RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ DE GHARDAIA

N d'ordre
N de série

FACULTÉ DES SCIENCES ET TECHNOLOGIES
DÉPARTEMENT DES MATHÉMATIQUES ET INFORMATIQUES

Mémoire présenté en vue de l'obtention du diplôme de

MASTER

Domaine : INFORMATIQUE

Spécialité : INTELLIGENCE ARTIFICIELLE POUR L'EXTRACTION DE CONNAISSANCES

par :

Eloualid REZZAG
Kaddour BAHAZ

Thème

Treillis de Galois : algorithmes de construction

Soutenue publiquement le : 25/06/2018
Devant le jury :

Samir CHABBI	MA	Univ.Ghardaia	Président
Messaoud BETKA	MA	Univ.Ghardaia	Examinateur
Abdelkader BOUHANI	MA	Univ.Ghardaia	Examinateur
Slimane OULAD-NAOUI	PhD	Univ.Ghardaia	Encadreur
Djelloul ZIADI	Prof	Normandie Université (France)	Co-Directeur

ANNÉE : 2017/2018

Remerciements

Louange à notre Seigneur *ALLAH* qui nous a doté de la merveilleuse faculté de raisonnement. Louange à notre créateur qui nous a incité à acquérir le savoir. C"est à lui que nous adressons toute notre gratitude en premier lieu.

Nos vifs remerciements accompagnés de toute notre gratitude vont tout d'abord à nos promoteurs M. Djelloul Zaidi et M. Slimane Oulad Naoui, Pour nous avoir proposé ce sujet, pour les conseils qu'ils n'ont cessé de nous prodiguer et surtout pour la confiance qu'il nous ont accordé et pour leurs patience pour la réalisation de ce projet.

Nous remercions particulièrement les honorables membres de jury qui ont pris la peine de lire et d'évaluer ce mémoire.

Nous tenons aussi à remercier l'ensemble des enseignants de l'université de Ghardaia, sans exception, ainsi que les employés qui ont rendu plus confortable notre formation au sein de université.

Sans oublier de remercier tous ceux qui ont collaboré de près ou de loin à l'élaboration de ce modeste travail.

Un grand **MERCI** à tous les membres de nos familles respectives et spécialement nos parents qui nous ont soutenu tout au long de nos études et qui ont fait en sorte, par leur amour, leur affection et leur soutien financier, que nous puissions avoir les meilleures conditions possibles pour terminer nos études et aller vers l'avant. Qu'ils trouvent ici notre gratitude et notre amour pour eux.

Dédicace

À mes parents..

Eloualid REZZAG

Dédicace

Grace à Dieu voila notre travail terminé et il est temps pour moi de partager ma joie avec tous ceux qui mont soutenu et encourager. A vous, ma mère et mon père qui ont consacré leur vie à notre éducation et à faire notre bonheur. A touts mes frères. A toute ma famille. A tous mes amis. A mon binôme Eloualid et sa famille.

BAHAZ Kaddour

ملخص

إن ترتيب البيانات أو تخزينها بنظام معين يسمح بإستغلال أفضل لها، عندما يثبث فعاليته في عديد فروع المعرفة و العلوم، قد يصبح هدفا في حد ذاته . سنحاول في هذا العمل المتواضع تسليط الضوء على واحد من أهم المواضيع في هذا الإتجاه ، نقصد ما يعرف بشبكات جالوا التي تعتبر الحجر الأساسي لفرع جديد من الرياضيات يدعى اصطلاحا بالتحليل الصوري للمفاهيم.

سنحاول في هذه المذكرة القيام بدراسة نظرية و تطبيقية لبعض خوارزميات إنشاء مخططات جالوا كما سنحاول تقديم نموذج موحد مبني على السلاسل الصورية و الآليات ذات الأوزان.

كلمات مفتاحية

شبكات جالوا، التحليل الصوري للمفاهيم، الأنماط الثنائية، توحيد الخوارزميات، التنقيب في البيانات، السلاسل الصورية، الآليات ذات الأوزان.

Résumé

Organiser les données ou les stocker avec un système spécifique qui permet une meilleure utilisation de celui-ci, s'il est prouvé son efficace dans de nombreuses branches de la connaissance et de la science, peut devenir une fin en soi. Dans ce travail humble, nous allons essayer de mettre en évidence l'un des sujets les plus importants dans cette direction, nous entendons ce qu'on appelle le treillis de Galois, qui constitue la pierre angulaire d'une nouvelle branche des mathématiques appelée l'analyse formelle des concepts. Nous essaierons dans ce mémoire d'étudier et appliquer certains algorithmes de construction de treillis de Galois et nous tenterons de présenter un modèle unifié basé sur des séries formelles et les automates à multiplicité.

Mots clés

Treillis de Galois, Analyse formelle de concepts, Contexte binaire, Unification d'algorithmes, Fouille de données, Séries formelles, Automates à multiplicités.

Abstract

Organizing the data or storing it with a specific system that allows a better use of it, if proven effective in many branches of knowledge and science, can become an end in itself. In this humble work, we will try to highlight one of the most important topics in this direction, we hear what is called the Galois lattices, which is the cornerstone of a new branch of mathematics called the formal analysis of concepts. We will try in this thesis to study and apply some Galois lattice construction algorithms and we will try to present a unified model based on formal series and multiplicity automata.

Keywords

Galois lattices, Formel concepts analysis, Binary context, Algorithm unification, Data mining, Formal series, Weighted automata.

Table des matières

Remerciement et dédicaces i
Résumés iv
Tables des marières vii
Liste des tableux viii
Liste des figures ix
Liste des algorithmes xi
Listedes abreviations xii
Introduction 1

1 Préliminaires **3**
1.1 Rappel mathématique 3
1.1.1 Ensembles et relations 3
1.1.2 Correspondance de Galois 6
1.2 Concepts pour les algorithmes sur les treillis 8
1.2.1 Ordre lexicographique 8
1.2.2 Graphes 9
1.2.3 Structures Algébriques 11
1.3 Structures de données Pour les treillis 14
1.3.1 Graphe orienté acyclique 14
1.3.2 Diagramme de Hasse 14
1.3.3 Trie 15

2 Analyse Formelle de concepts **17**
2.1 Contexte formel 17
2.1.1 Correspondance de Galois sur un contexte binaire 18
2.1.2 Concept formel 19
2.2 Treillis de concepts 19
2.3 FCA historique 20

3 Algorithmes de construction de treillis de Galois **23**
3.1 Algorithme de Bordat . . . 24
3.1.1 Couverture . . . 26
3.1.2 Recherche Insertion . . . 27
3.2 Algorithme de Nourine et Raynaud . . . 29
3.2.1 Calcul des concepts . . . 29
3.2.2 Calcul du graphe de couverture . . . 31
3.3 Algorithmes Incrémentaux . . . 33
3.3.1 Idées de base . . . 33
3.3.2 Algorithme de Godin et al. . . . 34
3.3.3 Algorithme . . . 36

4 Expérimentation **39**
4.1 Corpus . . . 39
4.1.1 Générateur de contexte . . . 39
4.2 Difficultés . . . 40
4.3 Tests . . . 41

5 Ébauche d'unification **44**
5.1 Modélisation . . . 44
5.2 Motifs fermés comme un polynôme . . . 45
5.3 Approche du totalité de la base . . . 47
5.4 Approche incrémentale . . . 49
Conclusion générale . . . 52
Annexe . . . 53
5.4.1 Parties maximales d'un Graphe bipartie . . . 53

Liste des tableaux

2.1 Contexte $\mathcal{C}(E, F, R)$ 18
2.2 Base de transaction 18

3.1 Context en Base de transaction 29

5.1 Base de données de transactions et les polynômes associés 48
5.2 Base sans la transaction bcf 50

Table des figures

1.1 Correspondance de Galois antitone 8
1.2 Correspondance de Galois isotone 8
1.3 Graphe bipartie 11
1.4 Diagramme de Hasse 15
1.5 Trie cousu 16

2.1 Diagramme de Hasse de contexte2.1 20

3.1 Trie décoré par les extentions 30

4.1 Fonction main de programme d'exécution 41
4.2 Test pour un contexte carré 42
4.3 Test pour un contexte à dix attributs 43

5.1 Un AMP associé à l'exemple de référence 49
5.2 Un AMP realise le polynome de la base de table 5.2 51
5.3 Graphe bipartie 54
5.4 Graphe tripartie 54

List of Algorithms

1 Bordat 25
2 Génération de couverture 27
3 Recherche Insertion 28
4 Génération des Concepts : GeneConcepts() 31
5 Graphe de couverture : CouverGraphe() 32
6 Nourine 33
7 Godin et al 35
8 Galois 37

9 Parties maximales 55

Liste des abréviations

FCA	Formel Concept Analysis
extent	Extention d'un concept
intent	Intention d'un concept
DAG	Directed Acyclic Graph
sup	supremum : plus petit majorant
inf	infemum : plus grand minorant
max	Plus grand element d'un ensemble
min	Plus petit element d'un ensemble
poset	partially ordered set
PSM	Polynôme de sous-sequences de motif
PSB	Polynôme de sous-sequences de base
AMP	Automate a multiplicité préfixiel

Introduction

Retrouver, comprendre et extraire les informations et les connaissances contenues dans une collection de données (dataset) est un besoin essentiel dans plusieurs branches de l'informatique (et dans d'autres disciplines) qui nécessitent l'invention des outils permettant de faciliter ces tâches, incluant entre outres des formats de stockage appropriés.

En ce sens l'analyse de concepts formels (**FCA**) [1] grâce à sa simplicité, son élégance et sa polyvalence peuvent jouer un rôle important et aider à combler le vide de ces besoins.

La **FCA** est une méthode pour l'analyse des données, la représentation des connaissances et la gestion de l'information qui a un grand nombre d'applications dans de nombreuses disciplines, telles que la linguistique, le génie logiciel, la psychologie, l'intelligence artificielle et la recherche d'informations...

La **FCA** a été inventée par Rudolf Wille (Wille, 1982) au début des années 80. Des travaux sur l'utilisation des treillis en mathématiques et informatique existent depuis l'introduction des correspondances de Galois [2] par Birkhoff [3] dans les années 40 citons par exemple les travaux de Barbut et Monjardet (Barbut, 1970) Et depuis ce temps, ce domaine a tiré l'intention de plusieurs chercheurs et l'intérêt de la **FCA** est accru de façon considérable (Priss, 2006).

Selon Carpineto et al. (Carpineto and Romano, 2004) l'analyse de concepts formels diffère de l'analyse statistique dans le sens au l'accent est mis sur la reconnaissance et la généralisation des similarités structurelles, telles que la relation d'inclusion à partir des descriptions de données et non sur les manipulations mathématiques de la distribution des probabilités.

L'utilisation de la **FCA** permet de transformer une collection d'objets décrits par un ensemble de propriétés (Wille a donné le nom de contexte à cette collection) à une structure très efficace appelée treillis de concepts (ou treillis de Galois), où chaque concept représente un sous-ensemble de la collection des objets décrits par des propriétés partagées.

1. pour : formel concept analysis
2. Évariste Galois (1811-1832) : mathématicien français, l'un des fondateurs de la théorie des structures algébrique
3. Garrett Birkhoff(1911-1996) : mathématicien américain.

Cette structure peut être extraite de plusieurs types de données homogènes ou hétérogènes (textuelles, semi-structurées ou structurées), et ensuite utilisée pour prendre en charge divers types de tâches de gestion de contenu.

Une remarque importante et qu'un concept d'un contexte peut-être vu comme une clique dans un graphe ou un rectangle maximal dans la matrice décrivant la relation binaire du contexte.

Tout cela a conduit les chercheurs à développer plusieurs algorithmes de construction des treillis de Galois et ces applications.

Le but de notre travail est l'étude des différentes approches utilisées par ces algorithmes (à travers l'implémentation de quelques-uns) et l'introduction d'un nouveau modèle basé sur les séries formelles et les automates à multiplicité afin d'unifier les différentes approches.

Plan de travail

Notre manuscrit est composé de quatre chapitre.

Dans le premier chapitre, nous rappelons les différents outils mathématiques nécessaires à la compréhension du reste du mémoire.

Dans le deuxième chapitre, nous rappelons l'analyse formelle de concepts.

Le troisième chapitre sera consacré à l'étude des principaux algorithmes de construction de Treillis de Galois.

Dans le quatrième chapitre, nous exposons une étude comparative des approches que nous avons implémenté et nous concluons se chapitre par une proposition d'un modèle basé sur les séries rationnelles dans un but d'unification.

Chapitre 1

Préliminaires

En tant qu'une application de la théorie des treillis, l'analyse formelle de concepts (**FCA**) a besoin des connaissances de base sur les ensembles, les relations binaires, et d'autres outils mathématiques.

De plus, l'étude des algorithmes sur les treillis de Galois nécessite la compréhension de structure informatiques tels que les graphes orientés acycliques, arbres lexicographiques...

Ce chapitre préliminaires, est composé de trois sections. La première section est un rappel mathématique générale. La seconde donne quelque concepts concernant les algorithmes de treillis. La troisième section est consacrée à la présentation des outils informatiques ainsi que les structures de données utilisés par ces algorithmes.

1.1 Rappel mathématique

La plupart des définitions et rappels de ce chapitre sont tiré de livre "Éléments de mathématiques discrètes" (Frécon, 2002) et de la thèse de notre M. Slimane Oulad Naoui (Oulad-Naoui, 2018), ainsi que d'autre articles spécialisés.

1.1.1 Ensembles et relations

Un *ensemble* (noté E ou F) est un regroupement en une entité d'objets distincts. Les objets appartenant à un ensemble sont dits éléments ou membres de l'ensemble, si x est un élément de E on dit que x appartient à E ($x \in E$). Dans le cas contraire, x n'appartient pas à E, noté $x \notin E$. Un ensemble est défini de trois façons :

En extension s'il est défini par la liste de ses éléments.

Par abstraction $E = \{x \mid K(x)\}$, où $K(x)$ est un prédicat.

Par compréhension $E = \{x \in U \mid p(x)\}$

La différence entre le deuxième et le troisième cas est que dans troisième le prédicat $p(x)$ est lié à un ensemble U plus vaste que E (Frécon, 2002).

Pour un ensemble E fini [1]. On appelle cardinalité le nombre d'éléments de sa liste, notée $|E|$. L'ensemble vide(noté $\emptyset$ ou $\{\}$) est l'ensemble ayant 0 élément($|\emptyset| = 0$), ou l'ensemble dont le prédicat est toujours faux.

Deux ensembles sont égaux s'ils ont la même extension, sinon ils sont dits différents. L'ensemble E est inclus au sens large [2] dans l'ensemble F (E est une partie ou sou-ensemble de F) si et seulement si tout élément de E est un élément de F , que l'on note $E \subseteq F$ ou $F \supseteq E$. L'ensemble des parties [3] est l'ensemble $\{A \mid A \subseteq E\}$ noté $\mathcal{P}(E)$ ou 2^E et son cardinal est égal à $2^{|E|}$.

Les opérations ensemblistes : l'union($\cup$), l'intersection($\cap$), la différence($\setminus$) et le complément par rapport à un ensemble référentiel U ($\complement_U$) sont définit comme suit :

$$\begin{aligned} A \cup B &= \{x \mid (x \in A) \vee (x \in B)\} \\ A \cap B &= \{x \mid (x \in A) \wedge (x \in B)\} \\ A \setminus B &= \{x \mid (x \in A) \wedge (x \notin B)\} \\ \complement_U A &= U \setminus A \end{aligned}$$

Si l'intersection de deux ensembles est vide, ils sont qualifiés de *disjoints*.

Soit $\Gamma = \{X, Y, Z..\}$ une collection d'ensembles, $\bigcup \Gamma$ est l'union de tous les éléments de Γ.

On dit que Γ est une couverture de E si et seulement si $E \subseteq \bigcup \Gamma$.

On appelle partition d'un ensemble E toute couverture formée de parties de E deux à deux disjoins. Autrement dit : $\Gamma = \{X_1, X_2, ..., X_n\}$ forme une partition de E si et seulement si :

$$\forall (i,j) \quad 1 \leq i, j \leq n \qquad (\ X_i \cap X_j = \emptyset) \wedge \ (\bigcup_{k=1}^{n} X_k = E) \tag{1.1}$$

Exemple 1.1 *L'ensemble* $\{\{a, b\}, \{c\}, \{d, e, f\}\}$ *est une partition de l'ensemble* $\{a, b, c, d, e, f\}$.

Définition 1.1 *Soit* $B \subseteq E$ *et soit* $A = \{X_1, X_2, ..., X_n\}$ *une collection de* n *parties de* E.

1. Dans le cas d'un ensemble infini dénombrable le cardinal est noté par convention $\aleph_0$.
2. L'inclusion au sens stricte $\subset$ ($E \subseteq F \wedge E \neq F$) n'est pas un ordre
3. The powerset, en Anglais.

On dit que B est un bloqueur de A si pour tout X_i de A :

$$B \cap X_i \neq \emptyset \tag{1.2}$$

On dit que B est minimal si

$$\nexists B' \subset B \quad tel\ que \quad (B' \cap X_i \neq \emptyset \quad \forall i) \tag{1.3}$$

Autrement dit il n'existe pas une partie (stricte) de B qui soit un bloqueur (Pfaltz and Taylor, 2002; Tekaya et al., 2005).

Exemple 1.2 *L'ensemble $\{c, d, f\}$ est un bloqueur de $\{\{a, b, c\}, \{c\}, \{d, e, f\}\}$. L'ensemble $\{c, f\}$ est un bloqueur minimal.*

Le *produit cartésien* de deux ensembles E et F est l'ensemble des paires ordonnées formées d'un objet de E et d'un objet de F. On note : $E \times F = \{(x, y) \mid x \in E \wedge y \in F\}$.

Une *relation* R de E à F est une partie de $E \times F$. Si $E = F$, la relation est en plus dite *interne*. Si le couple $(x, y) \in R$ on écrit $R(x, y)$ ou xRy, où x est l'*origine* et y son *image*. Une relation peut être représentée de différentes façons : en listant ses couples, par une matrice binaire dite la matrice caractéristique de la relation, ou encore par un graphe (un arc lie les éléments des couples de R, qui représentent les sommets du graphe). Une relation R binaire interne sur un ensemble E est :

Réflexive	:	$\forall x \in E$	$R(x, x)$
Symétrique	:	$\forall x, y \in E$	$R(x, y) \Rightarrow R(y, x)$
Antisymétrique	:	$\forall x, y \in E$	$R(x, y) \wedge R(y, x) \Rightarrow x = y$
Transitive	:	$\forall x, y, z \in E$	$R(x, y) \wedge R(y, z) \Rightarrow R(x, z)$

Une relation binaire interne R sur un ensemble E est une *équivalence* si et seulement si elle est réflexive, symétrique et transitive. Une relation d'équivalence sur un ensemble induit une partition de celui-ci en classes d'équivalence. La *classe d'équivalence* d'un élément e de E par R est l'ensemble des éléments de E dont e est l'image.

Une relation binaire R sur un ensemble E est un *ordre*, noté $\leq$, si et seulement si elle est réflexive, antisymétrique et transitive. Si pour tout couple $(x, y) \in E : x \leq y \vee y \leq x$, l'ordre est dit *total*, dans le cas contraire il est qualifié de *partiel*, et E est dit *poset*. Un ensemble ordonné sera noté $(E, \leq)$.

On dit que $z \in E$ est un majorant (resp minorant) d'une partie $X \subseteq E$ si, pour tout $x \in X$, $x \leq z$ (resp. $z \leq x$). On appelle borne supérieure (*supremum*) $\sup(X)$ (resp. inférieure

(*infimum*) inf(X)) le plus petit des majorants de X (resp. le plus grand des minorants de X). Si on plus il appartient a X on l'appel max(X) (resp. min(X)). Un ordre $\leq$ sur un ensemble E est dit bon, si toute partie non vide de E possède un plus petit éléments.

On définit sur un ensemble dénombrable [4] ordonné par $\leq$ une relation de couverture comme suit : $x \prec z$ (on lit x précède z ou z succède x) si $x \leq z$ et $\forall y$ tel que $x \leq y$ et $y \leq z$ alors $y = x$ ou $y = z$. Si $x \prec y$ on dit que y est un successeur immédiat de x (ou x est un prédécesseur immédiat de y).

On appelle chaîne d'un *poset* E toute partie de E totalement ordonnée, et anti-chaîne toute partie d'éléments de E deux à deux non comparable.

Si pour chaque pair $(x.y)$ d'un *poset* E :

— Il existe un plus grand minorant noté $inf(x, y)$ alors la relation $\leq$ structure E en demi-treillis inférieur.

— Il existe un plus petit majorant noté $sup(x, y)$ alors la relation $\leq$ structure E en demi-treillis supérieur.

— Il existe $inf(x, y)$ et $sup(x; y)$ alors la relation $\leq$ structure E en treillis complet (Frécon, 2002).

Pour une relation R entre E et F de graphe G, une *correspondance* est le triplet (E, F, G), où E est *l'ensemble de départ*, et F *l'ensemble d'arrivée*. Une *application* f de E dans F est une correspondance (E, F, G) telle que : $\forall x \in E \; \exists! \; y \in F \mid (x, y) \in G$.

Une application est :

— *Surjective* si tout élément de l'ensemble d'arrivée a au moins un antécédent,

— *Injective* si à deux éléments distincts correspondent toujours deux images distinctes.

— *Bijective* si elle est à la fois surjective et injective

1.1.2 Correspondance de Galois

Soient E et F deux ensembles munis de deux opérations (lois de composition [5]) $\bullet$ et $\star$ respectivement. Et soit f une application de E dans F.

Définition 1.2 *Si pour tout couple* $(x, y) \in E^2$*, On a :*

$$f(x \bullet y) = f(x) \star f(y) \tag{1.4}$$

alors f *est dite un morphisme.*

4. On dit qu'un ensemble E est denombrable s'il existe une bijection entre E et une partie de $\mathbb{N}$
5. Un loi de composition est une application de $E \times E \longrightarrow E$

Remarque 1.1 — *Si f est bijective elle sera dite isomorphisme.*
— *Si $E = F$ f est alors dite endomorphisme.*
— *Un automorphisme est un endomorphisme bijectif.*
— *un morphisme conserve la structure algébrique.*

Définition 1.3 *Soient E et F deux ensembles ordonnés par $\leq$ et $\leq'$ respectivement. On dit que E est isomorphe*[6] *à F s'il existe un bijection monotone entre E et F.*

Définition 1.4 *Soit $f : A \longrightarrow B$ une fonction ou A et B sont deux ensembles ordonnés par $\leq$ et $\leq'$ respectivement, et soit $(x; y) \in A^2$.*
— *Si :*

$$x \leq y \Rightarrow f(x) \leq' f(y) \tag{1.5}$$

On dit que f est isotone(monotone croissante).
— *Si :*

$$x \leq y \Rightarrow f(y) \leq' f(x)$$

On dit que f est antitone (monotone décroissante).

Soit $(E, \leq)$ un ensemble ordonné, une application $h : E \to E$ est un *opérateur de fermeture* si elle est : isotone, extensive et idempotente. C'est-à-dire, une applications ayant les trois propriétés suivantes :

Isotone	:	$\forall x \forall y$	$x \geq y \Rightarrow h(x) \geq h(y)$
Extensive	:	$\forall x$	$h(x) \geq x$
Idempotente	:	$\forall x$	$h(h(x)) = h(x)$

Définition 1.5 *Soient $(E, \leq)$ et $(F, \leq')$ deux ensemble ordonnés, et soit $f : E \longrightarrow F$ et $g : F \longrightarrow E$ deux fonction monotones.*
— *(f, g) est une correspondance de Galois antitone si pour tout couple $(p, q) \in E \times F$. On a : $q \leq' f(p) \Leftrightarrow p \leq g(q)$.*
— *(f, g) est une correspondance de Galois isotone si pour tout couple $(p, q) \in E \times F$. On a : $f(p) \leq' q \Leftrightarrow p \leq g(q)$.*

Propriétés

Soit (f, g) un correspondance de Galois sur les ensembles E et F.
On définit sur $E \to E$ la fonction $\phi = f \circ g$ et sur $F \to F$ la fonction $\psi = g \circ f$.
Les fonctions ϕ et ψ ont les propriétés suivantes (Caspard et al., 2007) :

6. Il s'agit ici de morphisme d'ordre par analogie avec les structures algébrique

— Les deux sont des fonctions croissantes.
— Les deux sont des fonctions idempotentes.
— La fonction ψ est extensive, et la fonction ϕ l'est si la correspondance est antitone.
— La fonction ψ est un opérateur de fermeture, la fonction ϕ l'est si elle est extensive.

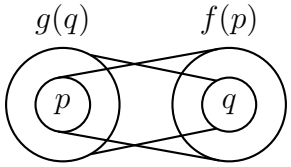

FIGURE 1.1 – Correspondance de Galois antitone

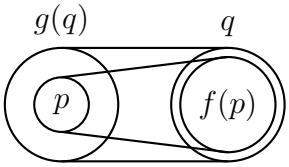

FIGURE 1.2 – Correspondance de Galois isotone

Treillis de Galois

Soit $(P, \leq)$ et $(Q, \leq')$ deux poset, et soit $f : P \longrightarrow Q$ et $g : Q \longrightarrow P$ deux fonctions tel que (f, g) soit une correspondances de Galois.

Soit l'ensemble $G = \{(p, q) \in P \times Q \mid q = f(p) \text{ et } p = g(q)\}$, et soit la relation $\preceq$ définit sur G par $(p, q) \preceq (p', q') \Rightarrow p \leq q'$. La structure $(G, \preceq)$ est appelée Treillis de Galois de correspondance (f, g).

1.2 Concepts pour les algorithmes sur les treillis

Dans cette section nous exposons des notions et concepts qui ne concerne pas l'analyse formelle de concepts mais utilisé par les algorithmes sur les treillis.

1.2.1 Ordre lexicographique

Soit $(A, \leq_A)$ un ensemble muni d'une ordre relation $\leq_A$, et soit $E = \{x_1, x_2, x_3, ...\}$ un ensemble de tuples d'éléments de A. Chaque élément x_i appartient à un produit cartésien $A^m, / \quad m \in \mathbb{N}$. (L'élément vide de E est le tuple qui ne contient aucun élément de A).

Soit $x_i = x_{1_i} x_{2_i} x_{3_i} ... x_{n_i}$ et $x_j = x_{1_j} x_{2_j} x_{3_j} ... x_{m_j}$ deux élément de E ($x_{k_l} \in A$ *et* $l \in \{i, j\}$)

On définit sur E (récursivement) un ordre $\leq_E$ comme suit :

— L'élément vide est inférieur a tout autre éléments.
— $x_i \leq_E x_j$ si et seulement si :

$$x_{1_i} \leq_A x_{1_j} \vee ((x_{1_i} = x_{1_j}) \wedge (x_{2_i} x_{3_i} ... x_{n_i} \leq_E x_{2_j} x_{3_j} ... x_{m_j}))$$

L'ordre $\leq_E$ est appelé ordre lexicographique.

Propriétés

— Si l'ordre $\leq_A$ est total alors la relation $\leq_E$ est totale.

— Si l'ordre $\leq_A$ est bon et les séquences de E sont fini alors alors l'ordre $\leq_E$ est bon.

L'ordre lexicographique permet de définir un ordre sur le monoïde A^*, il suffit de définir un ordre sur l'alphabet A.

Il permet aussi de définir un ordre total sur l'ensemble des parties d'un ensemble.

1.2.2 Graphes

Un graphe non orienté G est une paire (V, E) d'ensembles tels que $E \subseteq V \times V$, les éléments de E sont des paires(non ordonnées) d'éléments de V. Les éléments de V sont appelés *sommets* (ou *nœuds*) du graphe G, les éléments de E sont ses *arêtes*. L'ordre d'un graphe G et le nombre de ces sommets noté $|G|$. Un sommet v est incident avec une arête e si $e = (v, u)$ pour un certain sommet u. Pour une arête $e = (v, u)$, v et u sont dits *adjacents* ou *voisins*, Deux arêtes sont *adjacentes* si elles sont incidentes à un même sommet. Une *chaîne* est une suite de sommets $(v_0, v_1 \dots v_n)$ où toute paire de sommets successifs sont adjacents. Un *cycle* est une chaîne fermée, *i.e*, dont les extrémités (v_0 et v_n) coïncident. Un graphe est *connexe* s'il existe une chaîne entre toute paire de sommets, *acyclique* s'il ne possède aucun cycle. Un *arbre* est un graphe connexe acyclique. Un graphe $G'(V', E')$ est un *sous graphe* de $G(V, E)$ si $V' \subseteq V \wedge \ E' \subseteq E$. Un graphe est *complet* si tous ses sommets sont adjacents les uns avec les autres. Un sous graphe complet est dit une *clique*. Le graphe est dit *orienté* où les notions : arête, chaîne et cycle deviennent respectivement *arc*, *chemin* et *circuit* (Oulad-Naoui, 2018).

Un graphe est *étiqueté* si à ses arêtes sont associées des étiquettes quelconques souvent représentées par des lettres, des symboles ou des mots, ou encore des quantités. Le dernier cas correspond aux graphes *pondérés* (Oulad-Naoui, 2018)

Un parcours est une visite de tous les sommet d'un graphe. Il existe deux types de parcours :

— Le parcours en *largeur (Breadth-First Search (BFS))* : on commence par explorer un nœud source, puis ses successeurs, puis les successeurs non explorés des successeurs, et ainsi de suite.

— Le parcours en *profondeur (Depth-First Search (DFS))* : parcours récursif, a partir

d'un sommet source on parcours les voisins jusqu'au fin d'une chaîne puis faire des retours et exploration des sommets non visités de la même manière.

Parties maximales d'un Graphe bipartie

L'un des plus importants algorithmes de construction de treillis de Galois est basé sur le calcul des parties maximales d'un graphe bipartie (Bordat, 1986). Nous présentons ici une technique de calcul de ces parties.

Soit $G(V, E)$ un graphe dont V est un ensemble de sommets est E un ensemble d'arêtes.

On définit une fonction successeur δ tel que pour tout $x \in V$ on a :

$$\delta(x) = \{y \in V \mid (x, y) \in E\} \tag{1.6}$$

et si $X \subseteq V$ alors :

$$\delta(X) = \{y \in V \mid \exists x \in X, (x, y) \in E\} \tag{1.7}$$

G est un graphe bipartie s'il existe une partition de V en deux ensembles A et B tel que : $\delta(A) = B$ et $\delta(B) = \emptyset$. Dans la suite si G est bipartie on le note : $G(A, B, \delta)$.

G est un graphe tripartie s'il existe une partition de V en trois ensembles A, B et C tel que : $\delta(A) = B$, $\delta(B) = C$ et $\delta(C) = \emptyset$. Si G est tripartie on le note : $G(A, B, C, \delta)$.

Parties maximales Soit $G(U, V, \delta)$ un graphe bipartie de fonction successeur δ. Les parties maximales de G sont les ensembles des successeurs qui sont maximales au sens de l'inclusion, autrement dit : Pour $x \in U$, $\delta(x)$ est maximale si et seulement si :

$$\forall y \in U, \delta(x) \not\subset \delta(y)$$

Exemple 1.3 *Dans la figure 5.3, les parties maximales sont :* $\delta(1) = \{b, c\}$ *et* $\delta(3) = \{a, b, g\}$. *La partie* $\delta(4) = \{a, g\}$ *n'est pas maximale car elle est incluse dans* $\delta(3)$, *idem pour* $\delta(5)$, *et* $\delta(6) \subset \delta(1)$.

Une technique de calcul des partie maximales est la suivante :

A partir de $G(U, V, \delta)$ on construit un graphe tripartie $G(U, V, W, \mu)$ où W est un ensemble en bijection avec U, chaque élément x de U a un équivalent noté x' dans W. La fonction successeur μ est définie comme suit : $\mu(x) = \delta(x)$ si $x \in U$ et pour tout $x' \in W$, $\mu^{-1}(x') = V - \mu(x)$. On a alors la propriété suivante :

$\mu(x_1) \not\subset \mu(x_2)$ si et seulement s'il existe un chemin entre x_1 etx'_2 dans $G(U, V, W, \mu)$.

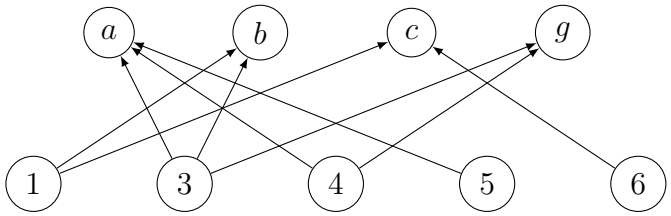

FIGURE 1.3 – Graphe bipartie

1.2.3 Structures Algébriques

Monoïde

Un monoïde $< A, \star, \epsilon >$ est un ensemble A muni d'une opération $\star$ associative et un élément neutre ϵ.

Si $\star$ est commutative on dit que le monoïde est commutatif.

Un groupe est un monoïde dont chaque élément x possède un symétrique $\forall x \in A, \exists y \in A$ tel que $x \star y = y \star x = \epsilon$.

Exemple 1.4 *L'ensemble Σ^* est un monoïde dans l'opération est la concaténation et l'élément neutre est le mot vide ϵ.*

L'ensemble des entiers naturels $\mathbb{N}$ est un monoïde dans l'opération est l'addition $(+)$ et l'élément neutre est 0.

Semi-Anneau

Un Semi-Anneau est un quintuplet $(E, +, \times, 0, 1)$ vérifiant les axiomes suivants :

— $(E, +, 0)$ est un monoïde commutatif.

— $(E, \times, 1)$ est un monoïde.

— $(\times)$ est distributif par rapport au $(+)$

— 0 est absorbant pour le produit, autrement dit : pour tout $x \in E \quad x \times 0 = 0 \times x = 0$.

Si $\times$ est commutatif alors le semi-anneau est commutatif. Et si $(+)$ est idempotente alors le semi-anneau est idempotent.

Exemple 1.5 *Le semi-anneau des entiers naturels* $(\mathbb{N}, +, \times, 0, 1)$. .

Le semi-anneau de Boole $(\{0, 1\}, \vee, \wedge, 0, 1)$. .

Séries formelles

Soit A un monoïde est $\mathbb{K}$ un semi-anneau.

On appelle série formelle toute application $\mathbb{S} : A \longrightarrow \mathbb{K}$. L'ensemble de toutes les séries formelles sur A est noté $\mathbb{K}\langle\langle A\rangle\rangle$. L'image de $w \in A$ est notée $\langle \mathbb{S}, w\rangle$ et appelée coefficient de w dans $\mathbb{S}$.

Une séries formelle peut être représentée par une somme :

$$\mathbb{S} = \sum_{w \in A} \langle \mathbb{S}, w\rangle w \tag{1.8}$$

L'étendue (ou support) d'une série formelle est le sous ensemble d'éléments de A dont le coefficient dans $\mathbb{S}$ n'est pas nulle. On appelle polynôme toute série dont l'étendue et fini. L'ensemble de tous les polynômes est noté : $\mathbb{K}\langle A\rangle$.

Une série formelle généralise la notion de langage formel en fournissant en plus une information de quantification pour les mots. En effet, les langages formels classiques sont considérés comme des cas particuliers des séries formelles vu que leur domaines de quantification sont l'ensemble des booléens $\mathbb{B}$. On peut dire qu'un mot appartient à un langage quand son coefficient est 1 ; sinon, son coefficient est nul et le mot n'appartient pas au langage en question.

Dans une série formelle, nous mesurons par un coefficient l'appartenance d'un élément donné à un ensemble. La valeur assignée par une série formelle à un élément peut modéliser un poids, une multiplicité, un coût comme le temps ou la distance ou tout autre quantification ou estimation associée à l'élément dans la série. Dans l'équation 1.8, w dénote un élément et $\langle \mathbb{S}, w\rangle$ symbolise son coefficient ou image par la série $\mathbb{S}$; $\langle \mathbb{S}, w\rangle w$ est, quant à lui, un terme de la série (Oulad-Naoui, 2018).

Soit $\mathbb{S}$ et $\mathbb{S}'$ deux séries et soit $k \in \mathbb{K}$.

On définit sur $\mathbb{K}\langle\langle A\rangle\rangle$ les opérations suivantes :

(i) produit externe :

$$\langle k\mathbb{S}, m\rangle = k\langle \mathbb{S}, m\rangle \; et \; \langle \mathbb{S}k, m\rangle = \langle \mathbb{S}, m\rangle k \tag{1.9}$$

(ii) addition terme à terme :

$$\langle \mathbb{S} + \mathbb{S}', m\rangle = \langle \mathbb{S}, m\rangle + \langle \mathbb{S}', m\rangle \tag{1.10}$$

(iii) produit de Cauchy :

$$\langle \mathbb{S}\mathbb{S}', m\rangle = \sum_{\substack{u,v \in A \\ uv = m}} \langle \mathbb{S}, u\rangle \langle \mathbb{S}', v\rangle \tag{1.11}$$

Les deux premières opérations induisent sur $\mathbb{K}\langle\langle A\rangle\rangle$ une structure de semi-module gauche et droit [7].

Automates à multiplicité

Comme les série formelles généralisent les langages formels, les automates à multiplicités (ou pondérés) généralisent les automates classiques. Dans un automate à multiplicité, nous introduisons des quantifications (ou poids) sur les transitions entre les états, et autorisons également chaque état de posséder une multiplicité d'entrée et une autre de sortie. Historiquement, les automates à multiplicité ont été largement utilisés pour modéliser de multiple systèmes dans diverses applications. Quoique les poids d'un automate à multiplicité peuvent à priori être arbitraires, leur nature (le semi-anneau de travail) est généralement dictée par le domaine d'application.

Formellement, un *automate à multiplicité* $\mathcal{A}$ sur un alphabet A à coefficients dans un semi-anneau $\mathbb{K}$ est un tuple $\mathcal{A} = (Q, A, \mu, \lambda, \gamma)$, où Q est un ensemble fini d'états, μ une fonction de Q dans $\mathbb{K}$ des poids initiaux ou d'entrée, λ une fonction de $Q \times A \times Q$ dans $\mathbb{K}$ des poids des transitions, et γ l'application de Q dans $\mathbb{K}$ des poids finaux ou de sortie. La *taille* d'un automate $\mathcal{A}$, notée $|\mathcal{A}|$, est le nombre de ses états.

Nous schématisons les automates à multiplicité de façon similaire que la représentation des automates classiques mettons en évidence les multiplicités sur les états/transitions.

Un chemin c dans un automate à multiplicité $\mathcal{A}$ est une succession de transitions : $(q_0, a_1, q_1), \ldots, (q_{n-1}, a_n, q_n)$ étiqueté par le mot $a_1 \ldots a_n$ obtenu par concaténation de symboles des transitions qui le constituent ; son poids $\omega(c)$ est le produit (en général, l'opération $\otimes$ du semi-anneau) des poids de ses transitions, en incluant le poids d'entrée de l'état initial et celui de sortie de l'état final :

$$\omega(c) = \mu(q_0)\lambda(q_0, a_1, q_1) \ldots \lambda(q_{n-1}, a_n, q_n)\gamma(q_n) \tag{1.12}$$

Posons $\mathcal{C}(u)$ l'ensemble des chemins étiquetés u, le poids d'un mot u dans un automate $\mathcal{A}$, noté par $\mathcal{A}(u)$, est la somme (en général, l'opération $\oplus$ du semi-anneau) des poids des éléments de $\mathcal{C}(u)$:

$$\mathcal{A}(u) = \sum_{c \in \mathcal{C}(u)} \omega(c) \tag{1.13}$$

Soient $\mathcal{A} = (Q_A, A, \mu_A, \lambda_A, \gamma_A)$ et $\mathcal{B} = (Q_B, A, \mu_B, \lambda_B, \gamma_B)$ deux automates à multiplicité. Une application h de Q_A dans Q_B est un *homomorphisme d'automates à multiplicité* si

7. un module est a un anneau se qu'un espace vectoriel a un corps

$\forall q \in Q_A$, elle garantit :

$$\begin{cases} h(\mu_A(q)) & = & \mu_B(h(q)) \\ h(\lambda_A(q,a,q\prime)) & = & \lambda_B(h(q),a,h(q\prime)) \\ h(\gamma_A(q)) & = & \gamma_B(h(q)) \end{cases} \tag{1.14}$$

L'extension d'une application h à un ensemble est définie en prenant l'union des images, par h, des éléments de l'ensemble considéré. Autrement dit :

$$h(Q) = \bigcup_{q \in Q} h(q) \tag{1.15}$$

Si h est, en plus, bijective, elle est dite un *isomorphisme d'automates à multiplicité.*

1.3 Structures de données Pour les treillis

1.3.1 Graphe orienté acyclique

Soit $G(V, E)$ un graphe orienté dont V est un ensemble de nœuds et E est un ensemble d'arcs.

Définition 1.6 *On appelle circuit une suite d'arcs consécutifs (chemin) dont les deux sommets extrémités sont identiques.*

La notion correspondante dans les graphes non orientés est celle de cycle (un circuit est parfois appelé cycle orienté).

Définition 1.7 *Un graphe orienté acyclique est un graphe orienté qui ne possède pas de circuit.*

Un graphe orienté acyclique est abrégé **DAG** pour Directed Acyclic Graph.

Un **DAG** est une façon simple (mais pas très efficace) de stocker un treillis (les nœuds sont les concepts et les arcs représentent l'ordre sous-jacent).

1.3.2 Diagramme de Hasse

Un treillis peut être représenté par un graphe particulier appelé diagramme de Hasse. Un diagramme de Hasse est un graphe orienté acyclique (DAG), dont les nœuds représentent les concepts et les arcs représentes la relation de couverture.

La figure 1.4. représente le diagramme de Hasse du treillis associé à l'ensemble $\mathcal{P}\{a, b, c\}$ ordonné par $\subseteq$.

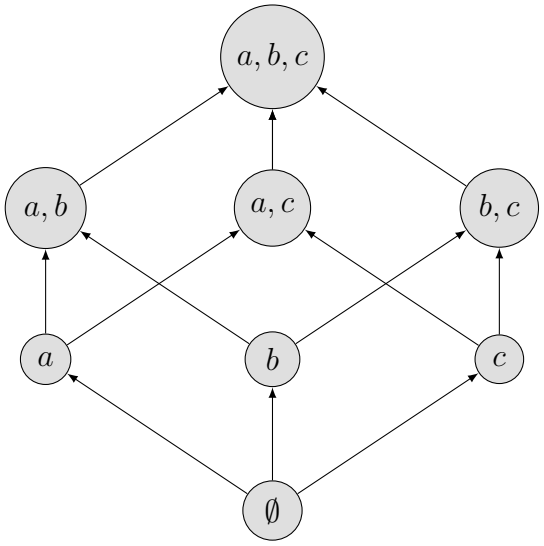

FIGURE 1.4 – Diagramme de Hasse

1.3.3 Trie

Un trie est une structure de donnée utilisée pour stocker des chaînes de caractères (ou des dictionnaires). C'est un arbre de recherche dans le principe de parcourt et le suivant :

— La racine est associée au mot vide.

— Pour chaque nœud les descendants représentent des chaînes ayants le même préfixe.

Il n'est pas nécessaire de stocker les chaînes sur les nœuds, pour connaître le mot associé, il suffit de faire un parcours depuis la racine (comme un automate).

Plusieurs algorithmes sur les treillis utilisé les tries pour stocker les concepts (Bordat, 1986; Nourine and Raynaud, 1999).

Arbre cousu

Définition 1.8 *Un arbre cousu est un arbre de recherche auquel on ajoute des pointeurs (liens)qui peuvent former un cycle,*

Pour ne pas perdre la structure de l'arbre, il faut marquer ces liens, par exemple dans un arbre binaire on ajouté un champ booléen **CousGauche** pour indiqué que le fils gauche est cousu.

On peut utiliser cette technique sur les Tries.
La figure 1.5 représente le treillis de la figure 2.1.

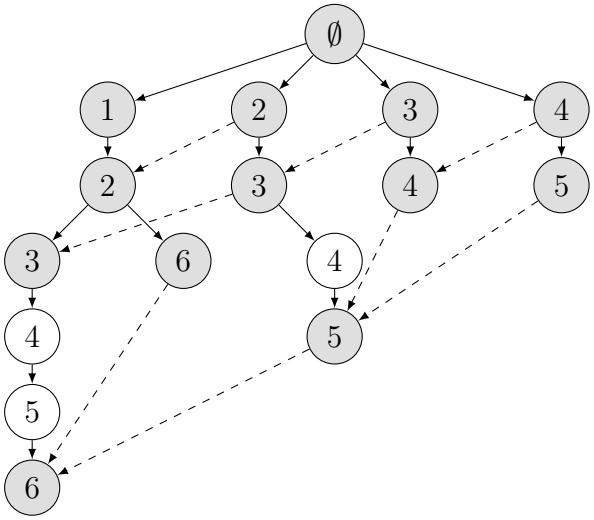

FIGURE 1.5 – Trie cousu

La structure de l'arbre est représenter par des flèches normale.
Les flèches en pointillés représentent des liens cousus.
Les nœuds en gras représentent des concepts.

Remarque 1.2 *Dans un trie, un noeud ne représente pas nécessairement un concept stocké, dans la pratique chaque noeud contient un champ (booleén par exemple) pour indiquer que c'est un concept.*

Conclusion

Dans ce chapitre nous avons présenté d'une part, les fondements mathématiques nécessaire à la compréhension de la **FCA**. Et les différents outils informatique concernant les Treillis de Galois. Le chapitre suivant sera consacré à l'analyse formelle de concepts.

Chapitre 2

Analyse Formelle de concepts

L'analyse formelle de concepts est un domaine récent de mathématiques appliquées, considéré comme une autre représentation de la théorie de treillis (introduite par Birkhoff) qui facilite son utilisation dans divers applications, selon Wille, l'objectif est : d'*"atteindre une théorie structurée qui expose les pensées formelles selon des interprétations significatives et permettre ainsi des communications et des discussions critiques de leurs contenus"* (Messaï, 2009). Dans ce chapitre on va introduire les fondements de base ainsi q'un état de l'art sur la **FCA** et ses applications.

2.1 Contexte formel

Un *contexte* formel $\mathcal{C}$ est un triplet (E, F, R) tel que :

— E est un ensemble (d'*objets*).

— F un ensemble (d'*attributs*).

— $R \subseteq E \times F$ une relation de E dans F.

Si les éléments de F prennent des valeurs binaires $\mathcal{C}$ est appelé *contexte binaire* (Njiwoua, 2005).

Exemple 2.1 *Soit :* $E = \{1, 2, 3, 4, 5, 6\}$, $F = \{a, b, c, d, e, f, g\}$ *deux ensembles.*
Soit : $R = \{(1, b), (1, c), (1, f), (2, a), (2, b), (2, c), (2, g), (3, a), (3, b), (3, e), (3, g), (4, a), (4, d), (4, e), (5, a), (5, d), (6, c)\}$ *une relation (Bordat, 1986).*
Le contexte $\mathcal{C}(E, F, R)$ *est représenté par le tableau 2.1.*

Remarque 2.1 *Dans la fouille de données un contexte binaire est appelé base de transactions. Une transaction est un sous ensemble de* F *(ensemble des attributs) identifier par un identificateur unique.*

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

TABLE 2.1 – Contexte $\mathcal{C}(E, F, R)$

Transaction Id	Attribut
1	bcf
2	$abcg$
3	$abeg$
4	ade
5	ad
6	c

TABLE 2.2 – Base de transaction

Le tableau 2.2 représente l'exemple précédent en base de transaction.

2.1.1 Correspondance de Galois sur un contexte binaire

Soit $\mathcal{C} = (E, F, R)$ un contexte binaire. On définit sur $\mathcal{P}(E)$ et $\mathcal{P}(F)$ (ordonnés par $\subseteq$) deux *antitones* :

$$f : \mathcal{P}(E) \to \mathcal{P}(F), f(X) = \{y \in F \mid \forall x \in X, (x, y) \in R\} \tag{2.1}$$

$$g : \mathcal{P}(F) \to \mathcal{P}(E), g(Y) = \{x \in E \mid \forall y \in Y, (x, y) \in R\} \tag{2.2}$$

Preuve(de l'*antitoné*) :
Soit X_1, X_2 deux parties de E tel que $X_1 \subseteq X_2$.

$$f(X_1) = \{y \in F | \forall x \in X_1 \, xRy\} \tag{2.3}$$

$$f(X_2) = \{y \in F | \forall x \in X_2 \, xRy\} \tag{2.4}$$

On a : $\forall y \in f(X_2)$, $xRy, \forall x \in X_2$, et comme $X_1 \subseteq X_2$ en déduire que $\forall x' \in X_1 \quad x'Ry$, alors $y \in f(X_1)$ soit $f(X_2) \subseteq f(X_1)$. La preuve est similaire pour g.

(f, g) est un correspondance de Galois *antitone* on peut définir les deux opérateurs de fermeture (Njiwoua, 2005; Caspard et al., 2007) :

$$\psi = g \circ f \quad , \quad \phi = f \circ g \tag{2.5}$$

Si pour un sous ensemble X de E (resp. Y de F) : $\psi(X) = X$ (resp. $\phi(Y) = Y$). X (resp.

Y) est un ensemble fermé de ψ (resp. de ϕ).

La relation $\subseteq$ structure l'ensemble des fermés de ψ (resp. de ϕ) en treillis complets. Ces deux treillis sont isomorphe (Njiwoua, 2005).

2.1.2 Concept formel

Soit X, Y deux parties de E et F respectivement. Si : $f(X) = Y$ et $g(Y) = X$ alors le couple (X, Y) est appelé *concept formel* dont X est l'*extention* et Y l'*intention*. Si (X, Y) est un concept alors X et Y sont des fermé de ψ et ϕ respectivement.

Si un contexte est vu comme une matrice binaire, un concept est un rectangle maximal (sous matrice remplie de 1).

On définit un ordre sur l'ensemble des concepts comme suit :

Soit (X, Y), (X', Y') deux concepts.

$$(X, Y) \leq (X', Y') \Leftrightarrow X \subseteq X' \vee Y' \subseteq Y \tag{2.6}$$

De la même façon on définit une relation de couverture :

$$C_1 \preceq C_2 \Leftrightarrow C_1 \leq C_2 \wedge \forall C_i \text{ tel que } C_1 \leq C_i \leq C_2 \text{ on a } \; C_1 = C_i \vee C_2 = C_i \tag{2.7}$$

2.2 Treillis de concepts

Théorème 2.1 *L'ensemble des Concepts ordonnés par la relation 2.7 forme un treillis complet, dont la borne supérieur est l'union des extentions et l'intersection des intentions, et la borne inférieur est l'intersection des extentions et l'union des intention.*

Ce théorème est appelé théorème fondamental de l'analyse formelle de concepts.

Et le treillis est appelé Treillis de Galois (ou Treillis de concepts).

Le diagramme de Hasse de contexte 2.1 est représenter par la figure 2.1

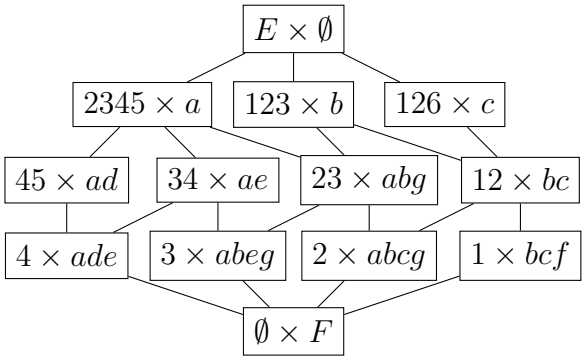

FIGURE 2.1 – Diagramme de Hasse de contexte2.1

2.3 FCA historique

Cette section est un historique de la théorie des treillis et l'analyse formelle de concepts, à partir d'informations issues de (Bertet, 2010; Monjardet, 2008; Messaï, 2009).

Avant 1970 : Treillis algébrique et contexte philosophique Même si de nombreuses disciplines peuvent être datées du temps d'Aristote, on peut trouver des prolégomènes plus proches de la FCA, par exemple, dans La Logique de Port-Royal (Arnauld et al., 1967), une ancienne logique conceptuelle, où un concept était traité comme une paire de son étendue et son intention (encore sans formalisme mathématique).

La notion de treillis définie comme une structure Algébrique munie de deux opérateurs appelés borne inférieure et borne supérieure a été introduite par Dedekind, puis oubliée.

Le terme de treillis a, quant à lui, été proposé par Birkhoff lors du premier symposium sur la structure de treillis en 1938, pour être finalement repris dans son ouvrage de référence (Birkhoff, 1940), l'introduction d'un treillis sous forme ordinale - structure ordonnée (i.e. transitive, antisymétrique et réflexive), et développée dans les travaux de Birkhoff (Birkhoff, 1967), celui-ci exhibe sous nom de *polarité* la correspondance de Galois associée a une relation binaire.

Le terme de correspondance de Galois a quant a lui été introduit par Ore (Ore, 1944), l'un des premiers algorithmes proposés est l'algorithme de Chein (Chein, 1969) qui calculer des rectangles maximaux (les fermés).

1970 : Treillis de Galois Barbut et Monjardet introduisent ainsi le terme de treillis de Galois se constitue autour d'un résultat principal qui établit que tout treillis fini est isomorphe au treillis de Galois ou treillis des concepts (Barbut, 1970), Cette bijection peut se voir comme une conséquence des propriétés de fermeture portées par toute correspon-

dance de Galois, et par conséquent s'applique également aux treillis de fermés (Caspard and Monjardet, 2003), Plusieurs travaux font apparaitre la notion d'éléments irréductibles d'un treillis qui permettent par exemple de caractériser certaines classes de treillis ex : (Norris, 1978), (Ganter, 1984), et (Bordat, 1986).

1990 à 2018 : Treillis des concepts et Analyse Formelle des Concepts Il s'agit de la restructuration dans le cadre de l'Analyse Formelle des Concepts par Wille (Wille, 1982), et finalisée par Ganter et Wille (Ganter and Wille, 1999), en tant qu'application de la théorie des treillis. Elle s'attache à étudier les concepts lorsqu'ils sont décrits formellement, c'est-à-dire que le contexte et les concepts sont complètement et précisément définis.

Le calcul des concepts d'un treillis de concepts à partir d'un contexte formel a fait l'objet de nombreux travaux de recherche qui ont abouti à la proposition d'une variété d'algorithmes dont les plus connus sont NextClosure (Ganter and Reuter, 1991), Close-by-One (Kuznetsov, 1993), Godin (Godin et al., 1995), Galois (Carpineto and Romano, 1996), (Carpineto and Romano, 2004), Nourine (Nourine and Raynaud, 1999), Divide&Conquer Valtchev and Missaoui (2001).

Chacun de ces algorithmes se distingue des autres par plusieurs critères dont la stratégie de calcul des concepts, la recherche de l'ordre entre ces concepts, les structures de données utilisées pour le stockage des résultats intermédiaires et le résultat final. Les principaux algorithmes ont fait l'objet d'une comparaison détaillée (Kuznetsov and Obiedkov, 2002). Cette comparaison a montré qu'aucun algorithme n'est meilleur que tous les autres sur tous les plans et que les performances d'un algorithme dépendent fortement des caractéristiques du contexte formel en entrée.

Utilisée la structure des treillis de Galois dans plusieurs domaines de l'informatique à savoir :

1. Recherche d'information ex : (Carpineto and Romano, 2005)
2. Intelligence artificielle ex : (Szathmary and Napoli, 2004)
3. Base de données orientées objet ex : (Gammoudi and Nafkha, 2002)
4. Fouille des données pour l'extraction des règles d'association ex : (Fu and Nguifo, 2004)
5. Services Web ex : (Messaï, 2009)
6. Extraction des connaissances à partir des données textuelles ou "Text Mining" ex : (Cherif et al., 2003)
7. La linguistique ex : (Priss, 2006)
8. L'ingénierie des logiciels ex : (Priss and Old, 2004)

En plus des autres les domaines : Médecine (Motameny et al., 2008) (Kaytoue et al., 2011) (Endres et al., 2012), Ecologie (Stumme and Maedche, 2001), Droit (Mimouni et al., 2015), Science politique (Kohler-Koch and Vogt, 2000), etc.

Beaucoup plus de documents de conférence à des conférences régulières telles que : Conférence internationale sur l'analyse de concept formel (ICFCA) [1], Concept Lattices et leurs applications (CLA) [2], ou Conférence internationale sur les structures conceptuelles (ICCS) [3].

Conclusion

Dans ce chapitre, nous avons présenté le formalisme mathématique de l'analyse formelle de concepts, ainsi qu'un état de l'art chronologique de la **FCA**. maintenant on peut entrer dans l'étude des algorithmes concernant la construction des treillis de Galois qui sera le sujet du prochain chapitre.

1. "International Conference on Formal Concept Analysis". dblp. Retrieved 2016-02-14.
2. "CLA : Concept Lattices and Their Applications". CLA. Retrieved 2015-11-14.
3. "International Conferences On Conceptual Structures – Conferences and Workshops". New Mexico State University. Retrieved 2016-02-14.

Chapitre 3

Algorithmes de construction de treillis de Galois

Le calcul des concepts d'un contexte formel est un problème difficile du fait que leur nombre est exponentiel en fonction de la taille du contexte dans le pire des cas (Pour un contexte $C(E, F, R)$ le nombre de concepts au pire des cas est $2^{|E|}$ ou $2^{|F|}$). Dans les bases réelles cette limite est rarement atteinte mais reste grande (et encore pire pour un contexte non binaire).

Basée sur un calcul probabiliste(On suppose que chaque attribut à une probabilité $p = \frac{k}{|F|}$ d'appartenir à un objet où k est un constant), la formule suivante donne une estimation du nombre de concepts (Carpineto and Romano, 2004) :

$$|\mathcal{C}| = \sum_{i=0}^{|E|} \sum_{j=0}^{|F|} \begin{pmatrix} |E| \\ i \end{pmatrix} \begin{pmatrix} |F| \\ j \end{pmatrix} p^{ij} (1-p^i)^{|F|-j} (1-p^j)^{|E|-i} \tag{3.1}$$

Dans l'équation 3.1 :

— $\mathcal{C}$ est l'ensemble des concepts.

— $\begin{pmatrix} n \\ m \end{pmatrix} = \frac{n!}{m!(n-m)!}$ le nombre de combinaisons sans répétition de n éléments pris m à m.

— p^{ij} La probabilité que les i objets partages les j attributs.

— $(1-p^i)^{|F|-j}$ La probabilité que les i objets ne partages pas d'autre attributs.

— $(1-p^j)^{|E|-i}$ La probabilité qu'il n y a pas d'autre objets qui partages les j attributs.

Pour construire le treillis complet le calcule de tous les concepts est nécessaire, et la

performance de l'algorithme est juger par la complexité de calcul d'un seul concept [1].

On peut classer les algorithmes en deux grandes classes :

1. **Algorithmes non incrémentaux** À chaque itération l'algorithme utilise la totalité de l'entrée, exemples : (chein , bordat ..).
2. **Algorithmes incrémentaux** L'algorithme utilise une partie de l'entrée, cette partie est augmentée de façon incrémentale à chaque itération, exemples : (Godin , Norris ...).

Les algorithme des deux classes utilisent quatre approches (Carpineto and Romano, 2004) :

1. **Calcul des concepts uniquement :** Certains sont basés sur l'opérateur de fermeture comme l'algorithme naïf qui exploite tout l'ensemble des parties, ou d'autres plus performant basé sur un ordre lexicographique ou autre propriétés des concepts (intersection des intentions ..), exemple : Norris .
2. **Prochains voisins :** Le diagramme de Hasse est construit au fur et à mesure qu'on calcule les concepts, commençant par la racine on calcule pour chaque concept les successeurs immédiats (parcours en largeur), exemple : Bordat.
3. **Utilisé les concepts pour construire le graphe :** Ici on calcule d'abord l'ensemble des concepts puis cet ensemble est utilisé pour construire le graphe, exemple : Nourine, Valtchev.
4. **Construction partielle :** On construit le diagramme pour une partie de contexte, ce diagramme est mis à jour en ajoutant de nouveaux objets, exemple : Godin , Carpineto et al.

Les sections suivantes représentent en détail un algorithme pour chaque approche.

3.1 Algorithme de Bordat

L'algorithme de Bordat est, peut-être, le seul existant dans l'approche : Prochains voisins. Le diagramme de Hasse est construit de façon incrémentale [2]. On commençant par le concept trivial $(\emptyset, F)$.

Pour chaque concept, l'algorithme génère les concepts voisins (couverture) et les ajoute au diagramme.

1. On parle d'une complexité qui dépende de la sortie
2. On parle de la construction de graphe, l'algorithme de Bordat est non incrémental

La construction est basée sur une structure de trie cousu (voir préliminaires) dont le parcours est lié a l'ensemble des identificateurs des objets où un ordre total est imposé.

L'ensemble des attributs est aussi totalement ordonné. Ceci est nécessaire pour le calcul des voisins.

L'algorithme utilise l'idée de théorème suivant :

Théorème 3.1 (Bordat) *$(A' \times B')$ couvre $(A \times B)$ dans $\mathcal{L}$ si et seulement si B' est une partie maximale de $G(E - A, B)$. Alors A' est l'union disjointe de A et des sommets de $G(E - A, B)$ qui admettent B' comme ensemble de successeur.*

Ici $(A' \times B')$ et $(A \times B)$ sont des concepts sur le contexte formel $C = (E, F, R)$, et $\mathcal{L}$ le treillis de Galois associé.

$G(E - A, B)$ est un graphe bipartie de $E - A$ vers B.

Couvre signifie que $(A' \times B')$ est un successeur immédiat de $(A \times B)$.

Notre version de l'algorithme est la suivant :

Algorithm 1: Bordat

Data: $C(E, F, R)$ tel que C est un contexte binaire
Result: Treillis de Galois stocké sur un Trie cousu T
begin
    T.insert$(\emptyset \times F)$
    Q.push$(\emptyset \times F)$
    /*Q est une file d'attente.*/
    while *(not Q.isEmpty)* **do**
        $c1 \longleftarrow Q.pop()$
        $c1.couvr(C, P)$
        /* la fonction couvr calcul la couverture de $c1$ sur C et la stocké sur une file P */
        while *(not P.isEmpty)* **do**
            $c2 \longleftarrow P.pop()$
            if $T.searchInsert(c1, c2))$ **then**
                $Q.push(c2)$

Le concept trivial $(\emptyset \times F)$ est inséré dans la racine du trie et dans la file Q. La pre-

mière boucle retire le dernier élément de la file (Concept en cours) est génère sa couverture (concepts successeurs). On les stockent sur une autre file P. La seconde boucle retire le dernier élément de la file P, faire une recherche sur le trie, deux cas se présentent :

Si le concept n'existe pas dans le trie alors on l'ajoute (ordre naturel avec le concept en cours).

Sinon un lien cousu est crées entre ce concept et le concept en cours.

Dans le premier cas la méthode searchInsert() renvoie vrai et le concept c_2 est alors ajouté à la file Q .

3.1.1 Couverture

Dans l'algorithme 1 la méthode *couvr*() calcule pour un concept donné c_i (le concept en cours). Le graphe associé et extraire les parties maximales puis calcule tout les successeurs immédiats de c_i.

L'algorithme est le suivant :

Algorithm 2: Génération de couverture

Data: Concept en cour c_i , Context$C$

Result: Successeur immédiat de c_i stocké dans une file P

initialisation

for $t_n \in E \setminus c_i.extent$ **do**

  $max_n \longleftarrow true$

for $t_n \in E \setminus c_i.extent$ **do**

  if $\neg max_n$ **then**

    continue ;

  $\mu_n \longleftarrow t_n$

  for $t_k \in E \setminus c_i.extent$ **do**

    if $t_k = t_n$ **then**

      continue ;

    if $t_k.extent \subseteq t_n.extent$ **then**

      $max_k \longleftarrow false$

      if $t_k.extent = t_n.extent$ **then**

        $\mu_n \longleftarrow \mu_n + t_k$

    else

      $max_n \longleftarrow false$

  if max_n **then**

    $c \longleftarrow$ gener(μ_n)

    $P.Push(c)$

L'ensemble μ_n contient les prédécesseurs d'une partie maximale.

La complexité de l'algorithme 2 est au pire des cas E^α avec $2 \leq \alpha < 3$(voir section 5.4.1).

Cette borne sera loin de la complexité réelle si le nombre des parties maximales est petit ce qui est le cas dans la pratique (Bordat, 1986) et si l'un des poins fort de cet algorithme.

3.1.2 Recherche Insertion

La méthode searchInsert() nécessite deux paramètres le concept en cours et un successeur immédiat. Elle renvoie vrai (boolien) si ce dernier est nouveau (n'exist pas dans le trie), faux sinon. Bordat propose une méthode pour connaître si un tel concept est nouveau :

Soit $c(A, B)$ les concepts en cours est $c'(A', B')$ un successeur immédiat, où A, A' representent l'extension et B, B' l'intension.

On sait que $A \in A'$ (et $B' \in B$) .

Deux cas se présentent : Les éléments de $A' \setminus A$ sont tous supérieurs aux éléments de A [3] dans ce cas le concept est nouveau (n'existe pas dans le trie). Sinon le concept existe déjà dans le Trie (Bordat, 1986).

Ceci n'est vraie que si les successeurs arrivent dans l'ordre (lexicographique) et il faut prendre ça en compte ceci dans l'implémentation.

Algorithm 3: Recherche Insertion

Data: deux concept (c_1, c_2) , Trie T
Result: vrai si c_2 est nouveau, faux sinon
initialisation ;
$a \longleftarrow min(c_2.extent \setminus c_1.extent)$;
$b \longleftarrow max(c_1.extent)$;
if $a > b$ **then**
    T.insert(c_2) ;
    return vraie ;
$N(c_1).cous \longleftarrow N(c_2)$;
return false ;

Dans cet algorithme $N(c)$ est le noeud qui contient le concept c , et $N(c).cous$ est un lien cousu.

La profondeur du trie T est limitée par $|E|$ et son largeur par $|F|$. Une recherche (ou insertion) s'efectue en temp $O(min(|E|, |F|))$.

Théorème 3.2 *Soit $|F|$ le nombre de comcepts dans le contexte $C(E, F, R)$.*
La complexité de l'algorithme 1 est au en temp $O(|F|.|E|^3)$

3. rappelons que ces éléments sont les attributs et qu'un ordre total est associé

3.2 Algorithme de Nourine et Raynaud

L'algorithme de Nourine et Raynaud est apparu en 1999 (Nourine and Raynaud, 1999) ; il est utilisé pour construire plusieurs type de treillis (Galois, complétion de Dedekind-MacNeille [4], treillis des idéaux [5] d'un poset, ...etc.), il suffit d'avoir un opérateur de fermeture.

C'est un algorithme semi incrémental (Kuznetsov and Obiedkov, 2002) ou la construction ce fera en deux étapes : D'abord il calcule les concepts et les stocke sur un arbre lexicographique, puis utilise cet arbre pour construire le graphe.

Dans toute la suite $\mathcal{C}(E, F, R)$ est un contexte, et $\mathcal{T}$ est la base de transaction associée.

3.2.1 Calcul des concepts

Le calcul des concepts se fera par une méthode simple (naïve) : Pour chaque transaction de base on fait des intersections entre l'ensemble des attributs et l'intention de chaque concept déjà calculé ; si le résultat est nouveau alors on calcule l'extention en ajoutant l'identificateur de la transaction à l'extention de concept utilisé ; sinon on fait une mise à jour de ce dernier de la même façon.

Les concepts sont stockés sur un arbre lexicographique (Trie) dont le parcours ce fera par l'intention (considérée comme chaîne de caractères) et les nœuds sont décoré par l'extention de chaque concept.

Transaction Id	Attribut
1	acd
2	ad
3	bcd
4	be

TABLE 3.1 – Context en Base de transaction

Exemple Le contexte représenté par la Table 3.1 contiens neuf concepts :

$$\{(1234, \emptyset), (123, d), (34, b), (12, ad), (13, cd), (4, be), (1, acd), (3, bcd), (\emptyset, abcde)\} \tag{3.2}$$

4. tout ensemble partiellement ordonné peut être complété (de plusieurs façons)pour formé un treillis complet le plus petit de ces complétions est appelé : complétion de Dedekind-MacNeille

5. Un idéal d'un ensemble ordonné $(E, \leq)$ est une partie non vide I de E telle que : 1-I est une section commençante, c'est-à-dire que tout minorant d'un élément de I appartient à I ; 2- I est un ensemble ordonné filtrant, c'est-à-dire que deux éléments quelconques de I possèdent toujours un majorant commun dans I.

. La figure 3.1 représente l'arbre associé.

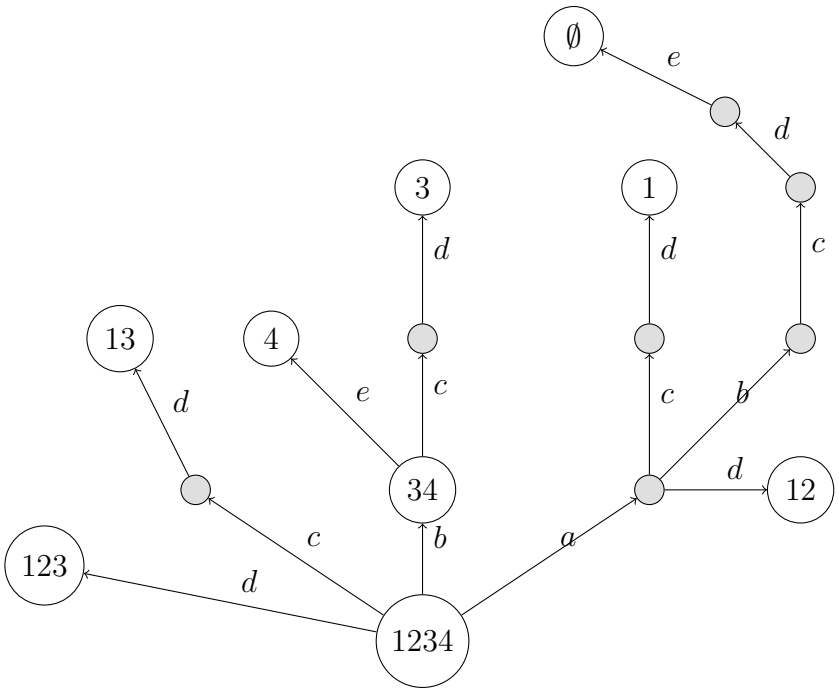

FIGURE 3.1 – Trie décoré par les extentions

L'algorithme de calcul des concepts est le suivant :

Algorithm 4: Génération des Concepts : GeneConcepts()

Data: Base de transaction $\mathcal{T}$
Result: Arbre lexicographique $\mathcal{F}$
initialisation ;
$\mathcal{F}.add((\emptyset, F))$;
for $t \in \mathcal{T}$ **do**
  for $C \in \mathcal{F}$ **do**
    $B \longleftarrow C.intent \cap t.attrib$;
    if $B \notin \mathcal{F}$ **then**
      $C'.intent \longleftarrow B$;
      $C'.extent \longleftarrow C.extent \cup t.id$;
      $\mathcal{F}.add(C')$;
    else
      $\mathcal{F}.update(B, t.id)$;

L'arbre $\mathcal{F}$ est initialisé avec le concept trivial $(\emptyset, F)$, où F est l'ensemble des attributs.

La méthode *update*() ajoute l'identificateur de la transaction t au concept dont l'intention est B qui existe déjà dans $\mathcal{F}$.

L'algorithme 4 est de complexité : $O((|F| + |\mathcal{T}|).|\mathcal{F}|)$ (Nourine and Raynaud, 1999).

3.2.2 Calcul du graphe de couverture

Soit $C(A, B)$ et $C'(A', B')$ deux concepts de $\mathcal{F}$ tel que : $C \leq C'$ dont A est l'extention et B est l'intention (rappelons que $C \leq C'$ signifie $A \subseteq A' \wedge B' \subseteq B$) On définit l'opérateur Δ comme suit :

$$\Delta(C, C') = A' \setminus A \tag{3.3}$$

Δ définit la différence entre les extentions des deux concepts i.e. l'ensemble des transactions contenant C' et ne contenant pas C.

Théorème 3.3 *$C(A, B) \prec C'(A', B')$ dans $\mathcal{F}$ si et seulement si $B \cap t_1 = B \cap t_2 \qquad \forall t_1, t_2 \in \Delta(C, C')$.*

C'est a dire l'existence d'une partie commune entre touts les transactions de $\Delta(C, C')$. La preuve de ce théorème se trouve dans l'article (Nourine and Raynaud, 1999).

Une conséquence de ce théorème est le corollaire suivant :

Corollaire 3.1 *$C(A,B) \prec C'(A',B')$ dans $\mathcal{F}$ si et seulement si B est un bloqueur minimal de $\Delta(C,C')$*

Autrement dit : $B = B' \cap t \ \forall t \in \Delta(C,C')$.

Nourine et al ont proposé la stratégie algorithmique suivante pour le calcul du graphe de couverture (Nourine and Raynaud, 1999) :

A chaque concept $C \in \mathcal{F}$ on associe une liste des successeurs immédiats.

Pour chaque concept $C(A,B)$, on calcule un sac (multi-ensemble) [6] S de tous les candidats possibles comme suit : $S = \{B \cap t \mid t \in \mathcal{T} \setminus A\}$. Pour que $C'(A',B')$ soit un successeur immédiat il faut et il suffit que B' apparaît $|\Delta(C,C')|$ fois dans S.

Algorithm 5: Graphe de couverture : CouverGraphe()

Data: Trie $\mathcal{F}$, Base $\mathcal{T}$

Result: les listes d'adjacence de chaque concept

initialisation ;

for $C \in \mathcal{F}$ **do**
  $COUNT(C) \longleftarrow 0$;

for $C \in \mathcal{F}$ **do**
  for $t \in \mathcal{T}$ **do**
    $B' \longleftarrow C.intent \cap t.attrib$;
    $COUNT(C') + +$;
    if $|C'.extent| = COUNT(C') + |C.extent|$ **then**
      $C.Immsuc.add(C')$;
  reset $COUNT$;

On associe a chaque concept un compteur COUNT initialiser à 0.

B' est l'intention d'un successeur candidat C' et sert à le trouver dans $\mathcal{F}$, le compteur de C' est ainsi incrémenté. Si C' vérifie la condition mentionnée en haut il sera ajouté au liste des successeurs de C.

La complexité temporelle de l'algorithme 5 est temp $O((|X| + |\mathcal{T}|).|\mathcal{T}|.|\mathcal{F}|)$.

6. Un multi-ensemble est un type d'ensemble qui autorise la redondance c.a.d la duplication des éléments

L'algorithme final est la suivant :

Algorithm 6: Nourine

Data: Base de transaction $\mathcal{T}$

Result: Trie des concepts, listes des successeur immédiat

initialization ;

GeneConcepts() ;

CouverGraphe() :

Théorème 3.4 *L'algorithme 6 nécessite un temp* $O((|X| + |\mathcal{T}|).|\mathcal{T}|.|\mathcal{F}|)$.

3.3 Algorithmes Incrémentaux

L'avantage majeur des algorithmes incrémentaux est le fait qu'ils suivent l'ordre naturel de construction des bases de données [7]. Les algorithmes incrémentaux répondent a la question suivante : On a un treillis $\mathcal{T}$ d'un contexte $\mathcal{C}$. Comment calculer le nouveau treillis $\mathcal{T}'$ si on ajoute un objet au contexte $\mathcal{C}$?

Dans tout la suite des sections, $\mathcal{C}$ est un contexte et $\mathcal{G}$ est le graphe (diagramme de Hasse) associé.

3.3.1 Idées de base

Avant de commencer adoptons la terminologie suivante :

L'ensemble des attributs d'un objet t est appelés description de t. Si on parle d'intersection de concepts (ou concept et objet) sa signifie l'intersection des intentions , en faite un concept est totalement définit par son intention.

L'idée générale est que les nœuds de $\mathcal{G}$ n'ont jamais disparus si on ajoute un nouveau objet.

Le nouveau graphe $\mathcal{G}'$ contient au moins toute les nœuds de l'ancien graphe $\mathcal{G}$.

Une autre idée est la proposition suivante :

Proposition 3.1 *Pour trouver les nouveaux concepts il suffit de considérer l'intersection du nouveau objet avec les concepts de l'ancien treillis.*

7. naturellement les données arrive de façon incrémentale.

La preuve est simple (Carpineto and Romano, 1996), il suffit de montrer que pour deux quelconque nouveaux concepts C_1 et C_2 la borne supérieur C_k est déjà calculée.

On fait il est claire qu'il existes (dans l'ancien treillis) deux concepts C_{11} et C_{22} tel que : $C_1 = C_{11} \cap t$ et $C_2 = C_{22} \cap t$,on a alors :

$$C_k = C_1 \cap C_2 = (C_{11} \cap t) \cap (C_{22} \cap t) = (C_{11} \cap C_{22}) \cap t$$

et comme $(C_{11} \cap C_{22})$ existe dans l'ancien treillis, le nouveau treillis contient nécessairement C_k.

Si nous prenons ces idées en considération, l'intersection de chaque concept avec le nouveau objet amène à l'un des situations possibles suivante :

L'intention de concept est un sous ensemble de la description de l'objet dans ce cas l'extention du concept est augmenté par l'identificateur de l'objet et le graphe est inchangé .

L'intention de concept et la description de l'objet sont incomparables le concept est non affecté

- Si l'intersection est vide le graphe est inchangé.
- Sinon un nouveau concept est créer .

L'intention de concept est un super ensemble de la description de l'objet le concept est non affecté mais un autre (nouveau) est créer et ajouté au graphe.

Le calcule de nouveaux concepts ne pose pas de problèmes (linéaire en nombre de concepts) [8] mais la mise à jour de graphe pour chaque nouveau concept demande beaucoup de calcule (recherche de successeurs et prédécesseurs immédiats, ajout et suppression des arcs ..) et c'est encor pire s'il y à redondance.

Dans la suite nous présentant deux approches (algorithmes) pour évité (ou réduire) la redondance.

3.3.2 Algorithme de Godin et al.

La première approche (Godin et al., 1991) utilise le cardinal de l'intention comme facteur de simplification des test, ceci pour deux raisons :

- Le fait qu' un ensemble de concepts dont l'intention à le même cardinalité forme une anti-chaîne [9] (peut être non complète).

8. voire la section précédente.
9. rappelons qu'un anti-chaîne est un ensemble d'éléments deux à deux non comparables

— Pour voir qu'un concept est présent dans un ensemble il suffit de vérifier les concepts de même cardinalité.

L'algorithme classifier l'ensemble des concepts sur des paquets OP_j tel que pour tout $C(X,Y) \in \mathcal{G}$ on à : $C \in OP_{|Y|}$.

Algorithm 7: Godin et al

Data: Treillis $\mathcal{G}$, nouveau objet t

Result: Treillis mis à jour $\mathcal{G}$

```
for C(X,Y) ∈ G do
    OP_|Y|.add(C(X,Y));
    NP_|Y| ⟵ ∅;
for j = 0 to maxcardinal do
    for C(X,Y) ∈ OP_j do
        if Y ⊆ t.attrib then
            C(X,Y) ⟵ C(X ∪ t.id, Y);
            NP_j.add(C(X,Y));
            if Y = t.attrib then
                exit;
        else
            A ⟵ Y ∩ t.attrib;
            if A ≠ Y_1 ∀C(X_1,Y_1) ∈ NP_|A| then
                C'(X ∪ t.attrib, A); /*nouveau concept */
                NP_|A|.add(C');
                Link(C,C');
                for k = 0 to |A| − 1 do
                    for C_1(X_1,Y_1) ∈ NP_k do
                        if Y_1 ⊂ A ∧ ∄C_2(X_2,Y_2) ∈ C_1.pred telque Y_2 ⊂ A then
                            Link(C',C_1);
                            if C_1 ∈ C.succ then
                                RemovLink(C,C_1);
                    if A = t.attrib then
                        exit;
```

L'algorithme parcours l'ensemble des paquets OP_j (pour old pack) de façon ascendante.

Un autre ensemble des paquets NP_j initialisé à l'ensemble vide est créé (utilisé pour évité la redondance des nouveaux concepts).

Pour chaque paquet s'il y à un concept C dont l'intention est inclus dans la description de l'objet t l'identificateur de t est ajouter à l'extention de C et ce dernier est ajouter à un nouveau paquet NP_j (pour new pack), et s'il y à égalité l'algorithme s'arrête.

Si la condition précédente n'est pas vérifier un nouveau concept C' est créé, l'algorithme cherche alors si ce dernier n'est pas présent dans $NP_{|A|}$ (A est l'intention de C'). la fonction **Link** ajoute un lien entre le concept utilisé C et le concept créé C'.

L'algorithme cherche les successeurs de C' dans les paquets de cardinalité inférieur utilisant la propriété suivante :

$$C_n \succ C' \ ssi \ C_n \geq C' \ et \ \nexists C_m \ tq \ C_n \geq C_m \geq C' \tag{3.4}$$

Tout lien entre C et un successeur C_i de C' est alors suprimé (par la fonction **RemovLink**).

Si à un moment donné l'intention d'un nouveau concept est égale au description de l'objet t l'algorithme s'arrête.

Théorème 3.5 *L'algorithme 7 nécessite $O(|\mathcal{G}|^2.|F|^2)$ pour mettre a jour le treillis $\mathcal{G}$ d'un contexte $\mathcal{C}(E, F, R)$ si on ajoute un nouveau objet dans le pire des cas.*

Une complexité quadratique on nombre de concept sachant que ce dernier augmente de façon exponentiel (Carpineto and Romano, 2004).

3.3.3 Algorithme

Pour améliorer l'algorithme précédant, Carpineto and Romano (1993, 1996) ont proposé une autre approche, utilisant la structure locale de treillis (voisinage de concept en cours de traitement dans l'ancien graphe).

L'algorithme parcourt l'ancien graphe $\mathcal{G}$ et pour chaque concept $C(X, Y)$ et calcule l'intersection A avec le nouveau objet t, plusieurs cas se présentent supposant que les successeurs de C sont $C_i(X_i, Y_i)$:

- **a :** $\exists i \ telque \ Y_i \subset A$: le concept n'est pas créé car il sera créé lors de l'examen de C_i ou l'un de ses successeurs.
- **b :** $\exists i \ telque \ Y_i = A$: le concept n'est pas créé car il existe déjà .
- **c :** $\exists i \ telque \ Y_i \supset A$: un nouveau concept est créé.

d : Y_i est incomparable avec A $\forall i$: un nouveau concept est créé.

Ainsi l'algorithme garantit qu'un concept n'est jamais créé deux fois [10].

Pour la mise à jour des liens, ou parcourt le graphe courant (contient l'anciens concepts et tous les nouveaux créés jusqu'à maintenant) et calcule les successeurs (stocker sur un ensemble S) et les prédécesseurs (stocker sur un ensemble P) utilisant la propriété 3.4.

Un lien est créé entre chaque élément de P et C, ainsi qu'entre C et chaque élément de S.

Tout lien entre un élément de P et un élément de S sera supprimé.

L'algorithme 8 est la version la plus récente (Carpineto and Romano, 2004).

Algorithm 8: Galois

Data: Treillis $\mathcal{G}$ d'un Contexte $\mathcal{C}(E, F, R)$, nouveau objet t

Result: Treillis mis à jour $\mathcal{G}$

if $\exists C_1(X_1, Y_1) \in \mathcal{G}$ tq $Y_1 = t.attrib$ **then**
  $C_1 \longleftarrow (X_1 \cup \{t.id\}, Y_1)$;
  for $C_i \in C_1.succ$ **do**
    $C_i \longleftarrow (X_i \cup \{t.id\}, Y_i)$;
  exit ;
$\mathcal{G}' \longleftarrow \mathcal{G}$;
if $f(E) \not\subset t.attrib$ **then**
  $\mathcal{G}.add(C_0(E \cup \{t.id\}, \emptyset)$;
  Link$((E, f(E), (E \cup \{t.id\}, \emptyset), \mathcal{G}))$;
$\mathcal{G}.add(C_1(t.id, t.attrib))$;
LinkConcept$((C_1(t.id, t.attrib), C_2(g(E), E), \mathcal{G}))$;
for $C(X, Y) \in \mathcal{G}'$ **do**
  if $Y \subseteq t.attrib$ **then**
    $\mathcal{G}.change(C, X \cup \{t.id\})$;
  $A \longleftarrow Y \cap t.attrib$;
  if $not(A = \emptyset \vee A = t.attrib \vee A = Y \vee (\exists C_i \in C.succ\ tq\ A \subset Y_i))$ **then**
    $\mathcal{G}.add(C'(X \cup \{t.id\}, A))$;
    LinkConcept$(C', \{C, C_1\}, \mathcal{G})$;

10. la preuve se trouve dans (Carpineto and Romano, 1996)

Conclusion

Dans le présent chapitre nous avons mentionné en détaille quelque algorithmes de constructions de treillis de Galois, ainsi que la complexité théoriques de chaque algorithme.

Dans le prochain chapitre on va essayé de faire des expérimentation sur ces algorithme ce qui permet de jugé pratiquement les ces performance.

Chapitre 4

Expérimentation

Dans ce chapitre on va parler de notre expérimentation sur l'implémentation de quelques algorithmes sur les treillis de Galois. Le langage utilisé est **JAVA** et l'environment est **ECLIPS** dans sa version **OXYGEN(3A)** sous linux (**UBUNTU 18.04 LTS**). La machine utilisée est un laptop samsung **NP350V5C** équipé d'un processeur **I5-3210M** a $2.5GHZ$ et $6G$ de **RAM**. Pour les structures de données et comme ces dernier (Trie cousu) ne sont pas pré-implémentées sur **JAVA** sont réécrites on basant sur des structure écrites par Sedgewick and Wayne (2011).

4.1 Corpus

Comme notre travail est centré sur la construction des treillis de Galois et non sur l'utilisation des ces derniers, (Apprentissage, classification). On a décidé qu'il n'est pas nécessaire d'utiliser un corpus réel et ça pour plusieurs raisons :

— La difficulté de trouver un corpus réel.
— Éviter le prétraitement des données réels.
— La facilité de construire d'un générateur de Contextes.

De toute façon ce n'est pas difficile d'appliquer notre manipulation sur des données réelles si elle existant.

4.1.1 Générateur de contexte

Un contexte est une relation (binaire) entre un ensemble d'objets et un ensemble d'attributs, il est représenté par une matrice formée que des 0 et des 1, dont chaque ligne (libellée par un identificateur d'objet) représente l'ensemble des attributs que l'objet possède, et

chaque colonne (libellée par un identificateur d'attribut) représente les objets possédant cet attribut.

Chaque objet (peut être vue comme une transaction) est identifié par un nombre et chaque attribut par une lettre.

Il nécessaire d'avoir un ordre total sur l'ensemble des attributs pour avoir un ordre lexicographique, l'utilisation de la table ASCII nous garantit sa et nous permet d'avoir des contexte allant jusqu'à 256 attributs.

L'astuce et de voir chaque ligne comme un nombre écrit en système binaire, ainsi il suffit de générer un nombre entier aléatoire et le transformer en binaire, puis on lit le nombre de gauche a droite si on trouve, alors l'attribut concerné est ajouté sinon il est rejeté.

Exemple 4.1 *Supposant que nous voulons un contexte de* 10×10 *:*

Première méthode : *Le nombre maximal qui contient* 10 *bits est* $2^{10} = 1024$*, on choisis aléatoirement* 10 *nombres dans l'intervalle* $[0, 1024]$*. Par exemple le cinquième nombre est* 785 *qui s'écrit en binaire* 1100010001 *on a alors :*

$Trans$	a	b	c	d	e	f	g	h	i	j
05	1	1	0	0	0	1	0	0	0	1

Deuxième méthode : *Dans le langage* **JAVA** *un* **long** *contient* 64 *bits et nous avons besoin de* 100 *bits ; on concatène deux nombres qui nous donne* 128 *bits, les dix premier pour la première transaction et ainsi de suite..*

La deuxième méthode est plus efficace. (les deux méthodes coïncide pour un contexte a 64 attributs).

4.2 Difficultés

La première difficulté est la définition des opérations ensembliste.

La méthode la plus efficace est l'utilisation des opérations bit à bit (bitwise) mais elle est un peut difficile pour les raisons suivants :

— limite de la taille de mot mémoire.

— bit de signe peut engendrer des anomalies

Pour cela on a utilisé les opérateurs de base prédéfinit de **JAVA** dont la complexité est acceptable (classes : **SET**, **ArrayList** ...) La deuxième difficulté est les structure de données

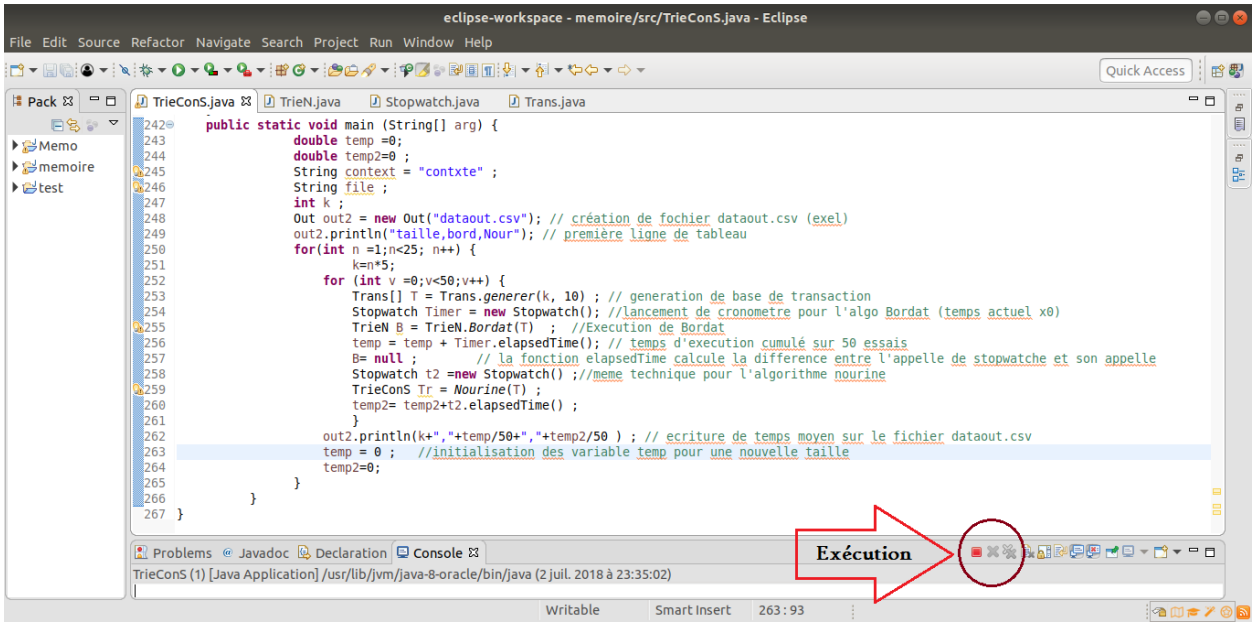

FIGURE 4.1 – Fonction main de programme d'exécution

(arbre cousu), La classe **TrieSET** écrite par Robert SEDGEWICK (le code source se trouve dans le site de l'université PRINCETON)é [1] est modifier pour avoir un (Trie cousu).

4.3 Tests

Les graphe représentés par les figures 4.2 et 4.3 sont des testes effectuer sur des contextes générés aléatoirement. Pour le premier les contextes sont carrés où on prend un moyen sur cent bases pour chaque taille.

Pour le deuxième le nombres des attributs est fixé(dix attributs par objet). La figure 4.1 montre le programme d'exécution. le calcule se fera de la manière suivante :

- création de fichier dataout.csv.
- lancement de première boucle, (itéré par taille taille), incrémenté par 5 dans chaque taille.
- lancement de deuxième boucle de 100 itérations.
- génération de contexte (base de transactions)
- calcule de temps actuel x_1 juste avant l'exécution de l'algorithme Bordat. (méthode stopWatch).

1. https ://algs4.cs.princeton.edu

— calcule de temps x_2 juste après fin d'exécution. le temps d'exécution est $x_2 - x_1$
— le temps d'exécution est cumulé sur les cent itération puis en divise par cent pour la moyen.
— écriture de résultat sur le fichier dataout.csv.
— faire la même pour l'algorithme de Nourine.

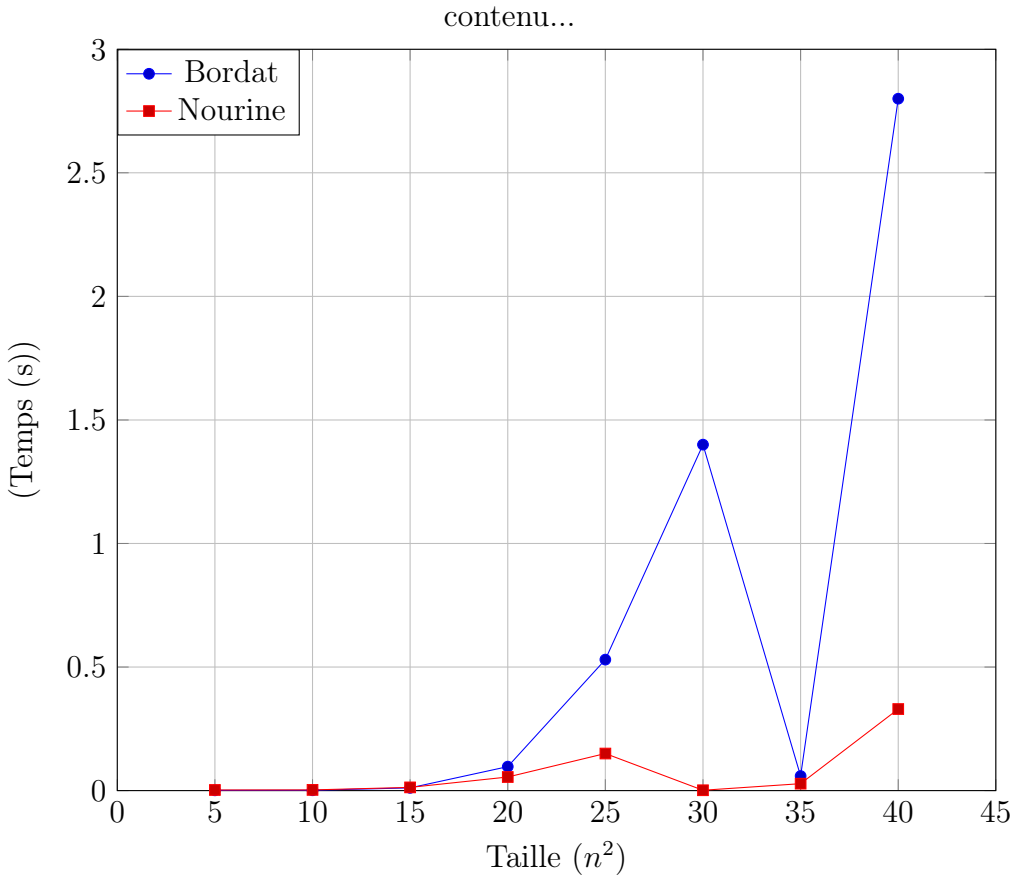

FIGURE 4.2 – Test pour un contexte carré

On peut remarquer que l'algorithme de Bordat à de bon performance pour des tailles petits. Des qu'on dépasse la taille 30×30 les testes deviens trop lent(ces tests on demandé une nuit entière). Dans son l'article Tekaya et al. (2005) ont mentionnés que l'algorithme de Nourine devient trop lent s'il est appliquer sur une bases stockée sur la disque a cause des retours.

Dans la figure 4.3 on peut voire que l'algorithme de nourine est beaucoup plus performant pour des teilles grands(linaire).

Dans ce travail, nous avons tenté de comparer, expérimentalement, quelques algorithmes

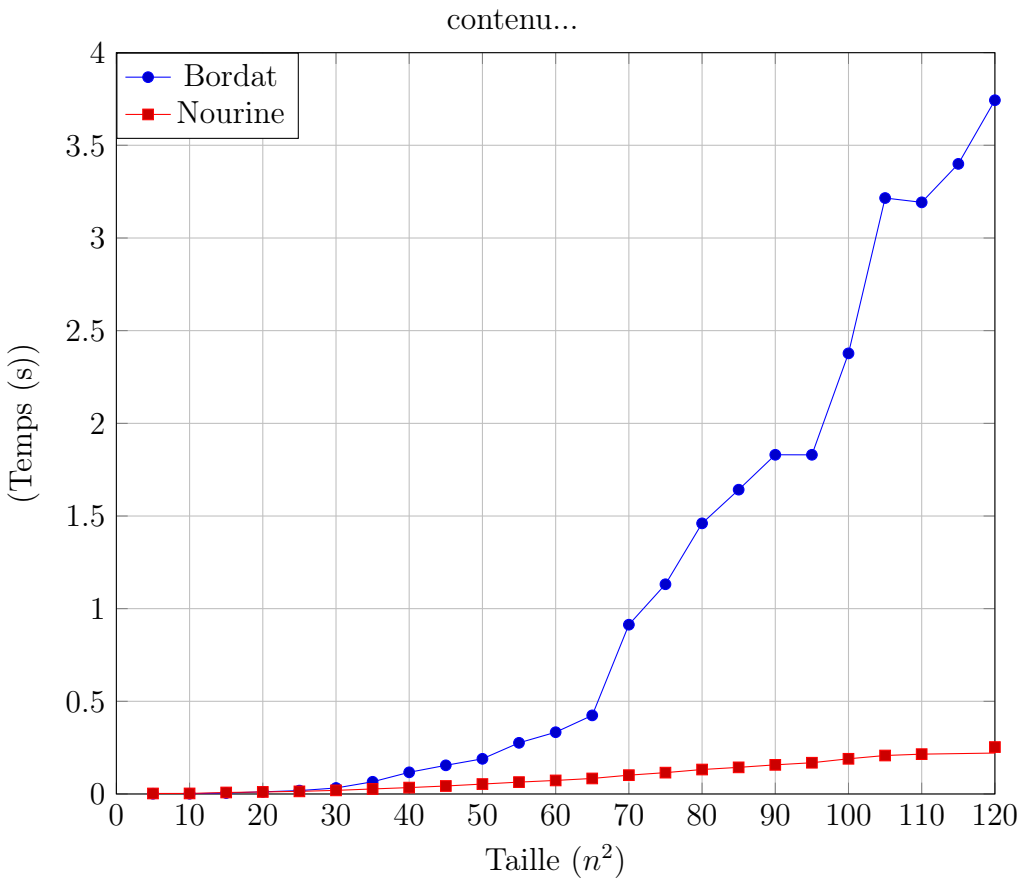

FIGURE 4.3 – Test pour un contexte à dix attributs

bien connus pour la construction des treillis de Galois. La remarque la plus importante est que la complexité théorique n'est pas un cretaire pour jugé la performance pratique d'un algorithme. mais aussi la taille est le type des données, Certains algorithme sont meilleur pour des contexte de petit taille. d'autre sont adaptés pour l'incrémentalité.

Chapitre 5

Ébauche d'unification

La recherche d'un modèle unificateur est le but principal de notre travail, mais à cause de la difficulté de traiter certains concepts que nous n'avions pas étudiés auparavant(automate à multiplicité, séries formelles) ce qui demande un peu de temps pour les comprendre de façon approfondie. Le présent ne constitue qu'une introduction.

5.1 Modélisation

Avant d'introduire notre point de vue sur un modèle unificateur il faut d'abord définir les concepts de base utilisés pour modéliser les différents acteurs de la *FCA* par les séries formelles. Pour cela on va adopter le même concept utilisé par M.Slimane Ouled Naoui dans sa thèse (à une semi-anneau pré) (Oulad-Naoui, 2018).

Soit $A = \{a_1, a_2, \ldots, a_m\}$ un alphabet de m symboles appelés *items*. Ceux-ci peuvent désignés, selon le domaine d'application, des produits achetés d'un supermarché, des pages visitées lors des surfs ou généralement une collection d'attributs ou d'événements.

Un *motif* est un sous ensemble de A. Il est appelé k-motif si son cardinal est k. Une *transaction* t_i est un ensemble non vide d'items distinguée par son unique identificateur i. Une *base de données* D est un ensemble de n transactions que nous dénotons comme un multi-ensemble $D = \{t_1, t_2, \ldots, t_n\}$.

Nous associant à chaque base D un ensemble d'identificateurs $E = \{1, 2, ..., n\}$

Dans une base de données D, nous définissons l' *extention* d'un motif x comme l'ensemble des transactions contenant x, i.e., pour lesquelles x est un sous-ensemble.

$$\text{exte}(x, D) = \{k \in E \mid x \subseteq t_k\} \tag{5.1}$$

Un motif x est *fermé* si aucun de ses sur ensemble $y \supset x$, n'à la même extention.

$$\forall y \supset x \ \text{exte}(x, D) \neq \text{exte}(y, D) \tag{5.2}$$

Sur l'ensemble des fermés on définit un ordre comme suit :

$$x < y \ sietseulementsi \ \text{exte}(x, D) \subset \text{exte}(y, D) \tag{5.3}$$

De la même façon On définit une relation de couverture :

$$x \prec y \ sietseulementsi \ x < y \ et \ \nexists \ z \ tel \ que \ x < z < y \tag{5.4}$$

5.2 Motifs fermés comme un polynôme

Un polynôme est une série formelle finie. Partant du constat que l'univers que nous modélisons est formé par des composants finis (alphabet, motifs, transactions, base de données), nous choisissons donc une modélisation reposant sur les polynômes.

Soit A un alphabet où nous imposons un ordre total [1], et soit l'application $\omega : \ \mathcal{P}(A) \longrightarrow A^*$ associée à chaque motif $x = \{a, b, \ldots\}$ le mot $\omega(x) = a_1 a_2 \ldots$ où les éléments de x sont ordonnés puis concaténés. Comme ω est injective on peut (on travaillons sur l'image de $\mathcal{P}(A)$) coder le motif x par le mot $\omega(x)$.

$$\omega(x) = a_1 \ldots a_k, \ \text{tel que} \ a_1 < \ldots < a_k \tag{5.5}$$

Le motif nul $\emptyset$ sera représenté par le mot vide ε. Nous adoptant la même notation sur $\mathcal{P}(E)$ sauf pour $\emptyset$ qui reste inchangé.

Dans ce qui suit, nous considérons des éléments du monoïde libre A^* et les coefficients associés pris dans le semi-anneau de $(\mathcal{P}(E), \cup, \cap, E, \emptyset)$(où E est un ensemble d'identificateurs). L'idée principale derrière notre formalisme est de coder un motif par un mot sur l'alphabet A et de même, tous ses sous-motifs par un polynôme sur le semi-anneau de comptage. Après avoir déduit le polynôme d'une base de données en effectuant la somme [2] de ceux correspondant aux transactions les constituant, la question est alors d'extraire de ce polynôme l'ensemble des termes où la condition (5.2) est remplie.

Il est aussi important de mentionner que puisque nous traitons essentiellement des motifs, qui sont rien d'autre que des ensembles d'items, nous suivons dans la suite une concaténation

1. Dans nos exemples, nous admettons pour simplifier l'ordre lexicographique.
2. Rappelons que somme signifie la première opération de semi-anneau utilisée, ici l'union

particulière de motifs. Cette opération doit en fait se conformer à deux conditions triviales : d'une part, l'absence de répétition d'items, et d'autre part le respect de l'ordre fixé au départ. À titre d'exemple, la concaténation de $a_i a_k$ et $a_j a_k$ génère le motif $a_i a_j a_k$ (supposons l'ordre $a_i < a_j < a_k$) et non pas $a_i a_k a_j a_k$. Plus formellement, la concaténation de deux items de a et b produit le résultat suivant :

$$ab = \begin{cases} ab & \text{si} \quad a < b \\ ba & \text{si} \quad b < a \\ a & \text{si} \quad a = b \end{cases} \tag{5.6}$$

Définition 5.1 (Polynôme de sous-séquences de motif (PSM)) *Soit $x = a_{i_1} a_{i_2} \dots a_{i_k}$ un k-motif. Le polynôme de sous-séquences de motif (PSM) $\mathbb{S}_x$ associé à x est défini comme suit(a une facteur multiplicateur prés [3]) :*

$$\mathbb{S}_x = \begin{cases} 1 & si \;\; x = \varepsilon \\ \prod\limits_{1 \leq j \leq k} (a_{i_j} + 1) = (a_{i_1} + 1)(a_{i_2} + 1) \dots (a_{i_k} + 1) & sinon \end{cases} \tag{5.7}$$

La formule (5.7) ci-dessus constitue la brique de base de notre formalisation à base de polynômes. En effet, les racines de cette formule découlent de notre encodage. Premièrement, le motif nul est codé par le terme 1ε, qui peut être simplifié selon la section 1.2.3 relatives aux préliminaires par le terme constant 1. Identiquement, un singleton $\{a\}$ est représenté, en deuxième lieu, par le terme $1a$ ou simplement a. Dans le même ordre d'idées, et afin de pister l'existence d'un item donné a dans nos polynômes, nous devons considérer soit son absence matérialisée par le motif nul ou sa présence marquée par le terme associé au singleton en question. Ainsi, nous aboutissons au polynôme $(a + 1)$. Notons au passage, que ce dernier polynôme est une généralisation de l'expressions $(a + \varepsilon)$ bien connue dans les langages rationnels pour dénoter les sous-séquences d'un mot (Oulad-Naoui, 2018).

Dans ce qui suit, nous dénotons, pour chaque $a \in A$, par $\overline{a}$ le polynôme $(a + 1)$, et par $\overline{a_{i_1} a_{i_2} \dots a_{i_k}}$ le PSM $\mathbb{S}_x$ associé au motif $x = a_{i_1} a_{i_2} \dots a_{i_k}$. Le polynôme $\mathbb{S}_x$ est donc le polynôme qui représente tous les sous motifs de x. Par exemple, nous associons au motif $x = abc$ son polynôme de sous-séquences $\mathbb{S}_x = \overline{abc} = (a + 1)(b + 1)(c + 1)$, ce qui nous donne le polynôme : $1 + a + b + c + ab + ac + bc + abc$. Notons qu'étant donné que ε est une

3. Se facteur sera justifier dans la suite

sous-séquence de tout mot, chaque polynôme doit alors inclure au moins le terme constant 1.

Le passage au polynôme de sous-séquences d'une base de données D est simple. Ce dernier sera dérivé à partir de ceux associés aux transactions qui la forment.

Définition 5.2 (Polynôme de sous-séquences de base de donnée (PSB)) *Soit $D = \{t_1, \ldots, t_n\}$ une base de n transactions. Le polynôme de sous-séquences de base de donnée (PSB) $\mathbb{S}_D$ associé à D est la somme des n polynômes de sous-séquences de ses transactions multiplies par l'identificateur de :*

$$\mathbb{S}_D = \sum_{i=1}^{n} \mathbb{S}_{t_i} \tag{5.8}$$

Remarque : D'habitude sur un semi-anneau, la première opération est appelée somme, dans notre cas somme signifie union.

Maintenant que nous avons terminé la modélisation de problème on va introduire trois approches pour la recherche des motifs fermés ainsi que l'ordre sous jacent.

5.3 Approche du totalité de la base

Avant de commencer illustrons l'idée derrière cette approche. Soit $F = \{a, b, \ldots, \}$ un ensemble d'attributs, sur l'ensemble des motifs $\mathcal{P}(F)$ on définit une relation comme suit : $x \sim y$ si et seulement si $g(x) = g(y)$ (aù (f, g) est une correspondance de Galois), la relation $\sim$ est un équivalence.

Soit $[x]$ la classe d'équivalence de x.

Proposition 5.1 *Si on adopte un orde sur $[x]$ basé sur la cardinalité croissante alors : $max([x])$ est fermé.*

Cette proposition est une conséquence des propriétés de l'opérateur de fermeture $\psi = g \circ f$.

Commençons par illustrer ces concepts à l'aide d'un exemple de référence. Prenons l'exemple de la table 2.2.

La table 5.1 montre une base de données de six transactions, où la troisième colonne donne, à travers l'équation (5.7), le PSM de chaque transaction [4](à un facteur multiplicatif

4. Rappelons qu'une transaction est un motif.

TABLE 5.1 – Base de données de transactions et les polynômes associés

i	t_i	$\mathbb{S}_{t_i}$
1	bcf	$1 + b + c + f + bc + bf + cf + bcf$
2	abcg	$1 + a + b + c + g + ab + ac + ag + bc + bg + cg + abc + abg + acg + bcg + abcg$
3	abeg	$1 + a + b + e + g + ab + ae + ag + be + bg + eg + abe + abg + aeg + beg + abeg$
4	ade	$1 + a + d + e + ad + ae + de + ade$
5	ad	$1 + a + d + ad$
6	c	$1 + c$

$$\mathbb{S}_D = \mathbf{E} + \mathbf{(2345)a} + \mathbf{(123)b} + \mathbf{(126)c} + (45)d + (34)e + (1)f + (23)g + (23)ab + (2)ac + \mathbf{(45)ad} + \mathbf{(34)ae} + (23)ag + \mathbf{(12)bc} + (3)be + (1)bf + (23)bg + (1)cf + (2)cg + (4)de + (3)eg + (2)abc + (3)abe + \mathbf{(23)abg} + (2)acg + \mathbf{(4)ade} + (3)aeg + \mathbf{(1)bcf} + (2)bcg + (3)beg + \mathbf{(2)abcg} + \mathbf{(3)abeg}$$

pré[5]). Nous avons calculé aussi dans la dernière ligne le polynôme de sous-séquences de la base toute entière via l'équation (5.8).

Pour chaque transaction le polynôme associé est multiplié par son identificateur et dans le **PSB** le facteur de chaque motif est l'union des facteurs des polynômes aù il apparaît.

Pour connaître les concepts la Proposition 5.1 est utilisée. Dans la table 5.1 les motifs fermés sont en gras. est on peut voir que tous les concepts sont calculés (à l'exception du concept trivial $(F, \emptyset)$).

Dans sa thèse M. Slimane Ouled-Naoui à proposé l'utilisation d'un automate à multiplicité préfixiel[6] **AMP**, et une méthode de construction basée sur les polynômes préfixiels.

La Figure 5.1 représente un automate à multiplicité préfixe reconnaissant les concepts de la base de l'exemple.

Cette méthode ne permet pas d'avoir tout le graphe de couverture, mais seulement une partie (l'automate ressemble à une arbre couvrant de diagramme de Hasse).

Pour trouver touts les successeurs des concepts on peut utiliser la technique des bloqueurs (Nourine and Raynaud, 1999), ou utilise la relation de couverture décrite par la formule 2.7.

5. Il s'agit de l'identificateur de chaque transaction.
6. Notre professeur a utilisé cet automate pour connaître les motifs fréquents.

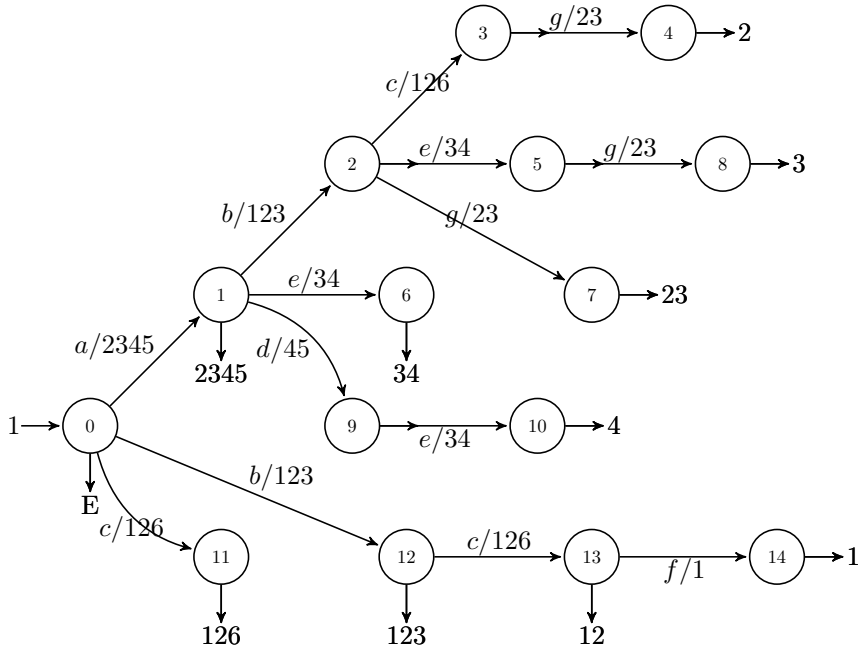

FIGURE 5.1 – Un AMP associé à l'exemple de référence

5.4 Approche incrémentale

L'avantage majeur de cette modélisation est qu'un approche incrementale est réalisable de façon très simple.

Supposons qu'on a une base dont on a calculé le polynôme des motifs fermés (et éventuellement l'automate associé). Et supposons qu'on veut ajouter une nouvelle transaction.

La technique est simple, d'abord on calcul le **PSM** de la nouvelle transaction, puis pour chaque sous motif du polynôme **PSM**, on fait une recherche sur le **PSB**. Si le sous motif est présent alors l'identificateur est ajouté au facteur de sous motif dans le **PSB**. Si le dernier sous motif (qui est la transaction toute entière) n'est pas présent, on l'ajoute au **PSB** avec comme facteur l'identificateur de la transaction. Si aucun sous motif n'est présent sauf le dernier il est ajouté. Dans touts les cas l'ensemble E (des identificateurs de transaction est mis à jour). En fin la Proposition 5.1 est utilisée pour mettre à joue les concepts.

La méthode est similaire pour l'automate, pour chaque sous motif on test si l'automate le réalise, si c'est le cas, alors sur tout le chemin on ajoute l'identificateur de la transaction a la multiplicité de chaque transition. Si le dernier sous motif n'est pas réalisable (ou si aucun sous motif n'est réalisable), des états sont ajouté dont les transitions sont étiquetées par par les attribues de la transaction en respectant le parcours préfixiel. Le polynôme **PSB** est utilisé pour connaître les nouveaux états de sortie.

TABLE 5.2 – Base sans la transaction bcf

i	t_i	$\mathbb{S}_{t_i}$
2	abcg	$1 + a + b + c + g + ab + ac + ag + bc + bg + cg + abc + abg + acg + bcg + abcg$
3	abeg	$1 + a + b + e + g + ab + ae + ag + be + bg + eg + abe + abg + aeg + beg + abeg$
4	ade	$1 + a + d + e + ad + ae + de + ade$
5	ad	$1 + a + d + ad$
6	c	$1 + c$

$$\mathbb{S}_D = \mathbf{E} + \mathbf{(2345)a} + (23)b + \mathbf{(26)c} + (45)d + (34)e + (23)g + (23)ab$$
$$+(2)ac + \mathbf{(45)ad} + \mathbf{(34)ae} + (23)ag + (2)bc + (3)be + (23)bg +$$
$$+(2)cg + (4)de + (3)eg + (2)abc + (3)abe + \mathbf{(23)abg} + (2)acg + \mathbf{(4)ade} + (3)aeg$$
$$+(2)bcg + (3)beg + \mathbf{(2)abcg} + \mathbf{(3)abeg}$$

Illustrons ça par un exemple. Prenons l'exemple précédant en supprimant la transaction bcf.

L'automate associé est representé par la Figure 5.2. On applicant la procedure mentionée, on tombe sur l'automate 5.1.

Le dernier approche est l'approche diviser pour régner est aussi réalisable soit horizontalement (division de l'ensemble d'attributs) soit verticalement (division de l'ensemble d'objets) en utilisant la composition des automates.

Depuis l'invention de l'analyse formelle de concepts, les chercheurs ont développés plusieurs algorithme sur la construction des treillis de Galois. (on peut cite entre dix et vingt algorithmes). Ce qui réduit la possibilité d'introduire de nouveaux algorithmes.

L'utilisation des séries formelles ouvre un autre voie dans qui permet d'autre possibilités. Et nous avons vu, dans cet chpitre qu'il existe plusieurs approches intéressantes.

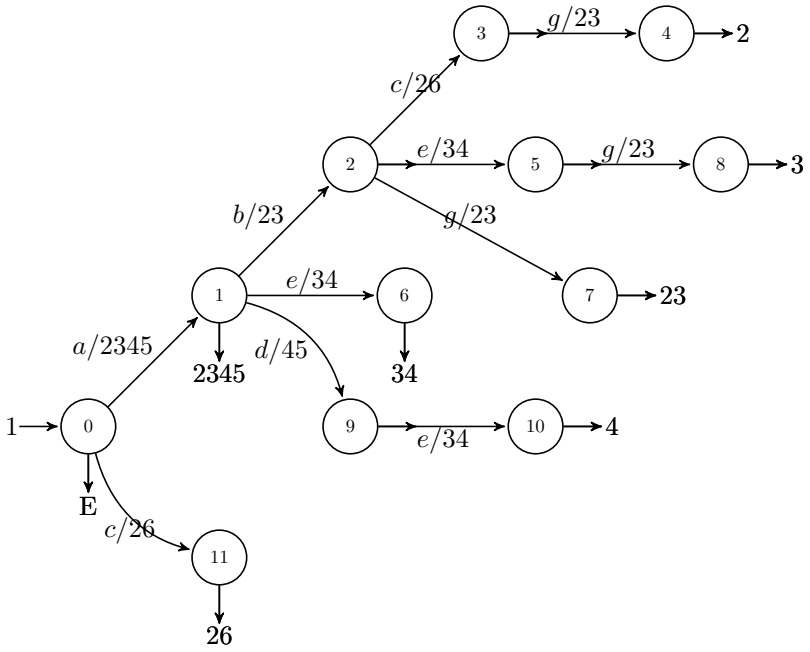

FIGURE 5.2 – Un AMP realise le polynome de la base de table 5.2

Conclusion générale

L'analyse formelle de concepts, malgré sa nouveauté, a prouvé sont utilité et sa puissance à travers les nombreux travaux et recherches. Chose que nous avons remarqué durant notre travail sur cette thèse. En plus, l'analyse formelle de concepts est beaucoup plus simple que l'analyse statistique. On peut citer plusieurs points fortes pour la **FCA** :

— facilite l'extraction d'information et des connaissances.
— stockage de données de façon plus pratique.
— rapidité de l'apprentissage automatique
— augmentation de la performance dans des tâches de l'intelligence artificielle
— ...

Les algorithmes de construction des treillis de Galois, malgré leur diversités, utilisent plusieurs approches différentes, principalement basé sur :

— Calcul des concepts uniquement.
— Prochains voisins.
— Utilisation des concepts pour construire le graphe.
— Construction partielle

La dernière approche est peut être le plus intéressante a cause de l'incrémentalité, et la technique diviser pour régner.

Nous avons remarquer que les performances des algorithme n'est pas juger par la complexité théorique uniquement, mais il existes d'autres facteurs comme la taille de base de données, l'incrémentalité et le nombre d'entré/sortie.

En fin l'introduction d'un modèle unificateur basé sur les séries formelles est les automates ouvre la voie sur la possibilité d'introduire de nouvelle approche, après l'introduction d'un modèle complet ce qui constitue une piste par des future travaux.

Annexe A

5.4.1 Parties maximales d'un Graphe bipartie

L'un des plus important algorithmes de construction de treillis de Galois est basé sur le calcul des parties maximales d'un graphe bipartie (Bordat, 1986). Nous présentons ici une technique de calcul de ces parties.

Soit $G(V, E)$ un graphe dont V est un ensemble de sommets est E un ensemble d'arêtes.

On définit une fonction successeur δ tel que pour tout $x \in V$ on a :

$$\delta(x) = \{y \in V \mid (x, y) \in E\} \tag{5.9}$$

et si $X \subseteq V$ alors :

$$\delta(X) = \{y \in V \mid \exists x \in X, (x, y) \in E\} \tag{5.10}$$

G est un graphe bipartie s'il existe une partition de V en deux ensembles A et B tel que : $\delta(A) = B$ et $\delta(B) = \emptyset$. Dans la suite si G est bipartie on le note : $G(A, B, \delta)$.

G est un graphe tripartie s'il existe une partition de V en trois ensembles A, B et C tel que : $\delta(A) = B$, $\delta(B) = C$ et $\delta(C) = \emptyset$. Si G est tripartie on le note : $G(A, B, C, \delta)$.

Parties maximales Soit $G(U, V, \delta)$ un graphe bipartie de fonction successeur δ. Les parties maximales de G sont les ensembles des successeurs qui sont maximales au sens de l'inclusion, autrement dit : Pour $x \in U$, $\delta(x)$ est maximale si et seulement si :

$$\forall y \in U, \delta(x) \not\subset \delta(y)$$

Exemple 5.1 *Dans la figure 5.3, les parties maximales sont :* $\delta(1) = \{b, c\}$ *et* $\delta(3) = \{a, b, g\}$. *La partie* $\delta(4) = \{a, g\}$ *n'est pas maximale car elle est incluse dans* $\delta(3)$, *idem pour* $\delta(5)$, *et* $\delta(6) \subset \delta(1)$.

Une technique de calcul des partie maximales est la suivante :

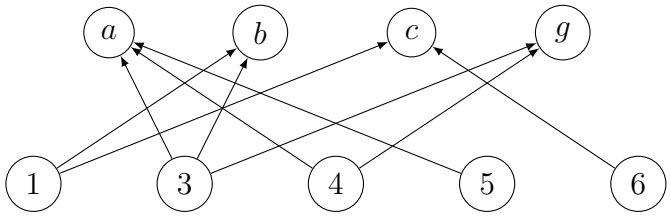

FIGURE 5.3 – Graphe bipartie

A partir de $G(U, V, \delta)$ on construit un graphe tripartie $G(U, V, W, \mu)$ où W est un ensemble en bijection avec U, chaque élément x de U a un équivalent noté x' dans W. La fonction successeur μ est définie comme suit : $\mu(x) = \delta(x)$ si $x \in U$ et pour tout $x' \in W$, $\mu^{-1}(x') = V - \mu(x)$. On a alors la propriété suivante :

$\mu(x_1) \not\subset \mu(x_2)$ si et seulement s'il existe un chemin entre x_1 etx'_2 dans $G(U, V, W, \mu)$.

Exemple 5.2 *La figure 5.4 représente un graphe tripartie déduit du graphe de la figure 5.3. On peu voir par exemple que comme* $\mu(4) \subset \mu(3)$ *il n'existe pas un chemin entre les nœuds* $3'$ *et* 4.

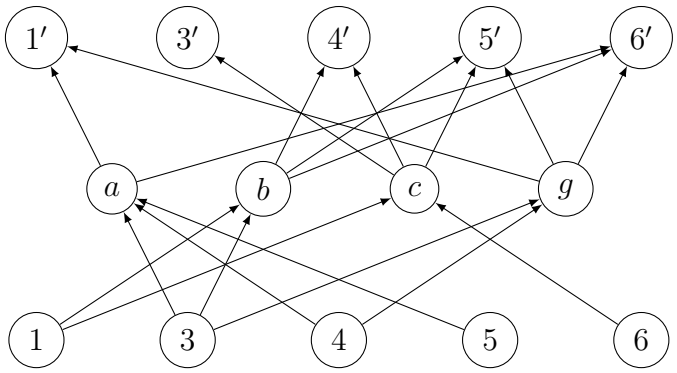

FIGURE 5.4 – Graphe tripartie

Selon Bordat, pour un graphe d'ordre n, un algorithme basé sur cette technique (construction du graphe tripartie, puis calcul des parties maximales), est de complexité $O(n^\alpha)$, avec $\alpha \simeq 2.5$, et c'est le meilleur résultat en son temps.

Bordat a proposé un autre algorithme moins performant mais plus facile à mettre en œuvre.

Soit $G(U, V, \delta)$ un graphe bipartie, avec $|U| = p$. Est supposons que les nœuds $a_k \in U$

sont triés de la façon suivante : $|\delta(a_i)| \leq |\delta(a_j)|$ si $i < j$.

Algorithm 9: Parties maximales

Data: Graphe bipartie $G(U, V, \delta)$

Result: Parties maximales de G stockée sur C

```
k ⟵ 1 ;
for 1 ≤ i ≤ p do
    max(a_i) ⟵ true ;
for 1 ≤ i ≤ p do
    if max(a_i) then
        C_k ⟵ {a_i} ;
        for i + 1 ≤ j ≤ p do
            if δ(a_j) ⊆ δ(a_i) then
                max(a_j) ⟵ false ;
            if δ(a_j) = δ(a_i) then
                C_k ⟵ C_k ∪ {a_j} ;
        k + + ;
```

Dans l'algorithme 9, à chaque nœud a_i est associer un booléen $\max(a_i)$. La dernière condition assure que l'ensemble C contient l'image réciproque d'une partie maximale.

La complexité de cette algorithme est de $O(n^3)$ au pire des cas. La performance augmente si le nombre des parties maximales est petit (Bordat, 1986).

Bibliographie

Arnauld, A., Nicole, P., Brekle, H. E., and Löringhoff, B. F. (1967). *L'art de penser : la logique de Port-Royal*, volume 2. Frommann.

Barbut, M. (1970). *Ordre et classification*. Hachette.

Bertet, K. (2010). *Structure de treillis : contributions structurelles et algorithmiques : quelques usages pour des données images*. PhD thesis.

Birkhoff, G. (1940). *Lattice theory*, volume 25. American Mathematical Soc.

Birkhoff, G. (1967). Lattice theory.

Bordat, J. P. (1986). Calcul pratique du treillis de galois d'une correspondance. *Mathématiques et Sciences Humaines*, 96 :31–47.

Carpineto, C. and Romano, G. (1993). Galois : An order-theoretic approach to conceptual clustering. In *Proceedings of ICML*, volume 293, pages 33–40.

Carpineto, C. and Romano, G. (1996). A lattice conceptual clustering system and its application to browsing retrieval. *Machine learning*, 24(2) :95–122.

Carpineto, C. and Romano, G. (2004). *Concept Data Analysis : Theory and Applications*. John Wiley & Sons.

Carpineto, C. and Romano, G. (2005). Using concept lattices for text retrieval and mining. In *Formal Concept Analysis*, pages 161–179. Springer.

Caspard, N., Leclerc, B., Monjardet, B., et al. (2007). *Ensembles ordonnés finis : concepts, résultats et usages*, volume 60. Springer.

Caspard, N. and Monjardet, B. (2003). The lattices of closure systems, closure operators, and implicational systems on a finite set : a survey. *Discrete Applied Mathematics*, 127(2) :241–269.

Chein, M. (1969). Algorithme de recherche des sous-matrices premières d'une matrice. *Bulletin mathématique de la Société des Sciences Mathématiques de la République Socialiste de Roumanie*, pages 21–25.

Cherif, C. L., Chevallett, J., Elloumi, S., and Jaoua, A. (2003). Une extension de la connexion de galois floue pour la recherche d'information. *Information interaction intelligence*, 3(2) :73–116.

Endres, D., Adam, R., Giese, M. A., and Noppeney, U. (2012). Understanding the semantic structure of human fmri brain recordings with formal concept analysis. In *International Conference on Formal Concept Analysis*, pages 96–111. Springer.

Frécon, L. (2002). *Éléments de mathématiques discrètes*. PPUR presses polytechniques.

Fu, H. and Nguifo, E. M. (2004). A parallel algorithm to generate formal concepts for large data. In *International Conference on Formal Concept Analysis*, pages 394–401. Springer.

Gammoudi, M. M. and Nafkha, I. (2002). A formal method for inheritance graph hierarchy construction. *Information Sciences*, 140(3-4) :295–317.

Ganter, B. (1984). Two basic algorithms in concept analysis, 831 (1984).

Ganter, B. and Reuter, K. (1991). Finding all closed sets : A general approach. *Order*, 8(3) :283–290.

Ganter, B. and Wille, R. (1999). *Formal Concept Analysis : Mathematical Foundations*. Springer, Berlin/Heidelberg.

Godin, R., Mineau, G., and Missaoui, R. (1995). Incremental structuring of knowledge bases. In *Proc. of KRUSE*, volume 95, pages 179–193. Citeseer.

Godin, R., Missaoui, R., and Alaoui, H. (1991). Learning algorithms using a galois lattice structure. In *Tools for Artificial Intelligence, 1991. TAI'91., Third International Conference on*, pages 22–29. IEEE.

Kaytoue, M., Kuznetsov, S. O., Napoli, A., and Duplessis, S. (2011). Mining gene expression data with pattern structures in formal concept analysis. *Information Sciences*, 181(10) :1989–2001.

Kohler-Koch, B. and Vogt, F. (2000). Normen-und regelgeleitete internationale kooperationen—formale begriffsanalyse in der politikwissenschaft. In *Begriffliche Wissensverarbeitung*, pages 325–340. Springer.

Kuznetsov, S. O. (1993). A fast algorithm for computing all intersections of objects from an arbitrary semilattice. *Nauchno-Tekhnicheskaya Informatsiya Seriya 2-Informatsionnye Protsessy i Sistemy*, 10(1) :17–20.

Kuznetsov, S. O. and Obiedkov, S. A. (2002). Comparing performance of algorithms for generating concept lattices. *Journal of Experimental & Theoretical Artificial Intelligence*, 14(2-3) :189–216.

Messaï, N. (2009). *Analyse de concepts formels guidée par des connaissances de domaine : Application à la découverte de ressources génomiques sur le Web*. Theses, Université Henri Poincaré - Nancy 1.

Mimouni, N., Nazarenko, A., and Salotti, S. (2015). A conceptual approach for relational ir : application to legal collections. In *International Conference on Formal Concept Analysis*, pages 303–318. Springer.

Monjardet, B. (2008). La construction des notions d'ordre et de treillis. *Journées nationales de l'Association des Professeurs de Mathématiques de l'Enseignement Public (APMEP)*.

Motameny, S., Versmold, B., and Schmutzler, R. (2008). Formal concept analysis for the identification of combinatorial biomarkers in breast cancer. In *International Conference on Formal Concept Analysis*, pages 229–240. Springer.

Njiwoua, E. M. N. (2005). Treillis de concepts et classification super-visée. *Technique et Science Informatiques*, 24(4) :449–488.

Norris, E. M. (1978). An algorithm for computing the maximal rectangles in a binary relation. *Revue Roumaine de Mathématiques Pures et Appliquées*, 23(2) :243–250.

Nourine, L. and Raynaud, O. (1999). A fast algorithm for building lattices. *Information Processing Letters*, 71(5) :199 – 204.

Ore, O. (1944). Galois connexions. *Transactions of the American Mathematical Society*, 55(3) :493–513.

Oulad-Naoui, S. (2018). *Fouille de motifs : formalisation et unification*. PhD thesis, Université Amar Telidji-Laghouat.

Pfaltz, J. L. and Taylor, C. M. (2002). Scientific knowledge discovery through iterative transformation of concept lattices. In *SIAM Workshop on Discrete Math. and Data Mining, Arlington, VA, USA*.

Priss, U. (2006). Formal concept analysis in information science. *Arist*, 40(1) :521–543.

Priss, U. and Old, L. J. (2004). Modelling lexical databases with formal concept analysis. *J. UCS*, 10(8) :967–984.

Sedgewick, R. and Wayne, K. (2011). *Algorithms*. Addison-Wesley Professional.

Stumme, G. and Maedche, A. (2001). Fca-merge : Bottom-up merging of ontologies. In *IJCAI*, volume 1, pages 225–230.

Szathmary, L. and Napoli, A. (2004). Knowledge organisation and information retrieval using galois lattices. In *Workshop on Knowledge Management and Organizational Memories-ECAI 2004 (16th European Conference on Artificial Intelligence)*, pages 73–78.

Tekaya, S. B., Yahia, S. B., and Slimani, Y. (2005). Algorithme de construction d'un treillis des concepts formels et de détermination des générateurs minimaux. *Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées*, 3 :171–193.

Valtchev, P. and Missaoui, R. (2001). Building concept (galois) lattices from parts : generalizing the incremental methods. In *International Conference on Conceptual Structures*, pages 290–303. Springer.

Wille, R. (1982). Restructuring lattice theory : an approach based on hierarchies of concepts. In *Ordered sets*, pages 445–470. Springer.