z) \geq (x \vee y) \wedge z.$$

Diagramme d'un treillis modulaire

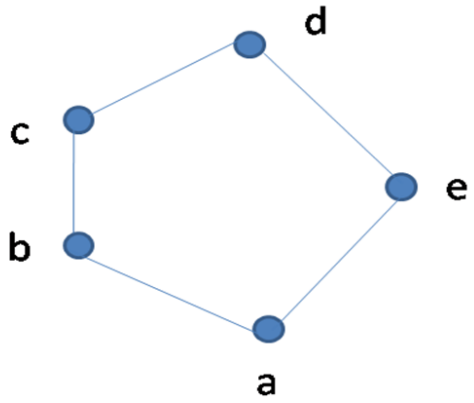

FIG. 2.4 – Treillis modulaire

Remarque 2 : Tout treillis distributif est modulaire.
Considérons maintenant, dans un treillis modulaire, trois éléments a, b, c liés par les relations :

$$a \leq b$$
$$c \wedge a = c \wedge b$$
$$c \vee a = c \vee b$$

Nous avons :
$a = a \vee (c \wedge a) = a \vee (c \wedge b) = (a \vee c) \wedge b = (b \vee c) \wedge b = b$
Ainsi, tout treillis modulaire vérifie la condition suivante :

$$(\mathbf{I}) \begin{cases} a \leq b \\ c \wedge a = c \wedge b \\ c \vee a = c \vee b \end{cases}$$
$$\text{entrainent } a = b$$

Ici, $a \leq b$ peut évidemment être remplacé par :
a et b sont comparables. Remarquons en outre que cette nouvelle condition (I), condition necessaire pour qu'un treillis soit modulaire, équivaut à l'inexistence de sous-treillis à cinq éléments du type N_5.
Montrons que la condition (I) est aussi suffisante, ce qui fournira une nouvelle définition d'un treillis modulaire et, c'est pour cela, que, dans un treillis T non modulaire, la condition (I) n'est pas verifiée. Par hypothèse, il existe dans T des éléments x, y et z tels que l'on ait, d'après (I) :

$$x \prec z \text{ et } x \vee (y \wedge z) \prec (x \vee y) \wedge z.$$

Cette dernière inégalité étant stricte ; considérons alors les éléments

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

nous avons $a \prec b$ et $\forall c \in T$,

$$a \vee c \leq b \vee c,\ a \wedge c \leq b \wedge c.$$

Prenons alors $c = y$; il vient :

$$a \vee c = x \vee (y \wedge z) \vee y)$$
$$b \vee c = [(x \vee y) \wedge z] \vee y \leq [(x \vee y) \wedge z] \vee (x \vee y) = x \vee y$$
donc :
$$b \vee c \leq a \vee c,$$
et par conséquent $a \vee c = b \vee c$

– calculons de même :

$$b \wedge c = (x \vee y) \wedge z \wedge y = y \vee z$$
$$a \wedge c = [x \vee (y \wedge z)] \wedge y \geq [x \vee (y \wedge z)] \wedge (y \wedge z) = y \wedge z$$

d'où

$$b \wedge c \leq a \wedge c$$

et par conséquent :

$$b \wedge c = a \wedge c$$

ainsi, la condition (I) n'est pas vérifiée, d'où le théorème suivant :

Théorème 3 :[15] Les treillis modulaires sont les treillis distributifs qui vérifiént la condition :

$$(\mathbf{II}) \begin{cases} a \text{ et } b \text{ comparables} \\ a \wedge c = b \wedge c \\ a \vee c = b \vee c \end{cases}$$

entrainent $a = b$

Théorème 4 :[15] Les treillis modulaires sont ceux qui n'admettent aucun sous-treillis à cinq éléments du type N_5.
Ce théorème permet d'établir une proposition analogue pour les treillis distributifs. On sait que d'après la proposition énoncée dans (2.3.2.1), dans tout treillis distributif, les égalités :

$$a \wedge c = b \wedge c$$

et

$$a \vee c = b \vee c \text{ entrainent } a = b.$$

Montrons réciproquement que tout treillis T dans lequel on a cette propriété est distributif. La propriété est valable, en particulier, si a et b sont comparables. Par conséquent, le treillis T est, au moins, modulaire. Soient x, y, z trois éléments de T ; posons :

$$a = (x \wedge y) \vee [z \wedge (x \vee y)],$$
$$b = (y \wedge z) \vee [x \wedge (y \vee z)].$$

Puisque T est modulaire, l'inégalité $x \wedge y \leq x \vee y$
permet d'ecrire :

$$a = [(x \wedge y) \vee z] \wedge (x \vee y),$$
$$b = [(y \wedge z) \vee x] \wedge (y \vee z).$$

ces expressions sont duales des précédentes.

Prenons $c = y$;

nous avons

$$a \wedge c = [(x \wedge y) \vee z] \wedge y$$

d'où, en utilisant l'inégalité $x \wedge y \leq y$ et la modularité.

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

l'expression de $b \wedge c$ s'obtient en permutant x et z :

$$b \wedge c = (z \wedge y) \vee (x \wedge y) = a \wedge c.$$

Par dualité, on a de même

$$b \vee c = a \vee c \text{ donc } a = b$$
$$\text{d'où } a \wedge x = b \wedge x$$
$$\text{or}$$
$$a \wedge x = \{[(x \wedge y) \vee z] \wedge (x \vee y)\} \wedge x = [(x \wedge y) \vee z] \wedge x$$

d'où, en utilisant encore l'inégalité $x \wedge y \leq x$ et la modularité,

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

De même ;

$$b \wedge x = \{(y \vee z) \wedge [x \vee (y \wedge z)]\} \wedge x = (y \vee z) \wedge x$$

Nous avons donc finalement ;

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

et le treillis est distributif, par conséquent, il est modulaire.

2.3.2.2. Treillis distributifs :[11,30,32,33,34]

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

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$
$$\text{et}$$
$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$$

– 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.

Diagramme d'un treillis distributif

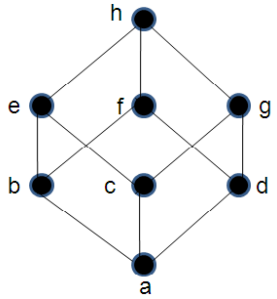

FIG. 2.5 – treillis distributif

Supposons T distributif alors :

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

Soit par application de la loi d'absorption

$$(x \vee y) \wedge (x \vee z) = x \vee [(x \wedge z) \vee (y \wedge z)]$$
$$(x \vee y) \wedge (x \vee z) = x \vee (x \wedge z) \vee (y \wedge z)$$
$$(x \vee y) \wedge (x \vee z) = x \vee (y \wedge z).$$

La réciproque s'établit de manière analogue en montrant que :

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

Proposition 1 : Dans un treillis distributif, on peut simplifier les propriétés algèbriques suivantes comme suit :

$$\forall x, y, z \in T,\ x \wedge z = y \wedge z\,;\ \text{et}$$
$$x \vee z = y \vee z\,;\ \text{alors}\ x = y\,;$$

Remarque 3 : Il faut bien distinguer la notion de sous-ordre de celle de sous-treillis. En effet, un treillis distributif peut contenir M_3 ou N_5 comme sous-ordre, mais pas comme sous-treillis.

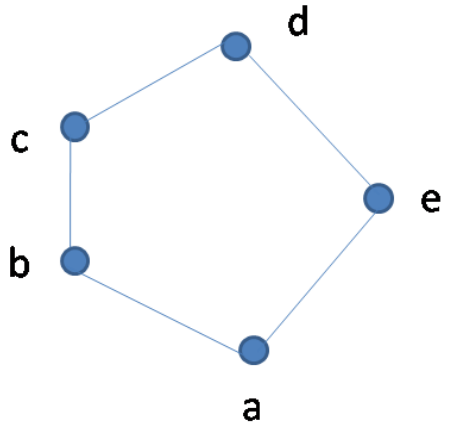

FIG. 2.6 – sous-treillis isomorphe à N_5

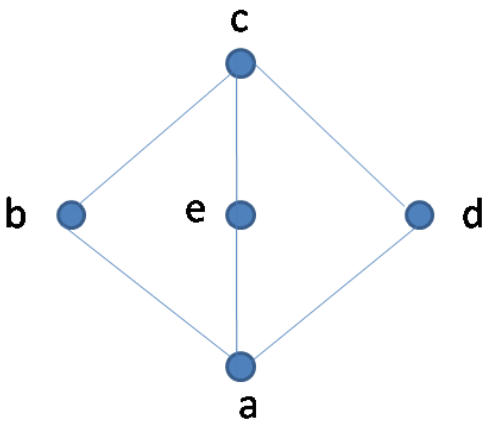

FIG. 2.7 – sous-treillis isomorphe à M_3

Exemples de treillis non distributifs :

Les figures 2.5 et 2.6 représentent des treillis qui ne sont pas distributifs en raison des égalités suivantes :

pour la figure 2.5 on a :

$a \vee (b \wedge c) = a \vee z = a$

$(a \vee b) \wedge (a \vee c) = b \wedge u = b$

or $a \neq b$

pour la figure 2.6 on a :

$a \vee (b \wedge c) = a \vee z = a$

$(a \vee b) \wedge (a \vee c) = u \wedge u = u$

or $a \neq u$

$a \wedge (b \vee c) = a \wedge u = a$

$(a \wedge b) \vee (a \wedge c) = z \wedge z = z$

or $a \neq z$

Remarque : 4 Avec cinq éléments distincts formant un ensemble ordonné, les seuls treillis distributifs possibles sont ;

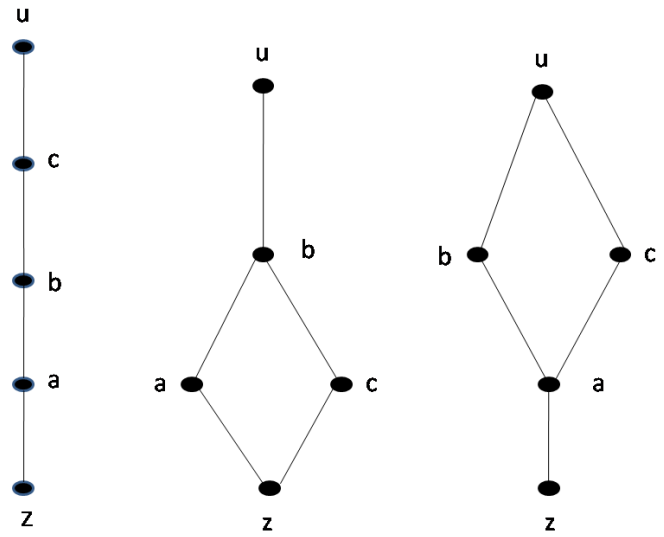

FIG. 2.8 – Treillis distributifs à 5 éléments

On vérifie sans peine que la chaine et ces deux treillis sont des treillis distributifs. Ce sont les seuls treillis distriburifs à cinq éléments.

Exemples de treillis distributifs :

– Il est facile de remarquer que $I(P)$ et $A(P)$ sont respectivement les ensembles des idéaux de P et des antichaînes maximales de P, ceux ci sont deux treillis distributifs appelés respectivement treillis des idéaux de P et treillis des antichaînes maximales de P.

– Le treillis T formé par l'ensemble de tous les sous-ensembles d'un ensemble X ordonné par la relation d'inclusion est un treillis distributif.
– l'ensemble des parties de E dans lequel les opérations sont $(\cap)$ et $(\cup)$ au sens ensembliste est un treillis distributif.

Propriété 1 : Soit T un treillis, il est distributif si et seulement si T est gradué et $|J(T)| = |M(T)| = |H(T)|$.
Sachant que :
$J(T)$ étant l'ensemble des éléments sup-irréductibles.
$M(T)$ l'ensemble des éléments inf-irréductibles.
et $H(T)$ la hauteur du treillis.

2.3.2.3.Treillis Complémenté :[30,32,33,34]

Définition du 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 :

$$x \vee y = M \text{ et } x \wedge y = m.$$

Diagramme d'un treillis complémenté :

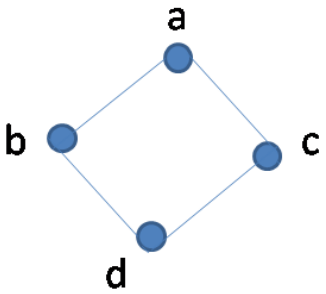

FIG. 2.9 – Treillis complémenté

Remarque 5 : Notons au passage qu'en raison de la commutativité, si y est le complément de x alors x est le complément de y.

– Un treillis dans lequel tout élément a au moins un complément est dit **Complémenté**

Remarque 6 : Dans un treillis distributif le complément d'un élément s'il existe alors il est unique.

2.3.2.4.Treillis Booléens :[30,32,33,34,45,49]

Un treillis ditributif et complémenté est un treillis Booléen.

Diagramme d'un treillis booléen

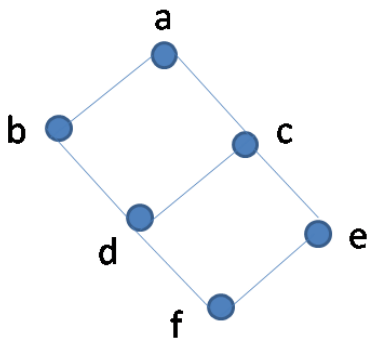

FIG. 2.10 – Treillis booléen

- Les treillis booléens forment la base algèbrique de la logique classique.
- Un treillis booléen est un treillis distributif qui est doté d'une opération de complémentation.

Exemples de treillis booléens :

- Un treillis distributif et complémenté est un treillis de boole
- Une algèbre de Boole est un treillis booléen
- Un treillis distributif borné et complémenté s'appelle aussi une algèbre de Boole.
- L'ensemble des fonctions booléennes muni des opérations $\vee$, $\wedge$ et de la complémentation forme une algèbre de boole et, de ce fait, un treillis booléen.

- L'ensemble T de tous les sous-ensembles d'un ensemble X ordonnés par la relation d'inclusion est un treillis de Boole. Il comporte un élément plus grand que tous les autres qui est l'ensemble X, et un élément plus petit que tous les autres qui est l'ensemble vide.
- Une algèbre de Boole $(0, 1)$ possédant deux opérations $\vee$ et $\wedge$ qui sont commutatives, associatives, idempotentes et vérifiant la loi d'absorption, forme un treillis.
- Ces opérations étant distributives et chaque élément possédant un complément, il en résulte que toute algèbre de Boole est un treillis de Boole.
  - La relation d'ordre sur l'ensemble qui sera notée par $\leq$ est définie par :

$$x \leq y \Leftrightarrow x \vee y = y.$$

Le résultat de l'opération $\vee$ est le suivant :

$$0 \vee 0 = 0$$
$$0 \vee 1 = 1$$
$$1 \vee 0 = 1$$
$$1 \vee 1 = 1$$

Le résultat de l'opération $\wedge$ est le suivant :

$$0 \wedge 0 = 0$$
$$0 \wedge 1 = 0$$
$$1 \wedge 0 = 0$$
$$1 \wedge 1 = 1$$

Chapitre 3

Aspects algorithmiques des treillis

Il existe deux familles d'algorithmes de construction ou de génération de treillis .Celle des incrémentaux, et celle des non incrémentaux. Les plus connus pour la première, sont ceux conçus par Norris(78), Godin(91), Oosthuisen(91), Carpineto(93), Dowling(93), Vatchev (00), construisent le treillis au fur et à mesure que les objets arrivent.
Ceux de la seconde famille sont conçus par Chen(69), Ganter(84), Bordat(86), Zabezhailo(87), Kuznetsov(93), Lindig(99), Nourine et Raynaud (99), Stumme(00), Nourine et al(02), Fu et Mephu(03), construisent le treillis une fois que tous les concepts sont connu.
Plusieurs travaux se sont attachés à comparer les algorithmes de construction de treillis entre eux, en termes de complexité algorithmique, espace mémoire et temps de calcul.

3.1 Algorithme de construction du graphe de couverture d'un treillis

Cet algorithme non incrémental a été conçu par Nourine et Raynaud pour la construction et le calcul du graphe de couverture d'un treillis ; il utilise une approche en deux étapes :
1. Génére la famille F représentée dans un arbre lexicographique.
2. Calcule les relations de couverture des éléments de F.

première étape : L'algorithme calcule l'arbre lexicographique (décrit dans [27,28]) représenté par la famille F générée par la base B.
Soit X un ensemble, P un ordre total sur X et B une base composée par l'ensemble des parties de X.
Désignons par F la famille générée par l'union des éléments de la base B, avec ;

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

on dit aussi que B est un générateur de F et celle-ci est dite fermée pour l'union.
Chaque élément de F est représenté par un couple $(f, \gamma(f))$ où ;

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

Il faut voir chaque constituant de F comme un mot de l'alphabet X ordonné dans l'ordre croissant ;
clairement, il y a une bijection entre ;

$$F = \{a_1, a_2, ..., a_k\} \rightarrow a_1, a_2, ..., a_k \,;$$

et

$$a_1 \prec a_2 \prec ... \prec a_k \,;$$

Montrons maintenant, comment construire cet arbre lexicographique à partir d'une base ;

$$B = \{b_1, b_2, ..., b_m\} \,;$$

– La racine (The root) correspond à F^0 ; l'ensemnble vide
– A l'étape i, c'est le calcul de la famille fermée par l'union F^i à partir de F^{i-1} en utilisant la formule suivante :

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

avec ;

$$B^i = \{b_1, b_2, ..., b_i\} \,;$$

Algorithme 1 Tree B

Data : La base B

Result :arbre lexicographique T_F de F

Begin $F = \{\emptyset\}$; (The root of T_F)

For each $b \in B$ do

For $f \in F$ do

1. $\rightarrow f' = f \cup b$
2. $\rightarrow$ **if** $f' \notin F$ **then** $F = F \cup \{f'\}$
3. $\rightarrow \gamma(f') = \gamma(f) \cup \{b\}$

end if

end for

end for

end

Théorème 5 : [37] L'algorithme 1 calcule l'arbre lexicographique de la famille F générée par la base B en

$$o((|X| + |B|) * |B| * |F|) \text{ étapes} ;$$

avec une complexité spatiale en

$$o((|X| + |B|) * |F|) ;$$

Preuve : Les lignes 1 et 3 sont faites en

$$o((|X| + |B|) ;$$

la ligne 2 vérifie si f' est dans l'arbre sinon l'inserer. Cette instruction est faite ou est implémentée en

$$o(|X|) ;$$

puisque, les enfants d'un noeud sont triés en fonction de P l'ordre total sur X.

Ainsi la complexité de la boucle interne For qui se répète ;

$$|B| \text{ fois ;}$$

est donc ;

$$o((|X| + |B|) * |F|) ;$$

Seconde étape : Cette étape consiste à calculer le graphe de couverture $G = (F, \subseteq)$ à partir de l'arbre lexicographique de la famille F générée par la base B ; et cela en utilisant le théorème de couverture suivant :

Théorème 6 : On définit ;

$$\Delta(f', f) = \gamma(f') \setminus \gamma(f) ;$$

et on note par ;

$$\prec \text{ la relation de couverture ;}$$

Soient f et $f' \in F$ tels que ;

$$f \subset f' ;$$

alors ;

$$f \prec f' \Leftrightarrow \forall (b_1, b_2) \in \Delta(f', f)\ b_1 \setminus f' = b_2 \setminus f ;$$

Preuve : Soient f et $f' \in F$ tels que ;

$$f \subset f' \text{ alors } f' ;$$

peut s'écrire ;

$$f' = f \cup \{b / b \in \Delta(f', f)\} ;$$

(1) Supposons que f est couvert par f' ; montrons que pour tous les b_1, $b_2 \in \Delta(f', f)$;

nous avons ;

$$b_1 \setminus f = b_2 \setminus f \,;$$

Supposons que

$$b_1 \setminus f \notin b_2 \setminus f \,;$$

ainsi donc ;

$$f" = f \cup b_1 \subset f' = f \cup \{b/b \in \Delta(f', f)\} \,;$$

par conséquent ;

$$f \subset f" \subset f' \,;$$

est en contradiction avec ;

$$f \text{ est couvert par } f' \,;$$

ainsi ;

$$b_1 \setminus f \subseteq b_2 \setminus f \,;$$

De même, montrons que ;

$$b_1 \setminus f \supseteq b_2 \setminus f \,;$$

(2) Inversement, on suppose que pour tout ;

$$b_1,\, b_2 \in \Delta(f', f) \,;$$

on a ;

$$b_1 \setminus f = b_2 \setminus f \,;$$

Soit $f" \in F$ tel que ;

$$f \subset f" \subset f' \,;$$

il est clair que ;

$$\gamma(f) \subset \gamma(f") \subset \gamma(f') \,;$$

et donc ;

$$f" = f \cup \{b/b \in \Delta(f", f)\} = f' \,;$$

$$\gamma(f") \setminus \gamma(f) \subseteq \gamma(f') \setminus \gamma(f) = \Delta(f', f)\,;$$

Le corollaire suivant est la conséquence du théorème 7.

Corollaire 1 : Soient f et $f' \in F$ tels que ;

$$f \subseteq f'\,;$$

alors ;

$$f \prec f' \Leftrightarrow \forall f' = f \cup b \text{ pour tout b} \in \Delta(f', f)\,;$$

Décrivons maintenant, comment calculer le graphe de couverture $G = (F, \subseteq)$. Considèrons la famille F générée par la base B utilisée dans l'algorithme 1.

La strategie de cet algorithme consiste à calculer l'ensemble des éléments de couverture noté par $Imsucc(f)$ pour chaque élément de la famille F.

En clair $f' \in F$ est candidat si $f \subset f'$ et f' peut être calculé par $f' = f \cup b$ pour certains $b \in B \setminus \gamma(f)$.

Posons ;

$$S(f) = \{f \cup b / b \in B \setminus \gamma(f)\}\,;$$

Cet ensemble de candidats $S(f)$ pour la couverture f, 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 $S(f)$. Pour se faire, nous calculons l'ensemble $S(f)$ en maintenant le nombre d'occurences de chaque élément f' dans le compteur $count(f')$, ensuite pour chaque élément $f' \in S$, on verifie si $|\Delta(f', f)|=$ $count(f')$ alors f' couvre f d'après le corollaire 1.

Etant donné l'arbre lexicographique T_F de la famille F générée par la base B, l'algorithme suivant construira le graphe de couverture de $G = (F, \subseteq)$.

Algorithme 2 Graphe de couverture de $G = (F, \subseteq)$
Data : arbre lexicographique de F et de $\gamma(F)$
Result : listes d'adjacence des Imsucc du graphe de couverture du treillis $(F, \subseteq)$
Begin
Initialiser $Count(f)$ à 0 pour tout $f \in F$
for $f \in F$ **do**
for $b \in B \setminus \gamma(f)$**do**

1. $\rightarrow f' = f \cup b$
   $Count(f') + +$
2. $\rightarrow$ **if** $|\gamma(f')| = Count(f') + |\gamma(f)|$ **then**
   $ImSucc(f) = ImSucc(f) \cup \{f'\}$

end if
end for
réinitialiser $Count$
end for
end

Algorithme 3 UCS(B)
Data : base B à m éléments
Result : liste d'adjacence des Imsucc du graphe de couverture du treillis $(F, \subseteq)$
Begin
$F = Tree(B)$
Graphe de couverture $G = (F, \subseteq)$
end

Théorème 7 : L'algorithme 2 calcule les listes d'adjacence des Imsucc du graphe de couverture pour le treillis $(F, \subseteq)$ en $O((|X| + |B|)\,|B|\,.\,|F|)$ étapes.

Preuve : En clair, l'algorithme 2 calcule le graphe de couverture du treillis $(F, \subseteq)$ par le corollaire 1.
Le calcul de $|\gamma(f)|$ et $|\gamma(f')|$ (instruction2) se fait en $O(|X|+|B|)$ (en temps) en utilisant la recherche dans l'arbre lexicographique ; d'où la complexité de la boucle interne (for) est de $O((|X| + |B|)\,|B|)$.
Reinitialiser le compteur *count* pour tous les éléments calculés par l'instruction1 se fait en $O(|B|)$ en les laissant dans une liste chainée.
Il est clair maintenant que la complexité de l'algorithme 2 est de $O((|X| + |B|) * |B| * |F|)$

3.1.1 Etude de la compléxité

Par décompte du nombre de boucles, cette étape a un coût de ;

$$O((|X| + |B|)\,|B|\,.\,|F|) \text{ en temps ;}$$

et

$$O((|X| + |B|)\,|F|) \text{ en espace ;}$$

Le premier algorithme est réalisé en ;

$$O((|X| + |B|)\,|B|\,.\,|F|).$$

La seconde partie de l'algorithme s'appuie sur une caractèrisation de $\prec$ dans notre cas :

Le second algorithme est réalisé aussi en ;

$$O((|X| + |B|)\,|B|\,.\,|F|).$$

Le coût total de l'algorithme est en ;

$$O((|X| + |B|)\,|B|\,.\,|F|).$$

3.1.2 Applications

Cet algorithme peut être utilisé et appliqué pour le calcul du graphe de couverture pour plusieurs types ou classes de treillis. Pour cela, on définit un ordre $P = (X, \leq)$ tels que :

$$A^u = \{x \in X / a \leq x, a \in A\}$$
$$A^l = \{x \in X / a \geq x, a \in A\}$$
$$\text{et} \downarrow x = \{y \in X / y \leq x\}$$

Application 1 : Treillis des Idéaux d'un ordre P : Soit $P = (X, \leq)$ un ensemble ordonné, posons $I \subset X$, I est un idéal de P, si ;

$$y \in I \text{ et } x \leq y \Longrightarrow x \in I \,;$$

On note par $I(P)$ l'ensemble des idéaux de l'ordre P.
Cet ordre lorsqu'il est ordonné par l'inclusion $(I(P), \subseteq)$ est un treillis distributif. $[I(P) = (I(P), \subseteq)]$ est appelé treillis des idéaux.

Corollaire 2 Soit $P = (X, \leq)$ un ordre et $B = \{\downarrow x, x \in X\}$ une base, alors $I(P)$ le treillis des idéaux est isomorphe à $(F, \subseteq)$.

Proposition 2 : [27] l'algorithme 3 calcule les listes d'adjacence du graphe de couverture pour le treillis des idéaux, $I(P) = (F, \subseteq)$ pour $P = (X, \leq)$ en temps $o(|X|^2 * |F|)$.

Preuve : Il est clair que $|B| = |X|$, l'algorithme dans a une complexité temporelle lineaire en la taille du treillis des idéaux $I(P)$; cela est dû aux régularités de l'arbre et le treillis des idéaux est un treillis distributif.

Application 2 : Completion de Dedekind-MacNeille : Soit $P = (X, \leq)$ un ensemble ordonné, une coupe de P est une paire (A, B) avec ;

$$A, B \subseteq X \text{ tels que } A^u = B \text{ et } B^l = A\,;$$

il est bien connu que les coupes ordonnées par ;

$$(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2 \text{ ou } B_1 \supseteq B_2\,;$$

forment un treillis complet, la completion de Dedekind de P notée $DM(P)$ [14] est le plus petit treillis complet contenant P comme un sous ordre.

Corollaire 3 : Soit $P = (X, \leq)$ un ordre et $B = \{X \setminus \downarrow x, x \in X\}$ une base ; alors $DM(P)$ est isomorphe à $(F, \supseteq)$.

Proposition 3 : [22] l'algorithme 3 calcule les listes d'adjacence du graphe de couverture pour la completion de DMN $(DM(P))$ tel que ;

$$DM(P) = (F, \supseteq) \text{ de } P = (X, \leq) \text{ en } o(|X|^2 * |F|).$$

Preuve : Il est clair que $|B| = |X|$, l'algorithme a une complexité temporelle de $o(|X|^3 * |F|)$.

Application 3 :Treillis des anti-chaines maximales : Soit $P = (X, \leq)$ un ordre ; on note par $AM(P)$ le treillis des antichaînes maximales de P.
Ce treillis noté ;

$$AM(P) = (AM(P), \leq)$$

est définit par :

$$A \leq B \Leftrightarrow A^l \subseteq B^l \text{ pour } A, B \in AM(P)\,;$$

Corollaire 4 : Soit $P = (X, <)$ un ordre strict ;

$$\text{et } B = \{\downarrow x \setminus \{x\} / x \in X\}\,;$$

une base alors ;

$$AM(P) \text{ est isomorphe à } (F, \supseteq)\,;$$

Proposition 4 : [29] L'algorithme 3 calcule les listes d'adjacence du graphe de couverture pour le treillis des antichaînes maximales, définit comme suit :

$$AM(P) = (F, \supseteq)$$
$$\text{de l'ordre } P = (X, \leq)$$
$$\text{en } o(|X|^2 * |F|).$$

Preuve : En clair ;

$$|B| = |X| ;$$

l'algorithme 3 a une complexité temporelle de ;

$$o(|X|^3 * |F|) ;$$

Application 4 : Treillis de galois : Soit $K = (X, Y, R)$ un contexte ou relation binaire ; un concept de K est une paire (A, B) avec $A \subseteq X$ et $B \subseteq Y$ tel que ;

$$A^u = B \text{ et } B^l = A ;$$

Il est bien connu que les concepts ordonnés par ;

$$(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2 ou B_1 \supseteq B_2 ;$$

forment un treillis complet, appelé treillis de galois de K, on le note par $Gal(K)$.

Corollaire 5 : Soit $K = (X, Y, R)$ un contexte ou une relation binaire ; et $B = \{X \setminus \downarrow y / y \in Y\}$ une base alors $Gal(K)$ est isomorphe à $(F, \subseteq)$.

Théorème 8 : L'algorithme 3 calcule les listes d'adjacence du graphe de couverture pour le treillis $Gal(K) = (F, \subseteq)$ en $O((|X| + |Y|)\,|Y|\,.\,|F|)$ comme complexité temporelle.

Conclusion : En conclusion, il est à constater que la méthode algorithmique de Nourine et Raynaud conduit à des applications ou algorithmes dont la complexité dans le pire des cas est plus faible que celle trouvée dans les algorithmes : (Calcul pratique du treillis de galois d'une correspondance de J.Bordat, Stepwise construction of Dedekind Mac-Neille de B.Ganter and al., Computing on line the lattice of maximal antichains of posets order de C.Jard).

Exemple 1 :

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 $F = \left\{\bigcup_{b\in I} / I \subseteq B\right\}$.

Par abus d'ecriture, on pose $\{1, ..., 5\} = \{12345\}$. On définit $B = \{x = \{245\}, y = \{1234\}, z = \{15\}\}$.

Appliquons l'algorithme 1 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** $b = \{245\}$ et $f = \emptyset$;
   $f' = f \cup b = \emptyset \cup \{245\} = \{245\} \notin F$;
   $F = F \cup \{f'\} = \{\emptyset\} \cup \{\{245\}\} = \{\emptyset, \{245\}\}$;
   $\gamma(f') = \gamma(f) \cup b = \{245\} = x$;
   $\gamma(F) = \{\emptyset, \{x\}\}$;
2. $F = \{\emptyset, \{245\}\}$, $\gamma(F) = \{\emptyset, \{x\}\}$;
   (a) **Pour** $b = \{1234\}$ et $f = \emptyset$;
   $f' = f \cup b = \{1234\} \notin F$;
   $F = \{\emptyset, \{245\}, \{1234\}\}$;
   $\gamma(f') = \gamma(\{1234\}) = y$;
   $\gamma(F) = \{\emptyset, \{x\}, \{y\}\}$;

(b) **Pour** $b = \{1234\}$ et $f = \{245\}$;

$f' = f \cup b = \{12345\} \notin F$;

$F = \{\emptyset, \{245\}, \{1234\}, \{12345\}\}$;

$\gamma(f') = \gamma(12345) = xyz$;

$\gamma(F) = \{\emptyset, \{x\}, \{y\}, \{xyz\}\}$;

3. $F = \{\emptyset, \{245\}, \{1234\}, \{12345\}\}$; $\gamma(F) = \{\emptyset, \{x\}, \{y\}, \{xyz\}\}$;

(a) **Pour** $b = \{15\}$ et $f = \emptyset$;

$f' = f \cup b = \{15\} \notin F$;

$F = \{\emptyset, \{245\}, \{1234\}, \{12345\}, \{15\}\}$;

$\gamma(f') = \gamma(\{15\}) = z$;

$\gamma(F) = \{\emptyset, \{x\}, \{y\}, \{xyz\}, \{z\}\}$;

(b) **Pour** $b = \{15\}$ et $f = \{245\}$;

$f' = f \cup b = \{1245\} \notin F$;

$F = \{\emptyset, \{245\}, \{1234\}, \{12345\}, \{15\}, \{1245\}\}$;

$\gamma(f') = \gamma(1245) = xz$;

$\gamma(F) = \{\emptyset, \{x\}, \{y\}, \{xyz\}, \{z\}, \{xz\}\}$;

(c) **Pour** $b = \{15\}$ et $f = \{1234\}$;

$f' = f \cup b = \{12345\} \in F$;

(d) **Pour** $b = \{15\}$ et $f = \{12345\}$;

$f' = f \cup b = \{12345\} \in F$;

Finalement, à partir de la base ;

$$B = \{x = \{245\}, y = \{1234\}, z = \{15\}\} ;$$

on obtient ceci :

$$F = \{\emptyset, \{245\}, \{1234\}, \{12345\}, \{15\}, \{1245\}\};$$
$$\gamma(F) = \{\emptyset, \{x\}, \{y\}, \{xyz\}, \{z\}, \{xz\}\};$$

Déssinons l'arbre lexicographique de F.

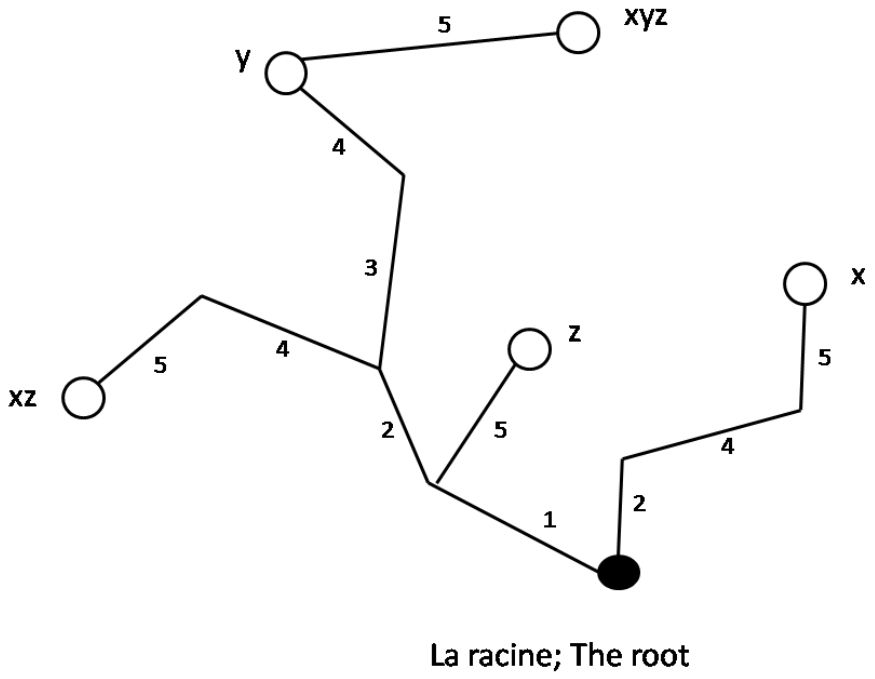

FIG. 3.1 – Arbre lexicographique de la famille F

Appliquons l'algorithme 2 qui consiste à calculer le graphe de couverture à partir de l'arbre lexicographique des familles ;

$$F = \{\emptyset, \{245\}, \{1234\}, \{12345\}, \{15\}, \{1245\}\};$$

et

$$\gamma(F) = \{\emptyset, \{x\}, \{y\}, \{xyz\}, \{z\}, \{xz\}\};$$

générées par la base ;

$$B = \{x = \{245\}, y = \{1234\}, z = \{15\}\};$$

Cet algorithme 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. $f = \emptyset$; $S(f) = S(\emptyset) = \{\emptyset \cup \{245\}, \emptyset \cup \{1234\}, \emptyset \cup \{15\}\}$
   $= \{\{245\}, \{1234\}, \{15\}\}$;
   calcul de $count(f')$ pour $f' \in S(f)$
   (a) Pour $f' = \{245\}, count(f') = count(\{245\}) = 1$;
   $\Delta(f', f) = \Gamma(f') \setminus \Gamma(f) = \{x\} \setminus \emptyset = \{x\}$;
   $|\Delta(f', f)| = 1 \Rightarrow |\Delta(f', f)| = count(f') = 1$;
   $(f, f') = (\emptyset, \{245\})$ est une couverture ;
   $ImSucc(f) = ImSucc(\emptyset) = \{\{245\}\}$;

(b) Pour $f' = \{1234\}\,; count(f') = count(\{1234\}) = 1;$
$\Delta(f', f) = \Gamma(f') \setminus \Gamma(f) = \{y\} \setminus \emptyset = \{y\}\,;$
$|\Delta(f', f)| = 1 \Rightarrow |\Delta(f', f)| = count(f') = 1;$
$(f, f') = (\emptyset, \{1234\})$ est une couverture ;
$ImSucc(f) = ImSucc(\emptyset) = \{\{245\}, \{1234\}\}\,;$

(c) **Pour** $f' = \{15\}\,, count(f') = count(\{15\}) = 1;$
$\Delta(f', f) = \Gamma(f') \setminus \Gamma(f) = \{z\} \setminus \emptyset = \{z\}\,;$
$|\Delta(f', f)| = 1 \Rightarrow |\Delta(f', f)| = count(f') = 1;$
$(f, f') = (\emptyset, \{15\})$ est une couverture ;
$ImSucc(f) = ImSucc(\emptyset) = \{\{245\}, \{1234\}, \{15\}\}\,;$

$count(f') = 0\,;$

2. $f = \{245\}, S(f) = S(\{245\}) = \{\{245\} \cup \{1234\}, \{245\} \cup \{15\}\}$
$= \{\{12345\}, \{1245\}\}\,;$

(a) Pour $f' = \{12345\}\,, count(f') = count(\{12345\}) =$
$1;$
$\Delta(f', f) = \Gamma(f') \setminus \Gamma(f) = \{xyz\} \setminus \{x\} = \{yz\}\,;$
$|\Delta(f', f)| = 2 \Rightarrow |\Delta(f', f)| \neq count(f');$
$(f, f') = (\{245\}, \{12345\})$ n'est pas une couverture ;

(b) Pour $f' = \{1245\}\,, count(f') = count(\{1245\}) = 1;$
$\Delta(f', f) = \Gamma(f') \setminus \Gamma(f) = \{xz\} \setminus \{x\} = \{z\}\,;$
$|\Delta(f', f)| = 1 \Rightarrow |\Delta(f', f)| = count(f') = 1;$
$(f, f') = (\{245\}, \{1245\})$ est une couverture ;
$ImSucc(f) = ImSucc(\{245\}) = \{\{1245\}\}\,;$

$count(f') = 0\,;$

3. $f = \{1234\}, S(f) = S(\{1234\}) = \{\{1234\} \cup \{245\}, \{1234\} \cup \{15\}\}$
   $= \{\{12345\}, \{12345\}\}$;

   (a) Pour $f' = \{12345\}, count(f') = count(\{12345\}) =$
   2;
   $\Delta(f', f) = \Gamma(f') \setminus \Gamma(f) = \{xyz\} \setminus \{y\} = \{xz\}$;
   $|\Delta(f', f)| = 1 \Rightarrow |\Delta(f', f)| = count(f') = 2$;
   $(f, f') = (\{1234\}, \{12345\})$ est une couverture ;
   $ImSucc(f) = ImSucc(\{1234\}) = \{\{12345\}\}$;

   $count(f') = 0$;

4. $f = \{12345\}$, $S(f) = S(\{12345\}) = \emptyset$;
   $count(f') = 0$;

5. $f = \{15\}, S(f) = S(\{15\}) = \{\{15\} \cup \{245\}, \{15\} \cup \{1234\}\}$
   $= \{\{1245\}, \{12345\}\}$;

   (a) **Pour** $f' = \{1245\}, count(f') = count(\{1245\}) =$
   1;
   $\Delta(f', f) = \Gamma(f') \setminus \Gamma(f) = \{xz\} \setminus \{z\} = \{x\}$;
   $|\Delta(f', f)| = 1 \Rightarrow |\Delta(f', f)| = count(f') = 1$;
   $(f, f') = (\{15\}, \{1245\})$ est une couverture ;
   $ImSucc(f) = ImSucc(\{15\}) = \{\{1245\}\}$;

   (b) **Pour** $f' = \{12345\}, count(f') = count(\{12345\}) =$
   1;
   $\Delta(f', f) = \Gamma(f') \setminus \Gamma(f) = \{xyz\} \setminus \{z\} = \{xy\}$;
   $|\Delta(f', f)| = 2 \Rightarrow |\Delta(f', f)| \neq count(f')$;
   $(f, f') = (\{15\}, \{12345\})$ n'est pas une couverture ;

$count(f') = 0$;

6. $f = \{1245\}$, $S(f) = S(\{1245\}) = \{\{1245\} \cup \{1234\}\}$
   $= \{\{12345\}\}$;
   (a) **Pour** $f' = \{12345\}$, $count(f') = count(\{12345\}) =$
   1;
   $\Delta(f', f) = \Gamma(f') \setminus \Gamma(f) = \{xyz\} \setminus \{xz\} = \{y\}$;
   $|\Delta(f', f)| = 1 \Rightarrow |\Delta(f', f)| = count(f') = 1$;
   $(f, f') = (\{1245\}, \{12345\})$ est une couverture ;
   $ImSucc(f) = ImSucc(\{1245\}) = \{\{12345\}\}$;

On va tout simplement récupèrer toutes les informations necessaires pour pouvoir déssiner le graphe de couverture $G = (F, \leq)$ du treillis.
Les couvertures (f, f') ou successeurs immédiats sont les suivants :

1. $(\emptyset, \{245\})$ est une couverture ;
   $ImSucc(\emptyset) = \{\{245\}\}$;
2. $(\emptyset, \{1234\})$ est une couverture ;
   $ImSucc(\emptyset) = \{\{245\}, \{1234\}\}$;
3. $(\emptyset, \{15\})$ est une couverture ;
   $ImSucc(\emptyset) = \{\{245\}, \{1234\}, \{15\}\}$;
4. $(\{245\}, \{1245\})$ est une couverture ;
   $ImSucc(\{245\}) = \{\{1245\}\}$;

5. $(\{1234\}, \{12345\})$ est une couverture ;
   $ImSucc(\{1234\}) = \{\{12345\}\}$;
6. $(\{15\}, \{1245\})$ est une couverture ;
   $ImSucc(\{15\}) = \{\{1245\}\}$;
7. $(\{1245\}, \{12345\})$ est une couverture ;
   $ImSucc(\{1245\}) = \{\{12345\}\}$;

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

$$F = \{\emptyset, \{245\}, \{1234\}, \{12345\}, \{15\}, \{1245\}\};$$

et

$$\gamma(F) = \{\emptyset, \{x\}, \{y\}, \{xyz\}, \{z\}, \{xz\}\};$$

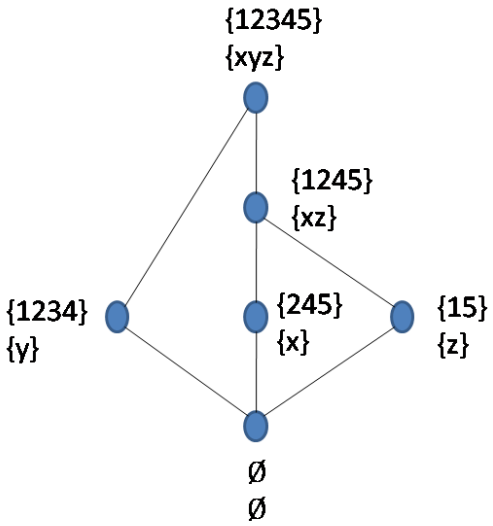

FIG. 3.2 – Graphe de couverture $G = (F, \subseteq)$

3.2 Algorithme de construction de treillis de concepts

Un nouvel algorithme, Genall élaboré par S.Ben Tekaya et al. [4,5] est proposé pour améliorer celui de L.Nourine et al en temps et espace mémoire. Cet algorithme construit le treillis dans lequel chaque concept formel est décoré par l'ensemble des générateurs minimaux qui lui est associé. Ce choix de l'algorithme de Nourine et al., peut être justifié par le fait qu'il présente la meilleure complexité théorique, alors qu'en pratique, ses performances restent insuffisantes comparé aux autres algorithmes tel que celui de Ganter [21].

Les données d'entrée de cet algorithme sont représentées sous forme de contexte formel.

Celui-ci définit un ensemble d'objets X qui possède des attributs.

Ce contexte est représenté sous forme d'un tableau dont les objets sont en lignes et les attributs dont l'ensemble est Y en colonnes.

3.2.1 Définitions

Contexte formel ou d'extraction : Un contexte formel ou d'extraction est un triplet $K = (X, Y, R)$ où : X et Y sont respectivement l'ensemble des objets et celui des attributs et $R \subseteq X * Y$ est une relation binaire exprimant que $\forall(x, y) \in R$, y est un attribut de l'objet x.

Contexte binaire : Un contexte $K = (X, Y, R)$ est dit binaire si les éléments de Y ont uniquement deux valeurs (0 ou 1) qui indiquent respectivement l'absence, ou la présence de l'attribut concerné dans la description d'un projet.

Correspondance de Galois : L'intérêt d'un treillis de concepts est d'organiser l'information concernant des groupements d'objets possédant des propriétés communes. On définit g et h deux fonctions qui permettent d'établir le lien (la correspondance) entre les objets et les attributs d'un contexte K comme suit :
$g : P(X) \longrightarrow P(Y)$
$g(O) = \{y \in Y / \forall x \in O, xRy\}$.
$h : P(Y) \longrightarrow P(X)$
$h(A) = \{x \in X / \forall y \in A, xRy\}$.
Le couple (g, h) est appelé correspondance ou connexion de Galois entre $P(X)$ et $P(Y)$.

Fermeture des ensembles : Les deux fonctions g et h vont servir à calculer la fermeture de O et A qui représentent respectivement un sous-ensemble d'objets et un sous-ensemble d'attributs. Pour cela , on compose les fonctions g et h comme suit :

$$O" = h(g(O))$$
$$A" = g(h(A))$$

On dit qu'un ensemble est fermé s'il est égal à sa fermeture. Ainsi O est fermé si :

$$O = O"$$
et
$$A = A"$$

Concept formel : Un concept formel est une paire $C_k = (O, A)$ avec $O \subseteq X$ et $A \subseteq Y$ tel que :

$$O = \{x \in X / \forall a \in A, (x, a) \in R\}$$
où $O = h(A)$

est l'extension ou objets couverts, elle est notée $\text{Ext}(C_k)$.

$$A = \{y \in Y / \forall o \in O, (o, y) \in R\}$$
où $A = g(O)$

est l'intension (ou attributs partagés), elle est notée $\text{Int}(C_k)$.

Ensemble de concepts L :

$$L = \{(o, a) \in P(X) \times P(Y) / O = h(A), A = g(O)\}.$$

Ordre sur les concepts : soient $C_1 = (O_1, A_1)$ et $C_2 = (O_2, A_2)$ deux concepts, La relation d'ordre dénotée $\leq$ est une relation définie entre les concepts de la manière suivante :
$(O_1, A_1) \leq (O_2, A_2) \Longleftrightarrow O_1 \subseteq O_2 \Longleftrightarrow A_1 \supseteq A_2$.
$C_1 \leq C_2 \Longleftrightarrow (O_1, A_1)$ est un sous-concept de (O_2, A_2).
(O_2, A_2) est un sur-concept de (O_1, A_1).

Ordre lexicographique : L'ordre lexicographique est l'ordre naturel utilisé pour générer certains objets combinatoires. Soit X un alphabet totalement ordonné. On étend l'ordre total sur X en un ordre lexicographique sur les ensembles formés par X, comme suit :
Soient A, B deux sous-ensembles distincts d'un ensemble X totalement ordonné. On dit que A est inférieur lexicographiquement à B si le plus petit élément qui distingue A et B appartient à A. on le note $A \prec_{lex} B$

Arbre lexicographique : Nous nous proposons de représenter une liste de mots ordonnée. Si l'ordre sur les mots est obtenu comme un ordre lexicographique à partir des composants des mots, nous pouvons représenter un ensemble de mots en utilisant la structure d'arbre de recherche lexicographique.

Soit le dictionnaire constitué des mots suivants :
AI, AIL, AILE, AINE, ALE, BAT, BAS ; il peut être représenté par l'arbre suivant :

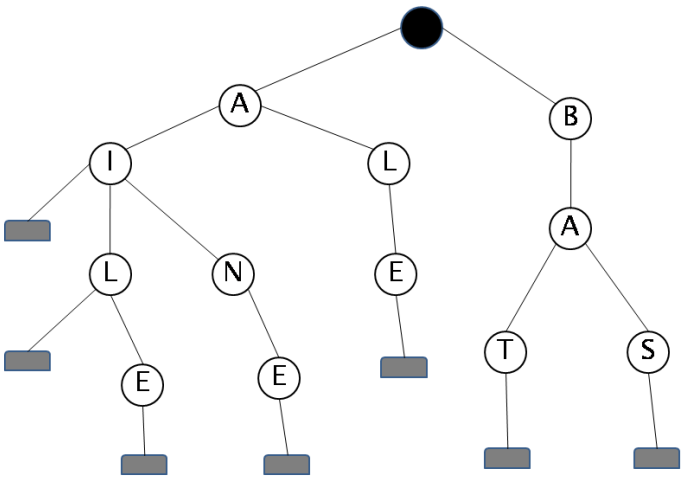

FIG. 3.3 – Arbre lexicographique associé au dictionnaire

Chaque mot du dictionnaire est associé à un chemin de la racine (rond noir) à un noeud (rectangle gris) dans l'arbre précédent.

Treillis de Galois : L'ensemble des concepts L muni de la relation d'ordre $\leq$ sur L est appelé treillis de Galois ou de concept et on le note $T = (L, \leq)$.

Face : Soit C_k=(O, A) un concept formel et soit $pred_i(C_k)$ le ième prédécesseur immédiat de C_k dans un treillis de Galois extrait d'un contexte K.
La ième face du concept formel C_k correspond à la différence entre son intension et l'intension de son ième prédécesseur.
Soit p le nombre de prédécesseurs immédiats du concept formel C_k et F la famille des faces.
La famille des faces F_{C_k} du concept formel C_k est exprimée par la relation suivante :
$F_{C_k} = \{A - Intent(pred_i(C_k))\}$, $i \in \{1, ..., p\}$.

Bloqueur : Soit $G = \{G_1, ..., G_n\}$ une famille de n ensembles ; un bloqueur B de la famille G est un ensemble où son intersection avec tout ensemble $G_i \in G$ est non vide.

Bloqueur minimal : Un bloqueur B est dit minimal s'il n'existe pas $B_1 \subset B$ et $\forall G_i \in G$, $B_1 \cap G_i \neq \emptyset$.

Générateur minimal : Soit C_k un concept formel et F_{C_k} sa famille de faces. L'ensemble G des générateurs minimaux associés à l'intension A du concept formel C_k, correspond aux bloqueurs minimaux associés à la famille de ses faces F_{C_k}.

Exemple 2 : Il s'agit d'une application permettant de calculer une correspondance de galois, un concept et les opérateurs de fermeture à partir d'une matrice binaire générée par un contexte $K = (X, Y, R)$.

R	a	b	c	d	e	F	g	h
1	1	1	1	1	1	1	1	
2	1	1	1	1	1	1		1
3	1	1	1	1	1		1	1
4	1	1	1	1		1		
5	1	1		1	1		1	
6	1	1	1		1			1
7	1		1			1		

FIG. 3.4 – matrice decrivant la relation R du contexte $K = (X, Y, R)$

$X = \{1, 2, 3, 4, 5, 6, 7\}$ et $Y = \{a, b, c, d, e, f, g, h\}$;
$g(\{1, 2, 3, 4\}) = \{a, b, c, d\}$ et
$h(\{a, b, c, d\}) = \{1, 2, 3, 4\}$ On voit que $(\{1, 2, 3, 4\}, \{a, b, c, d\})$ est un concept ; mais $(\{1, 2\}, \{e, f\})$ n'est pas un concept ; du moment que :
$g(\{1, 2\}) = \{a, b, c, d, e, f\}$ et $h(\{e, f\}) = \{1, 2\}$
si $X = \{2, 4, 5\}$, alors $g(X) = \{a, b, d\}$
$X" = \{1, 2, 3, 4, 5\}$
si $X = \{1, 5\}$, alors $g(X) = \{a, b, d, e, g\}$
$X" = \{1, 3, 5\}$

si $Y = \{a\}$, alors $h(Y) = \{1, 2, 3, 4, 5, 6, 7\}$

$Y" = \{a\}$

si $Y = \{a, c\}$, alors $h(Y) = \{1, 2, 3, 4, 6, 7\}$

$Y" = \{a, c\}$.

Seuls les ensembles $\{a\}$ et $\{a, c\}$ sont fermés.

3.2.2 Algorithme Genall

Ainsi, une étude soigneuse de l'algorithme de Nourine et al. a prouvé qu'il passe par deux étapes :
La première étape est une procédure de tri pour le stockage des concepts formels.
La seconde étape est une procédure de calcul de l'ordre sous-jacent entre les différents concepts formels.
Par contre, Genall recueille en un seul passage de la base toutes les informations exigées qui sont :
1. L'ensemble des concepts formels et leurs générateurs minimaux associés.
2. L'ordre partiel sous-jacent.

1 : **Algorithme Genall [4,5] pour la construction du treillis des concepts décoré par les générateurs minimaux**.
Entrée : Le contexte d'extraction $K = (X, Y, R)$
Sortie :Arbre lexicographique de F,$ImmSucc$,$liste - gen$
Debut
2 : $F = Y$
{ Initialiser la famille des concepts à l'ensemble des attributs }
3 : **Pour** chaque tuple $t \in K$ **Faire**
4 : $L = \emptyset$ { Initialiser la liste de l'ensemble des concepts trouvés dans une itération }

5 : **Pour chaque** concept $C_k \in F$ **Faire**

6 : $C.intent = C_k.intent \cap t.items$

7 : **Si** $C.intent \notin F$ **Alors**

8 : { Nouveau concept }

9 : $F = F \cup C$

10 : $C.extent = C_k.extent \cup t.TID$

11 : $C.ImmSucc = \{t.items\} \cup \{C_k\} \setminus \{C\}$

12 : $L = L \cup C$

13 : **Sinon**

14 : { Concept existant }

15 : $C.extent = C.extent \cup C_k.extent \cup t.TID$

16 : **Si** $C \notin L$ **Alors**

17 : $L = L \cup C$

18 : **Fin Si**

19 : $LP = \{t.items\} \cup \{C_k\} \setminus \{C\}$

{ Mise à jour de $C.ImmSucc$ }

20 : **Pour** chaque $P_k \in LP$ **Faire**

21 : **Pour** chaque $Succ \in C.ImmSucc$ **Faire**

22 : $Compare - Concept(Succ, P_k)$

23 : **Fin Pour**

24 : **Fin Pour**

25 : **Fin Si**

26 : **Fin Pour**

{ Finaliser l'ensemble des successeurs immédiats et calcul des générateurs minimaux }

27 : **Pour** Chaque concept $C_k \in L$ **Faire**

28 : $C_k.ImmSucc = Find - Succ(C_k, L)$

29 : **Pour** Chaque $Succ \in C_k.ImmSucc$ **Faire**

30 : $face = Succ \setminus C_k$

{ Calcul de la face du successeur immédiat}

31 : **Pour** Chaque $face_k \in Succ.liste - face$ **Faire**

32 : $Succ.liste - face = Compare - Face(face, face_k)$

33 : **Fin Pour**

34 : **Fin Pour**

35 : **Fin Pour**

36 : **Fin Pour**

Fin

Structure	Champs	Description
F		La famille des concepts formels C_k
L		La liste des concepts calculés dans une itération **k**
C_k	intent extent ImmSucc liste-face liste-gen	Le concept formel L'intension du concept C_k L'extension du concept C_k Liste des successeurs immédiats du concept C_k (les concepts qui le couvrent) Liste des faces du concept (la différence avec ses fils) Liste des générateurs minimaux du concept
T	TID items	La transaction de la base L'identificateur de la transaction Les items qui existent dans la transaction

FIG. 3.5 – Notations utilisées dans l'algorithme Genall

3.2.3 Etude de la complexité

Nous allons étudier la complexité, dans le pire des cas, de l'algorithme Genall [4,5].
Soient, σ l'ensemble des transactions d'une base, Y l'ensemble des attributs de la base et F l'ensemble des concepts trouvés. L'étude de la boucle Pour (ligne 5) donne :
L'instruction 6 s'effectue en $o(|Y|)$ étant donné que dans le pire des cas, nous allons effectuer une intersection avec tous les attributs de la base.
L'instruction 7 se fait en $o(|Y|)$ puisqu'on peut avoir au maximum $|Y|$ branches.
Par ailleurs, la complexité du bloc d'instructions $9-12$ (partie alors) étant inférieure à celle du bloc d'instructions $15-25$, nous ne comptabiliserons que cette dernière :
L'instruction 15 peut se faire dans le pire des cas en $o(|O|)$.
La recherche dans la liste L (ligne 16) est réalisée en $o(|F|/2)$ dans le pire des cas.
En effet, une itération peut donner autant de nouveaux concepts qu'il y a dans $|F|$.
La boucle Pour (ligne 20) peut se répéter au maximum deux fois.
Celle de la ligne 21 se répète $|ImmSucc|$ fois.
La fonction $Compare-Concept$ se fait en $o(|Y|)$.

De ce fait, la complexité des instructions $5-25$ est en $o(|F|+2\,|Y|+|\sigma|+|F|\,/2+2\,|Y|\,.\,|ImmSucc|)$.
L'instruction 29 s'effectue en $o(|jImmSucc|)$, alors que la complexité de l'instruction 32 est en $o(|Y|+|liste-gen|)$.
Par conséquent, la complexité du bloc comprenant les instructions 27 à 36 est en
$o(|F|\,/2+|ImmSucc|\,/2.(|Y|+|liste-gen|))$.
Puisque cet algorithme se répète $|K|$ fois (boucle Pour de la ligne 3). la complexité totale de l'algorithme est alors en :
$o(|K|\,.\,|F|\,.[1/2.\,|ImmSucc|^2\,.(|Y|+|liste-gen|)+|K|+|F|\,/2])$.
Bien que, l'algorithme Genall donne plus de résultats (calcule l'ensemble des concepts, leur ordre et l'ensemble des générateurs minimaux associés à chaque concept) que l'algorithme de Nourine et Raynaud, sa complexité reste toujours linéaire en fonction de $|F|$.

3.2.4 Comparaison avec d'autres travaux

L'algorithme de L.Nourine et Raynaud est semi-incrémental. En effet, il commence par construire d'une façon incrémentale l'ensemble des concepts, ce qui donne l'arbre des concepts. Ensuite, il utilise cet arbre pour la construction du diagramme. Or dans cette étape,[4,5] un retour systèmatique à la base est nécessaire pour le calcul de l'ordre. Chacune des étapes se fait en $o((|Y|+|O|).|O|.|F|)$. Par contre dans GENALL, le calcul des concepts et l'ordre sous-jacent se fait en une seule étape, avec en plus le calcul de l'ensemble des générateurs minimaux associés à chaque concept.

Ce calcul suppose que l'ordre est déjà établi. Cette hypothèse est due au fait que le calcul des concepts et de leur treillis constitue une étape lourde et coûteuse.

Dans notre approche, les générateurs minimaux sont déterminés au fur et à mesure de la construction du treillis. Malgré ce surcoût de calcul, la complexité de cet algorithme n'a pas augmenté.

3.2.5 Résultats expérimentaux

Les algorithmes Genall et celui de Nourine et al., ont été implémenté en language C sur deux machines identiques de type Pentium IV pour évaluer leurs performances.
Pour examiner l'efficacité pratique de l'algorithme GENALL, une sérié d'expérimentations a été menée sur des bases de données réelles et synthétiques.
Les résultats expérimentaux ont prouvé que l'algorithme Genall est plus efficace pour les contextes d'éxtraction denses comparé à l'algorithme de Nourine et al.
Effectivement le temps de réponse de l'algorithme Genall surpasse largement celui de Nourine et al (il est en moyenne 40 à 83 fois plus rapide suivant le nombre d'objets et d'attributs).

Exemple 3 :

R	a	b	c	d	e	f	g	h	i
1	x	x					x		
2	x	x					x	x	
3	x	x	x				x	x	
4	x		x				x	x	x
5	x	x		x		x			
6	x	x	x	x		x			
7	x		x	x	x				
8	x		x	x		x			

FIG. 3.6 – Le contexte d'extraction K

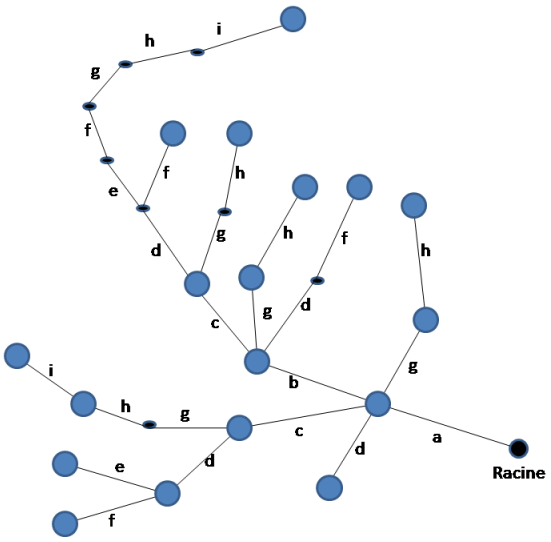

FIG. 3.7 – La structure de Trie associée au contexte d'extraction donné par la fig 3.6

Considérons le contexte d'extraction illustré par la figure 3.6 avec l'ensemble d'attributs $I = \{a, b, c, e, f, g, h, i\}$ et l'ensemble de transactions du contexte d'extraction, dénotées de 1 à 8.

Considérons la transaction $4 = \{acghi\}$, la famille **F** après le traitement des trois premières itérations est comme suit :

$\mathbf{F} = \{(abcdefghi, \emptyset), (abg, 123), (abgh, 23), (abcgh, 3)\}$.

L'ensemble des successeurs immédiats de chaque concept est comme suit :

$\{abg\}.Immsucc = \{\{abgh\}\}$

$\{abgh\}.Immsucc = \{\{abcgh\}\}$

$\{abcgh\}.Immsucc = \{\{abcdefghi\}\}$

Première étape : Génération des concepts formels

Chaque concept formel de **F** est manipulé individuellement, par conséquent, pour le premier concept formel C_1, $C_1.intent = \{abcdefghi\}$, l'application de l'opération d'intersection avec la quatrième transaction donne un nouveau concept formel avec une intension égale à $\{acghi\}$.

L'extension de ce nouveau concept formel, dénoté C_5 sera calculée plus tard.

Ainsi la famille **F** est mise à jour en lui ajoutant C_5 :

$\mathbf{F}$=$\{(abcdefghi, \emptyset), (abg, 123), (abgh, 23), (abcgh, 3),$
$(acghi, \emptyset)\}$.

L'ensemble des successeurs immédiats de C_5 est initialisé avec C_1 et l'extention de C_5 est initialisée avec l'union de $C_1.extent$ et l'identificateur de la transaction en question.
Par conséquent, $C_5.extent = \{4\}$ et $C_5.Immsucc = \{\{abcdefghi\}\}$ et la liste **L** est initialisée avec $\{acghi\}$.
Le processus mentionné ci-dessus est répété pour tous les concepts formels existants dans la famille **F**.
A la fin de cette première étape, nous obtenons la famille des concepts formels mise à jour.
$\mathbf{F} = \{(abcdefghi, \emptyset), (abg, 123), (abgh, 23), (abcgh, 3),$
$(acghi, 4), (ag, 1234), (agh, 234), (acgh, 34)\}$.
La liste des successeurs immédiats de chaque concept formel est comme suit :
$\{acghi\}.Immsucc = \{\{abcdefghi\}\}$;
$\{ag\}.Immsucc = \{\{abg\}, \{acghi\}\}$;
$\{agh\}.Immsucc = \{\{abgh\}, \{acghi\}\}$;
$\{acgh\}.Immsucc = \{\{abcgh\}, \{acghi\}\}$;
La liste L contient toutes les intensions des concepts formels découverts dans l'itération courante :
L=$\{\{ag\}$,$\{agh\}$,$\{acgh\}$ $\{acghi\}\}$.
Seconde étape : Raffinement de la liste des successeurs immédiats et determination des générateurs minimaux.
Pour le premier concept formel de la liste L, nous avons $\{ag\}.Immsucc = \{\{abg\}, \{acghi\}\}$.

L'intension du concept formel $\{agh\}$ couvre $\{ag\}$ et nous avons $\{agh\} \subset \{acghi\}$.
Ainsi , il est necessaire de remplacer dans $\{ag\}.Immsucc\{\{acghi\}\}$ par $\{agh\}$.
En effet, l'ancien arc liant $\{ag\}$ à $\{acghi\}$ est un lien transitif.
Par conséquent $\{ag\}.Immsucc = \{\{abg\}, \{agh\}\}$.
Pour chaque successeur immédiat de $\{ag\}$ nous devons calculer $liste - face$ et $liste - gen$ qui contient la liste des générateurs minimaux, en utilisant la fonction $Compare - Face$.
Par conséquent :
1) $\{abg\}.liste - face = \{b\}$, $\{abg\}.liste - gen = \{b\}$.
2) $\{agh\}.liste - face = \{h\}$, $\{agh\}.liste - gen = \{h\}$.
Après le traitement de $\{ag\}.Immsucc$, nous devons manipuler la liste de successeurs du concept formel dont l'intension est égale à $\{agh\}$, c'est à dire $\{agh\}.Immsucc$.
Ainsi nous avons $\{agh\}.Immsucc = \{\{abgh\}, \{acghi\}\}$.
Nous commençons par raffiner la liste des successeurs immédiats $\{agh\}.Immsucc$.
Cependant, nous constatons que $\{acgh \subset \{acghi\}$, alors nous devons remplacer dans $\{agh\}.Immsucc$, {acghi} par $\{acgh\}$.
Ainsi $\{agh\}.Immsucc = \{\{abgh\}, \{acgh\}\}$.
Après le calcul de $liste - face$ de $\{abgh\}$,

nous obtenons :
$\{abgh\}.liste - face = \{h\} \cup \{b\}$.
Etant donné que $\{h\} \cap \{b\} = \phi$, alors $\{b\}$ ne peut pas être un bloqueur pour l'ensemble des faces $\{\{h\}, \{b\}\}$.
C'est pourquoi la liste des générateurs minimaux est remplacée par $\{bh\}$, i.e $\{abgh\}.liste - gen = \{bh\}$.
Le processus de raffinement décrit est appliqué sur $\{acgh\}.Immsucc$.
Ainsi, $\{acghi\}.Immsucc = \{abcdefghi\}$;
$\{abcdefghi\}.liste - face = \{defi\} \cup \{bdef\}$;
$\{abcdefghi\}.liste - gen = \{d, e, f, bi\}$;
Les processus de génération et de raffinement sont répétés pour toutes les transactions restantes.
La figure 3.8 décrit le treillis des concepts formels associé au contexte d'extraction K, présenté dans la table de la figure 3.6.
Certains générateurs minimaux, sont indiqués par des flèches, décorant les noeuds des concepts formels.

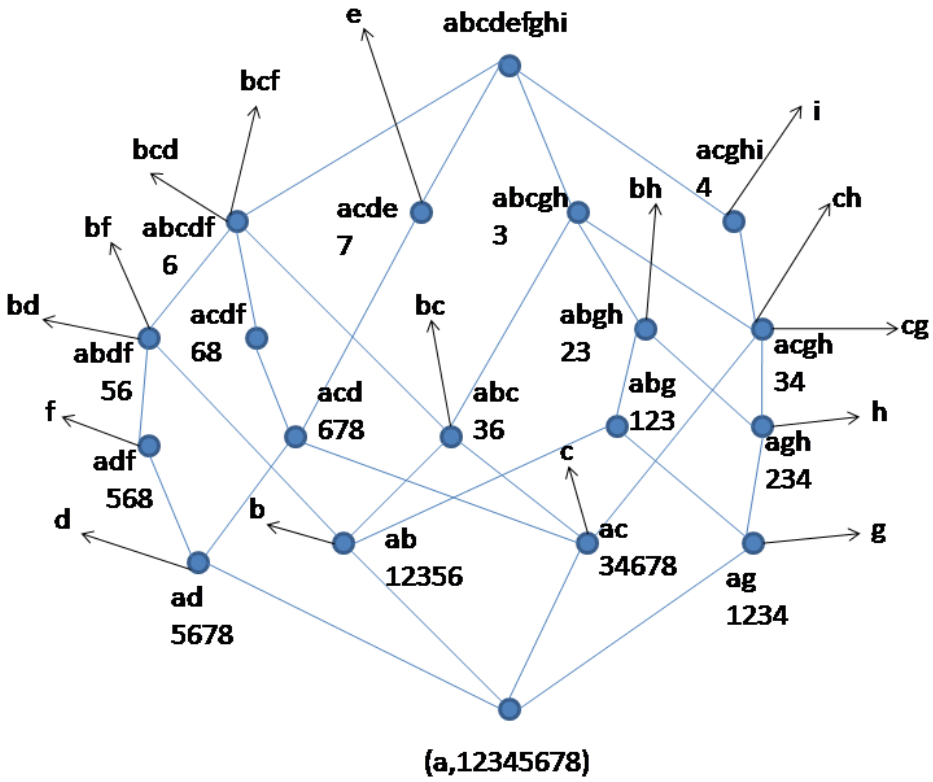

FIG. 3.8 – Treillis des concepts formels extrait du contexte K, décoré par des générateurs minimaux

3.3 Algorithme de génération de treillis quelconques

Dans ce chapitre, nous présenterons tout d'abord un algorithme [40] permettant de verifier si un graphe orienté sans circuit représente un treillis. Le meilleur algorithme connu à ce jour est celui de Goralcikova et al et avec une complexité en $O(n^{5/2})$. Celle-ci est une conséquence d'une borne de la taille du graphe de couverture d'un treillis qui a une complexité en $O(n^{3/2})$.
En effet, si celui-ci n'est pas un graphe de couverture, l'algorithme de Goralcikova et al cité plus haut permet de calculer la réduction transitive en $O(n^{5/2})$. L.Nourine a élaboré un algorithme trivial qui vérifie si T est un treillis tout en testant l'existence de la borne supérieure pour tout couple de sommets de T et admettant un élément minimal.
On déroulera une application afin de voir les différentes étapes et instructions utilisées pour la génération et la construction de ce treillis.
Avant de définir et d'écrire cet algorithme, nous définirons les propriétés et le lemme suivants :

Lemme 4 : [40] Soit $G = (X, U)$ le graphe de couverture d'un treillis.
Alors, $2^{1/2}n^{3/2}(1 + o(1))/2 \leq |X| \leq (2n)^{3/2}(1 + o(1))/3$.

Propriété 2 :[40] Soit $T = (X, <)$ et $x \in X$. Si $a \leq b$ Alors on a $a \vee x \leq b \vee x$.

Preuve : Il est clair que $\forall z, t \in X$, $z \leq z \vee t$ et $t \leq t \vee z$. Donc puisque $a \leq b$, on a $a \vee b = b$ et $x \vee b = (x \vee a) \vee b$ et par conséquent $x \vee a \leq x \vee b\square$

Propriété 3 :[40] Soit $T = (X, <)$ et $x \in X$. Alors $[\perp, x]$ (resp.$[x, \top]$) est un sous treillis de T.

Preuve : Soient $y, z \in [\perp, x]$, on a clairement $\perp \leq y \vee z \leq x$, donc $y \vee z \in [\perp, x]\square$.
Similairement, on démontre que $[x, \top])$ est un sous treillis de T.
Soit $\tau = x_1, x_2, ..., x_n$ une extension linéaire de P.
Il est clair que pour tout $k \in [1, n]$ $Y_{k-1} = x_{k-1}, ..., x_n$ est un filtre.
L'idée générale de cet algorithme est de vérifier que Y_{k-1} est un sup-demi-treillis pour k variant de n à 2 ; Il se base sur le lemme suivant :

Lemme 5 :[40] Soit $G = (X, U)$ un graphe oriénté sans circuit et $Y_{k-1} = x_{k-1}, ..., x_n$ tel que tout couple d'éléments de Y_{k-1} admet une borne supérieure.
Alors pour $z \in Y_{k-1}$, $x_k \vee z$ existe, si et seulement si $y \vee z$ tel que $y \in ImSucc(x_k)$ admet un élément minimal $w = y \vee z$.

Preuve : Si pour $z \in Y_k$, $x_k \vee z$ existe, alors pour tout $y \in ImSucc(x_k)$ $y \vee x_k \geq x_k \vee z$.
Puisque $x_k \vee z \geq x_k$, il existe $t_0 \in ImSucc(x_k)$ tel que $x_k \geq t_0$.
Il est évident que $x_k \vee z \geq z$ et donc $x_k \vee z \geq t_0 \vee z$.
Si pour $z \in Y_k$ il existe $t_0 \in ImSucc(x_k)$ tel que $w = t_0 \vee z$, soit l'élément minimal de $y \vee z$ tel que $y \in ImSucc(x_k)$ alors il est clair que $w \geq z$ et $w \geq x_k$.
Soit $v \in X$ tel que $v \geq z$ et $v \geq x_k$. Alors $v \in Y_k$ et il existe $t \in ImSucc(x_k)$ tel que $v \geq t$ et puisque $v \geq t \vee z \geq w \geq x_k \vee z$, on conclut que $x_k \vee z = w$

3.3.1 Algorithme de L.Nourine

Théorème 9 :[40] L'algorithme suivant reconnait si un graphe $G = (X, U)$ est le graphe de couverture d'un treillis en $O(n^{5/2})$.

Algorithme : reconnaissance de treillis.

Donnée : La réduction transitive d'un graphe sans circuits $G = (X, U)$, donnée par sa liste d'adjacence de prédécesseurs.

Résultat : G est-il le graphe de couverture d'un treillis ?

Début ;

Calculer une extension linéaire $\tau = x_1, x_2, ..., x_n$ de G ;

$Y = (x_n)$ est un sup-demi-treillis.

Pour $i = n - 1, 2$ **faire** le calcul des bornes supérieures entre x_i et ses successeurs.

Pour $y \in]x_i, x_n]$ **Faire** $x_i \vee y = y$; marquer y.

Calcul des bornes supérieures entre x_i et les éléments qui lui sont incomparables.

Pour $j = n, i + 1$ **Faire**

Si x_j est non marqué **alors** x_j est incomparable à x_i

Si w=min($x_i \vee z$ tel que $z \in Imsucc(x_i)$ existe

alors $x_i \vee x_j = w$.

Sinon G n'est pas le graphe de couverture d'un treillis.

Vérifier que x_1 est un élément minimum.

Fin.

3.3.2 Etude de la complexité

A chaque étape i de l'algorithme :
Le bloc 1 calcule la borne supérieure entre x_i et ses successeurs. Noter que pour tout z successeur de x_i, $x_i \vee z = z$.
Il est clair que le temps total est borné par $o(|U|)$.
Le bloc 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 incomparable à x_i, et teste s'il possède un élément minimum.
Clairement, le calcul de W coûte $o(|Imsucc(x_i)|)$ et la vérification qu'il possède un minimum revient à verifier si l'élément le plus petit de W suivant l'extension linéaire est minimum.
Puisque le test de comparabilité se fait en $o(1)$ en utilisant les bornes supérieures déjà calculées, alors la compléxité totale est en $o(|X| \,.\, |Imsucc(x_i)|)$
soit $o(|X| \,.\, |U|)$.

Exemple 4 : Afin de bien comprendre les différentes étapes et instructions utilisées par cet algorithme, nous avons jugé utile d'appliquer un exemple sur le diagramme ci-dessous représentant un ordre qui n'est pas un treillis :

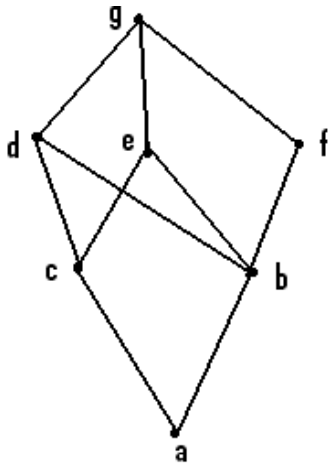

FIG. 3.9 –

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 τ qui lui sont incomparables dans G.
$Imsucc(x_i)$: l'ensemble des successeurs immédiats de x_i.
$B(a) = \{a \vee y$ tel que $y \in Imsucc(x_i)\}$
Soit $\tau = a, b, c, d, e, f, g.$ une extension linéaire
l'algorithme demarre à partir de f avant dernier élément de l'extension lineaire.

Pour f :

$Succ(f) = \{g\}$;

$Inc(f) = \Phi$;

Pour e :

$Succ(e) = \{g\}$;

$Inc(e) = \{f\}$;

$Imsucc(e) = \{g\}$;

$B(f) = f \vee g = \{g\}$;

$minB(f) = g$;

Pour d :

$Succ(d) = \{g\}$;

$Inc(d) = \{e, f\}$;

$Imsucc(d) = \{g\}$;

$B(e) = e \vee g = \{g\}$;

$minB(e) = g$;

$B(f) = f \vee g = g$;

$minB(f) = g$;

Pour c :

$Succ(c) = d, e, g$;

$Inc(c) = \{f\}$;

$Imsucc(c) = \{d, e\}$;

$B(f) = f \vee d, f \vee e = \{g\}$;

$minB(f) = g$;

Pour b :

$Succ(b) = \{d, e, f, g\}$;

$Inc(b) = \{c\}$;

$Imsucc(b) = \{d, e, f\}$;

$B(c) = \{c \vee d, c \vee e, c \vee f\} = \{d, e, g\}$;

$minB(c)$ n'existe pas, car d et e sont incomparables.

Conclusion : G n'est pas un treillis.

Conclusion générale et perspectives

Plusieurs essais et tentatives d'élaboration d'algorithmes ont été faits ; les résultats sont très encourageants, vu que les complexités de ces algorithmes approchent celles des algorithmes existants.

Ces tentatives algorithmiques ont été faites en utilisant les éléments inf-irréductibles, sup-irréductibles, les deux en même temps, les niveaux et la hauteur d'un ensemble ordonné et autres. En perspectives, en tenant compte des études et recherches actuelles, on essaiera de cerner pratiquement toutes les propriétés algébriques existantes et qui sont très nombreuses, puis, proposer un algorithme de génération de treillis consistant dont la complexité est meilleure, afin d'enrichir et de rapprocher au mieux ces treillis du domaine informatique pour une utilisation rationnelle dans les domaines d'application, qui sont sans aucun doute très nombreux.

Bibliographie

[1] H.Ait-Kaci, R.Boyer, P.Lincoln, R.Nasr. Efficient implementation of lattice operations, ACM Trans. Program. Languages Systems 11 (1) (1989) 115146.

[2] M.Barbut, B.Monjardet. Ordre et classification. Algèbre et combinatoire. Hachette, tome 1, 1970.

[3] M.Barbut, B.Monjardet. Ordre et classification. Algèbre et combinatoire. Hachette, tome 2, 1970.

[4] Ben Tekaya, Ben Yahia et Slimani. Construction d'un treillis de concept formel et de detérmination des générateurs minimaux, Proceding of the 7 th African conférence on research in computer science, Hammamet, Tunisie, 22-25 novemdre 2005.

[5] Ben Tekaya, Ben Yahia et Slimani. Algorithme de construction d'un treillis de concept formel et de detérmination des générateurs minimaux, ARIMA. CARI04, p171-193, 11/2005.

[6] C.Berge. Graphes et hypergraphes ed.Dunod, Paris.

[7] K.Bertet. Autour de quelques composantes de la theorie des treillis, Laboratoire L31, Universite de la Rechelle, journée de L'ANR DAG, 02/06/2010, Clermont-Ferrand.

[8] G.Birkhoff. Lattice Theory, American Mathematical Society, Providence, RI, 1st édition, 1940.

[9] G.Birkhoff. Lattice Theory, American Mathematical Society, Providence, RI, 3rd édition, 1967.

[10] J.Bordat. Calcul pratique du treillis de Galois d'une correspondance , Mathématiques, Informatiques et Sciences Humaines, vol. 24, n 94, 1986, p.31-47.

[11] M.Carvallo. Monographe des treillis et algèbre de boole, Gauthier-Villars, Paris 1966.

[12] M.Chein. Algorithme de recherche de sous-matrice premire dune matrice, Bull. Math. R. S. Roumanie 13 (1969).

[13] V.Dahl, A.Fall. Logical encoding of conceptual graph lattice, in : G.W. Mineau, B. Moulin, J.F. Sowa (Eds.), Proc. 1st Internat. Conference on Conceptual Structures, August 47, Universit Laval, Quebec, 1993.

[14] B.A. Davey, H.A. Priestley. Introduction to Lattices and Orders, 2nd edn., Cambridge University Press, Cambridge, 1991.

[15] J.Dubreiley. Lecons d'algèbre moderne, DUNOD, Paris 1964.

[16] H.Faure, E.Heurgon. Structures ordonnées et algèbre de boole, Gauthier-Villars, Paris 6ème, 1971.

[17] H.Faure, B.Lemaire. Mathématiques pour l'informaticien, Gauthier-Villars, Paris, Tome1, 1973.

[18] A.Floc'h, L.Fisette et R.Missaoui. Un algorithme efficace de construction de générateurs pour l'identification des règles d'association, num special de la revue des nouvelles technologies de l'information, vol 1.n1, editions cépadues, 2003, p. 135 - 146.

[19] H.Fu, E.M.Nguifo. Etude et conception d'algorithmes de génération de concepts formels, Revue Ingénierie des Systèmes d'Information, vol. 9, no 3-4, 2004, p.109132, Hermes-Lavoisier.

[20] B.Ganter, R.Wille. Formal Concept Analysis, Springer-Verlag, Heidelberg, 1999.

[21] B.Ganter. Two basic algorithms in concept analysis, Technical Report, No. 831, Technische Hochschule Darmstadt, 1984.

[22] B.Ganter, S.O.Kuznetsov. Stepwise construction of the DedekindMacNeille, in : Proc. Conceptual Structures : Theory, Tools and Applications, Lecture Notes in Computer Sci.,Vol. 1453, Springer, Berlin, 1998, pp. 295302.

[23] R.Godin, J.Gecsei. Lattice model of browsable data spaces, Inform. Sci. 40 (1996) 89116.

[24] R.Godin, H.Mili. Building and maintaining analysis-level class hierarchies using Galois lattices, in : Proc. OOPSLA93, 1993, pp. 394410.

[25] R.Godin, R.Missaoui, H.Alaoui. Learning algorithms using a Galois lattice structure, in : Proc. 1991 IEEE International Conference on Tools for AI, San Jose, CA, 1991, pp. 2229.

[26] R.Godin, R.Missaoui, A.April. Experimental comparison of navigation in Galois lattice with conventional information retrieval methods, Internat. J. Man. Machine Studies 38 (1993) 747767.

[27] M.Habib, R.Medina, L.Nourine, G.Steiner. Efficient algorithms on distributive lattices, RR, LIRMM 95-033, June 1995.

[28] M.Habib, L.Nourine. Tree structure for distributive lattices and its applications, Theoret. Comput. Sci. 165 (2) (1996) 391405.

[29] C.Jard, G.V.Jourdan, J.X.Rampo. Computing on-line the lattice of maximal antichains of posets, Order 11 (3) (1994) 197210.

[30] J.Kuntzmann. Algèbre de boole, Dunod, Paris, 1953.

[31] L.Lesieur, R.Croisot. Leçon sur la théorie des treillis, Les structures algébriques et treillis géometriques, Gauthier-Villars, Paris, 1953.

[32] J.Levy-Bruhl. Introduction aux structures algèbriques, DUNOD,Paris, 1968.

[33] S.Mac Lane, G.Birkhoff. Algèbre. Structures fondamentales, Gauthier-Villars, Paris 6ème, Tome 1, 1971.

[34] S.Mac Lane, G.Birkhoff. Algèbre. Les grands théorèmes, Gauthier-Villars, Paris 6ème, Tome 2, 1971.

[35] M.Missikoff, M.Scholl. An algorithm for insertion into lattices : Application to type classification, June 1989.

[36] E.M.Norris. An algorithm for computing the maximal rectangles of a binary relation, J. ACM 21 (1974) 356366.

[37] L.Nourine, O.Raynaud. A fast algorithm for building lattices, Information Processing Letters, no 71, 1999, p.199204.

[38] L.Nourine. Une structuration algorithmique de la théorie des treillis- Habilitation á diriger des recherches. PHD Thesis, Univérsité Montpelier II, 13/07/2000.

[39] L.Nourine, O.Raynaud. A fast incremental algorithm for building lattices. Journal of exprimental and theoritical artifitial intéligence, 14, 217-227. 2002.

[40] L.Nourine. Introduction à l'algorithmique des treillis. LRMM, CNRS et Univérsité de Montpelier II, 28/01/2001.

[41] N.Pasquier. data mining : algorithmes d'extraction et de réduction des règles d'association dans les bases de données, Doctorat d'univérsité de Clermont FerrandII, France 2000.

[42] D.Ponasse, J.C.Carega. Algorithme et topologie Booléenne, Masson, Paris, 1979.

[43] Y.Renaud. Quelques aspects algorithmiques sur les systèmes de fermeture, Thèse de Doctorat de Blaise clermont Ferrand II, 08/12/2008.

[44] P.Ribemboim, T.Y.Kong. Channing of partially ordred sets, C.R.Acad.Sci. Paris, t. 319, Série I, p.533-537, 1994.

[45] D.Richard. Treillis de boole, Anneaux booleens, Algèbre de boole. Univérsité de Clermont Ferrand.

[46] B.Sadi. Suite d'ensembles partiellement ordonnés, ARIMA, vol.4, 2006.

[47] D.Talem, B.Sadi. Calcul d'invariant dans les ensembles partiellement ordonnés. COSI'2009. Annaba 25 - 27 Mai, p.87-96.

[48] P.Valtchev, R.Missaoui, P.Lebrun. A fast algorithm for Building the Hasse Diagram of a Galois Lattice, Proceeding of the colloque LACIM 2000, Montreal (CA), september 2000, p.293-306.

[49] G.Villars, Mouton. Combinatoire, Graphes et Algèbre, DUNOD, Paris 6ème, 1973.

[50] R.Wille. Restructuring lattice theory : An approach based on hierarchies of contexts, in : I. Rival (Ed.), Ordered Sets, NATO ASI, No. 83, Reidel, Dordrecht, 1982, pp. 445470.

[51] R.Wille. Lattices in data analysis : How to draw them with a computer, in : I. Rival (Ed.), Algorithms and Orders, Kluwer Academic Publishers, Dordrecht, 1989, pp. 3358.

[52] M.J.Zaki, S.Parthasarathy, M.Ogihara, W.Lie. New algorithms for fast discovery of association rules, Technical Report, University of Rochester, 1997.