UNIVERSITÉ DU QUÉBEC À MONTRÉAL

GÉNÉRATION DES RÈGLES D'ASSOCIATION: TREILLIS DE CONCEPTS DENSES

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR
ALAIN BOULANGER

JUIN 2009

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

ABRÉVIATIONS

Abréviation	Description
AFC	Analyse formelle de concepts
BDB	Base(s) de données binaires
BDI	Base(s) de données inductives
EPO	Ensemble partiellement ordonné
ICF	Iceberg de concepts formels (fréquents)
IGMFr	Iceberg de générateurs minimaux fréquents
IIFeFr	Iceberg d'itemsets fermés et fréquents
IIFr	Iceberg d'itemsets fréquents
OLAP	On-line analytical processing
ROP	Relation d'ordre partiel
SGBD	Système de gestion de base de données
SGBDI	Système de gestion de base de données inductive
SGBDR	Système de gestion de base de données relationnelle
SGBDR/O	Système de gestion de base de données relationnelle/objet
ssi	Si et seulement si
TCD	Treillis de concepts denses
TCF	Treillis de concepts formels
TGM	Treillis de générateurs minimaux
TI	Treillis d'itemsets
TIFe	Treillis d'itemsets fermés
TIFeFr	Treillis d'itemsets fermés fréquents
TIFr	Treillis d'itemsets fréquents
t.q.	Tel que

LISTE DES SYMBOLES

Symbole	Description
$\wp$	Ensemble de tous les sous-ensembles d'un ensemble (de l'anglais powerset)
$\perp$	Base d'un treillis (ou en anglais bottom)
$\top$	Sommet d'un treillis (ou en anglais top)

RÉSUMÉ

La fouille de données est l'extraction non triviale d'informations implicites, inconnues et utiles à partir des données (Piatetsky-Shapiro & Frawley, 1991). Plus récemment, la notion de systèmes de gestion de base de données inductive (SGBDI) propose l'union de la base de données traditionnelle à la fouille de données et d'une base de motifs ou patrons de données. Ces derniers sont les agents fondamentaux dans les SGBDI. Dans ce mémoire le motif examiné est le concept formel. Cependant, pour un ensemble de données (nommé contexte formel dans l'AFC) de grande taille où les données sont fortement corrélées, l'extraction peut être difficile à cause des problèmes de combinatoire inhérente à cette structure. Il est vrai que l'extraction de la collection des concepts formels fréquents, donc un iceberg plutôt qu'un treillis, est une solution. Cependant, d'une part, pour un seuil de fréquence trop faible, l'extraction des concepts formels fréquents peut demeurer difficile et la combinatoire de l'extraction demeure. D'autre part, les utilisateurs pourraient préférer conserver le treillis mais appliquer une certaine relaxation sur le formalisme des concepts formels. Cette relaxation se ferait en acceptant des exceptions dans les concepts dont les seuils sur les exceptions seraient choisis par l'utilisateur. En dernier lieu, le contexte formel pourrait bien avoir des erreurs dans ses transactions. Ces erreurs pourraient donc être la cause du nombre indu de concepts formels extraits. Une relaxation au niveau de l'extraction des concepts formels pourrait être une solution à ce problème.

Notre contribution se situe au niveau d'un motif en particulier et de son mode d'extraction. Ce mémoire propose donc un concept tolérant des exceptions bornées par des seuils, soit les concepts denses et explore la possibilité d'extraire un tel motif par l'algorithme incrémental par cardinalité. En dépit du fait que le concept ne soit plus formel mais tolérant des exceptions, les principales notions de l'analyse formelle de concepts, (e.g. la relation de précédence, le treillis) sont fortement désirées.

Mots-Clés : concepts formels, concepts denses, treillis de concepts formels, analyse formelle de concepts, concepts tolérant des exceptions, algorithme d'extraction de concepts, représentation condensée.

CHAPITRE I
INTRODUCTION

1.1 Historique et situation

Depuis bien des années, les systèmes transactionnels ont amassé des gigaoctets, des téraoctets, des pétaoctets de données. L'utilité de ces données dans le cadre des systèmes transactionnels est sans conteste. Mais qu'advient-il de ces données une fois les transactions traitées ? Dans plusieurs secteurs d'activités commerciales notamment le domaine bancaire, on ne peut se débarrasser de ces données qui sont protégées par certaines législations. Dans la plupart de ces types d'entreprises, ces données sont archivées. Peut-on trouver à ces données dormantes une nouvelle utilité ? Outre leur information transactionnelle, est-on capable de leur trouver une information complémentaire quelconque ?

Vers la fin des années 1980, les systèmes experts, issus des recherches en intelligence artificielle, ont vu le jour. Certains de ces systèmes utilisaient des données archivées dans le but de comprendre certains évènements ou corrélations. Avec l'arrivée de l'informatique décisionnelle, de nouveaux outils, principalement visuels, ont été créés. De ce nombre on trouve les tableaux de bord, les systèmes d'aide à la décision et des outils statistiques permettant d'analyser les données des systèmes actuels tout comme les données archivées[1].

[1] Vers le début des années 1990, l'éditeur de logiciels SAS était l'un des premiers grands éditeurs à offrir des outils logiciels à ses clients dont le but était l'utilisation de données récentes et/ou anciennes. Ces données étaient colligées par voie d'outils statistiques et la présentation des résultats étaient offertes sous forme de graphes. SAS était sans aucun doute un précurseur dans ce qui allait être connu sous le nom de d'outil d'aide à la décision ou de l'intelligence d'affaire. Les autres grands éditeurs de logiciel se sont empressés de sauter dans le train déjà en marche.

Vint ensuite, le développement fulgurant du matériel et par la suite des logiciels[2]. Ces derniers prenant avantage de la parallélisation des processus, de la vitesse des réseaux de fibres optiques locaux, métropolitains et étendus, des données de toutes parts vinrent inonder les disques ultra-rapides stockés dans des parcs de disques[3]. Les coûts toujours moins onéreux et les outils technologiques de toutes sortes ont rendu l'informatique omniprésente. Le matériel et logiciel n'étaient plus confinés aux applications d'affaires et de gestion des industries et des divers ministères gouvernementaux et sociétés d'état. La médecine, la recherche scientifique en biologie, chimie, physique, génétique, l'éducation, les médias et les arts ont été servis également. Cependant les outils des années 1980 et du début des années 1990 avaient vécu mais ne pouvaient survivre à cette (r)évolution. De nouvelles architectures technologiques ont vu le jour notamment, les entrepôts de données. De nouveaux outils ont été conçus, entre autres, les outils d'analyse des données[4]. L'un des modèles ou schémas les plus exploités dans ces outils est surement le cube de données. Il n'en demeure pas moins que ces outils produisent largement des résultats de nature descriptive. . Il est à noter cependant que ce type d'architecture ne répond pas à tous les besoins. La recherche documentaire en est un exemple. Dans certains domaines tels que les recherches scientifiques, on demande des analyses plus fines, plus sophistiquées. Cette sophistication trouve son écho dans ce que l'on nomme la fouille de données. La définition suivante, tirée de (Godin, 2006) l'explique.

Définition 1.1 – Fouille de données

La fouille de données, ou encore le forage, la prospection, l'exploration de données, aussi appelée découverte de connaissance dans les bases de données, est l'extraction non triviale d'informations implicites, inconnues et utiles à partir des données (Piatetsky-Shapiro & Frawley, 1991). Elle s'appuie sur des méthodes d'extraction sophistiquées à la croisée des domaines de l'apprentissage machine et des statistiques.

---

[2] Et également la mondialisation des marchés. Nous pourrions ajouter la mondialisation des connaissances avec l'internet.

[3] De l'anglais : disk farms.

[4] Outil d'analyse de données, de l'anglais : On-line analytical processing (OLAP).

1.2 Base de données inductive

Dans (Imielinski & Mannila, 1996) les auteurs proposent la notion de bases de données inductives. Une tentative de définition des concepts des bases de données inductives (ci après DBI) est présentée ci-dessous.

Définition 1.2 – Bases de données inductives

Une base de données inductive (BDI) réunie la base de données traditionnelle ainsi qu'une base de motifs (ou patrons), ceux-ci étant des agents fondamentaux des BDI. Un système de gestion de base de données inductive (SGBDI) possèderait un langage de requête permettant la liaison entre la base de données traditionnelle et la base de motifs (De Raedt, 2003). Ce langage pourrait allier le SQL standard et d'autres énoncés garantissant la propriété de fermeture et intégrant les contraintes utilisées dans les motifs de données. L'un des buts d'un SGBDI est de gérer et synthétiser les connaissances dans les BDI.

Cette définition est une tentative et il est notoire que le conditionnel fut employé à certains endroits dans notre définition. Selon (Imielinski & Mannila, 1996), les BDI et SGBDI sont l'évolution naturelle de nos SGBD modernes mais on n'en est pas encore à même de trouver sur les tablettes un produit commercial

L'un des problèmes auquel font face le domaine de la fouille de données et des bases de données inductives est la multiplicité des motifs. Certains de ces motifs ont des contraintes particulières, d'autres utilisent les mêmes contraintes tout au moins par définition mais le modus operandi peut être différent. D'autre part, aucun de ces motifs n'a de propriétés universelles et donc seraient à même de servir le plus grand nombre dans la fouille de données et la recherche de connaissances.

Prenons par exemple l'itemset, l'un des motifs les plus connus et étudiés. C'est un motif unidimensionnel formé d'éléments que l'on nomme items ou attributs. Ce dernier est

particulièrement adapté aux études de marketing entre autres, le panier à provisions du consommateur. En effet, l'ordre des lignes du ticket d'achat de provisions à une date donnée n'a aucune importance. Ce qui importe c'est le mix de produits achetés. Règle générale, une collection d'itemsets sert d'étape intermédiaire à la prospection des règles d'association. Ces dernières sont un type de motif raffinant la recherche de connaissances sur les itemsets car elles permettent, entre autres, de trouver quels sont les produits phares incitant à l'achat d'autres produits. L'extraction de la collection des tous les itemsets est cependant difficile du à la complexité temporelle des algorithmes et à la complexité spatiale. Il est à noter que d'autres domaines exploitent les itemsets (et les règles d'association) notamment dans la découverte de connaissances dans les listes de recensement, la journalisation sur le web et sur les systèmes informatiques (Boulicaut, Bykowski, & Rigotti, 2003).

Les itemsets ne sont de peu d'aide pour des domaines tels que les systèmes de télécommunications et de télédiffusions ou, dans le domaine de la biologie moléculaire et de la génétique où l'on étudie le séquençage de l'ADN, les itemsets nous sont de peu d'aide si l'ordre de séquençage est important. Puisque cet ordre est important, on peut utiliser comme patrons les épisodes ainsi que les épisodes fréquents. De ces deux motifs, on peut extraire les règles épisodiques. L'épisode est unidimensionnel comme l'itemset et a comme éléments de base l'événement et l'horodatage. La morphologie de l'épisode se présente sous trois formes soient, séquentielle, parallèle ou arbitraire[5] (Mannila, Toivonen, & Verkamo, 1995).

Un motif qui prend en compte les deux dimensions d'une table ou base de données binaire est le concept formel. Ce dernier est un bi-ensemble qui contient un ensemble d'objets appelé extension et un ensemble d'attributs ou d'items appelé intension. La particularité de ce motif est que la recherche de connaissances est possible sur les deux dimensions. Dans l'exemple de la recherche documentaire, ce motif est d'une grande utilité. L'exploration des connaissances sur le web est un exemple typique comme l'illustre le logiciel CREDO (Carpineto & Romano, 2004). Des domaines comme la gestion des services hospitaliers (Jay, Kohler, & Napoli, 2008) et la biologie (Besson, 2005) s'intéressent de plus

[5] Un mélange d'épisodes séquentiels et parallèles.

en plus aux concepts formels. Un autre avantage du concept formel est que l'intension est aussi un itemset fermé et que la cardinalité de l'extension produit le support de l'itemset. Tout comme la collection des itemsets, une collection de concepts formels peut être difficile à calculer. Une représentation condensée de la collection des concepts formels est la collection des concepts formels fréquents. Mentionnons également que nous assistons depuis peu à la naissance des concepts denses aussi appelés des bi-ensembles denses (et pertinents).

Tous ces motifs supportent des contraintes qu'elles soient monotones, anti-monotones, syntaxiques ou autres. Mais ces contraintes ne sont pas toujours appliquées ou implémentées de la même façon. Par exemple, l'extraction des concepts formels utilisant la représentation horizontale du contexte formel est guidée par les objets; la contrainte utilisant la fréquence est monotone. L'élagage ne pourra se faire qu'à la fin de l'extraction. Par contre, en se servant l'extraction incrémentale sur la représentation verticale du contexte formel, la contrainte de fréquence est anti-monotone et l'élagage est assuré durant l'extraction. Ceci est également vrai pour l'extraction des itemsets. Comme on peut le voir le modus operandi des contraintes peut dépendre du type de motifs utilisé, du type d'extraction et possiblement d'autres facteurs.

Tous ces exemples illustrent le difficile parcours qu'il reste à franchir afin de proposer un SGBDI qui satisfait la plupart des motifs et leurs particularités, de leurs contraintes, intégrés dans des langages de requêtes, proposant des interfaces usagers faciles d'utilisation et qui promeut l'exploration des données tout en livrant des résultats exacts ou approximatifs selon le choix de l'utilisateur dans un délai raisonnable.

1.3 Notre contribution

Dans des contextes formels de grande taille, denses et où les données sont fortement corrélées, il peut être difficile d'extraire les concepts formels dû à son aspect combinatoire. On peut alors choisir de n'extraire que les concepts formels fréquents. Là encore, ceci peut servir les besoins d'un domaine ou d'une communauté mais pas tous les domaines ou

communautés. D'autre part, dans certains cas la granularité du concept formel peut être trop fine et ne tient pas compte du fait que le contexte formel, comme toute base de données transactionnelle, peut contenir des erreurs. Dès lors certaines communautés, particulièrement en biologie moléculaire[6], sont plus intéressées à la "régularité" d'un motif, donc qui accepterait certaines formes d'exceptions dans les concepts formels. Par définition ces concepts ne sont plus formels, nous les qualifierons de concepts denses. C'est à dire que des exceptions sont en nombre limité et cette limitation est guidée par des seuils de tolérance à l'erreur définis par les usagers.

À ce jour très peu de travaux de recherche ont été effectués sur le domaine des concepts "formels" tolérant des erreurs. Certains de ces travaux se concentrent sur l'acceptation des erreurs sur une seule dimension du concept[7]. D'autres recherches focalisent sur la réduction du nombre de concepts en les fusionnant selon certaines heuristiques[8]. Une piste prometteuse est la recherche sur les DRBS (bi-ensembles denses et pertinents) présentés dans (Besson, Robardet, & Boulicaut, 2006). L'inconvénient est que si des usagers sont intéressés uniquement à la collection de concepts denses, sans égard à la pertinence ou que celle-ci soit faible, l'extraction de ceux-ci par l'algorithme DR-MINER proposé dans (Besson, Robardet, & Boulicaut, 2006) est difficile, voire quasi-impossible dans des contextes formels denses.

D'autre part, l'algorithme incrémental par cardinalité proposé dans (Godin, Missaoui, & Alaoui, 1995) permet d'extraire les concepts formels dans des temps très acceptables. Bien que sa complexité temporelle théorique de $O(|C|^2\ |A|^2)$, en pratique l'exécution de l'algorithme est quasi-linéaire pour beaucoup de contextes formels. Dès lors, pourquoi ne pas tenter d'implémenter le principe des concepts denses avec ce type d'algorithme. (Rouane,

---

[6] On peut également penser aux domaines de la physique nucléaire, de la mécanique quantique, des systèmes de pronostiques et diagnostiques en médecine, des systèmes tutoriels intelligents.

[7] En acceptant souvent la possibilité de perdre l'intégrité du treillis de concepts formels ou denses.

[8] Même commentaire qu'en 7.

Nehme, Valtchev, & Godin, 2004) ont présenté l'algorithme Magalice qui s'inspire des travaux de (Godin, Missaoui, & Alaoui, 1995) . Vint ensuite MagaliceA (Nehmé, Valtchev, Rouane, & Godin, 2005) qui calcule le treillis de concepts formels en utilisant la représentation verticale des données. Ce mémoire a retenu MagaliceA comme point de départ de nos recherches.

Parmi les problèmes que l'on peut rencontrer dans cette entreprise est que toutes nos lectures sur le sujet des concepts denses pointent sur des algorithmes de type traitement en lot. MagaliceA, étant un algorithme incrémental, construit ces concepts formels au fur et à mesure et on est assuré que ces concepts formels sont complets[9] et corrects uniquement à la fin de l'extraction. Il en sera de même pour les concepts denses. Mais quels sont les principes ou les heuristiques qui guideront l'inclusion d'exceptions pour des concepts "pas tous à fait finis" lors d'une itération?

Notre contribution, à ce stade, est purement exploratoire. Elle vise à vérifier si l'extraction des concepts denses peut se faire en utilisant un algorithme incrémental. Cette collection de concepts devrait suivre les principes de l'analyse formelle de concepts (AFC) et les liens de précédence de son treillis doivent être générés comme c'est le cas dans MagaliceA. La qualité de ces concepts denses et leur exactitude en fonction des seuils de tolérance choisis par les usagers doivent être vérifiées. En prime, peut-on dire, nous examinerons quel sera l'impact de ce type d'extraction sur la génération des générateurs minimaux produit par un tel algorithme.

La structure de ce mémoire va comme suit. Le chapitre 2 présente les itemsets et l'algorithme APRIORI. On notera dans ce chapitre que l'extraction de la collection complète des itemsets est difficile à cause de la nature exponentielle du problème. L'une des solutions est de n'extraire qu'un sous-ensemble de cette collection soit la collection des itemsets fréquents. Le chapitre 3 examine le formalisme ingénieux des représentations condensées ε-adéquates. En effet, la collection des itemsets fréquents est une représentation condensée ε-

[9] Dans le sens de complétude.

adéquate de la collection complète des itemsets et de son treillis. Il en va de même pour la collection des itemsets fermés et de la collection des concepts formels. Nous devrons donc examiner, à la lumière de ce formalisme, si notre collection de concepts denses sera une représentation condensée ε-adéquate ou non. Le chapitre 4 donne les bases en mathématiques discrètes des treillis en général. Le chapitre 5 focalise sur la collection des concepts formels et de son treillis. L'iceberg des concepts formels fréquents sera également présenté. On examine les principes de l'algorithme incrémental par cardinalité et un pseudo-code de haut niveau de MagaliceA est présenté. Le chapitre 6 présente la caractérisation des concepts formels sous extraction et les motifs nommés générateurs minimaux. Le chapitre 7 trace l'état de l'art à ce jour pour ce qui est de la condensation sur les deux dimensions du concept formel. On verra dans ce chapitre que la recherche est toute naissante. Le chapitre 8 présente notre algorithme, MAD-G qui tient pour **M**agalice**A** extrayant les concepts ***Denses*** employant une méthode ***Gloutonne***. Le chapitre 9 présente les expérimentations, les résultats et une discussion sur ces résultats, la performance de MAD-G versus DR-MINER, la qualité des concepts denses, les impacts sur la caractérisation des concepts sous extraction, les générateurs et enfin si la collection des concepts denses est bel et bien une représentation condensée ε-adéquate. Le chapitre 10 conclut ce mémoire.

CHAPITRE II
ITEMSETS ET RÈGLES D'ASSOCIATION

Ce chapitre considère deux motifs importants, celui des itemsets et des règles d'association. La section 2.1 débute par des définitions d'usage sur la base de données binaires et les itemsets. La section 2.2 aborde les fonctions d'évaluation nommément le support et la fréquence et, introduit les itemsets fréquents. La section 2.3 présente l'algorithme APRIORI qui extrait les itemsets fréquents. La section 2.4 aborde les règles d'association, sa fonction d'évaluation la plus connue (la confiance) et présente un algorithme d'extraction des règles d'association. Une discussion sur deux problèmes relatifs aux collections des itemsets termine ce chapitre.

Il est à noter qu'au sujet des fonctions d'évaluation, il n'existe pas de meilleure(s) fonction(s). Les fonctions de support, de fréquence et de la confiance sont présentées dans ce chapitre parce qu'elles sont très connues et utilisées. Cependant, il existe d'autres fonctions d'évaluation et il incombe, habituellement, à l'expert du domaine de choisir celles qui lui sont les plus appropriées.

Tant pour les définitions et que pour les exemples, ce chapitre est fortement inspiré de (Godin, 2006) chapitre 19.

2.1 Itemsets et autres éléments fondamentaux

Dans cette section, nous présentons les bases de données binaires, les multisets et les itemsets.

Définition 2. 1– Base de données binaires

Soit R, un ensemble de symboles appelés items. Une transaction est un sous-ensemble de R. Une base de données binaire r sur R est un multiset de transactions.

Définition 2. 2 - Multiset

Le terme multiset est utilisé lorsqu'il est possible de trouver plus d'une transaction ayant les mêmes valeurs.

Exemple 2.1

Soient R un ensemble d'items tel que présenté à la figure 2.1(a) et r une base de données binaire[10] sur R tel que présenté à la figure 2.1(b) (Godin, 2006)[11]. Il a été dit dans les définitions ci-dessus, qu'une BDB est un multiset, c'est à dire, que les doublons sont permis dans ce type de base de données. On trouve (figure 2.1 (b)) dans r deux transactions ayant les mêmes valeurs. Les transactions 1 et 2 ont des valeurs identiques : F, P et V. Dès lors, la BDB r est un multiset.

[10] Ci-après BDB.

[11] Illustration d'un ensemble de produits dans une épicerie et d'une série de transactions pour laquelle nous désirons analyser le panier d'achat.

Ensemble d'items R

Code	Item
A	Antiacide
B	Bière
C	Croustille
F	Fromage
H	Huîtres
P	Pain
S	Salsa
T	Pâté
V	Vin

(a)

BDB r sur R

Transaction	Items achetés
0	{P, T}
1	{F, P, V}
2	{F, P, V}
3	{P, T, V}
4	{P}
5	{B, C, S}
6	{P, T, V}
7	{B, H}
8	{T}
9	{A, B, C, H, S}

(b)

(Godin, 2006)

Figure 2.1 – (a) Ensemble d'items et (b) BDB

❑

Définition 2. 3– Itemset

Soit R, un ensemble d'items. I est un itemset si $I \subseteq R$. Soit $i \in \mathbb{N}$, on note I_i, l'itemset de la transaction à la position i dans une BDB. On note k-itemset, un itemset ayant une taille de k items.

Exemple 2.2

Dans le tableau de la figure 2.1(b), on trouve les valeurs F, P et V à la transaction 1. On peut donc écrire que I_1 = FPV. Il est dit de cet itemset qu'il est un 3-itemset puisqu'il a 3 items. On peut également affirmer que I_1 est itemset *légal* puisque tous ses items se retrouvent dans R.

❑

On peut utiliser un treillis[12] comme outil graphique pour représenter une collection d'itemset. On nomme d'ailleurs ce dernier, un treillis d'itemsets. Il est à noter que

[12] Le chapitre 4 de ce mémoire sera dédié à la théorie sur les treillis.

l'extraction de la collection des itemsets à partir d'une BDB ne produit pas toujours un treillis à coup sûr.

Exemple 2.3

Soient R l'ensemble d'items suivant {F, P, V} et r sur R, une BDB tel qu'illustré à la figure 2.2. La figure 2.3 présente des le treillis des itemsets de cette BDB[13] une fois extraite par un algorithme quelconque.

BDB r sur R pour l'exemple 2.3

Transaction	Items achetés
0	{P, V}
1	{F, P, V}
2	{F, P, V}
3	{F, V}
4	{P}
5	{P}
6	{P}

Figure 2.2 – BDB sur R pour exemple 2.3

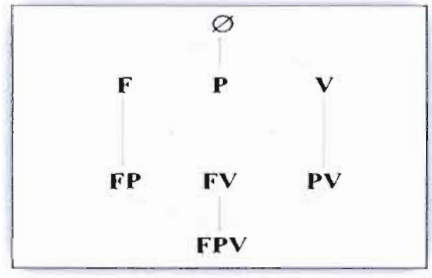


Figure 2.3 – Treillis d'itemsets

❑

[13] Dans certains documents scientifiques, on retrouve le plus grand itemset (qui est habituellement l'ensemble de tous items de R) au sommet du treillis et l'ensemble vide à la base du treillis. Afin de demeurer consistant avec le reste de ce mémoire, nous considérons que l'ensemble vide est à au sommet du treillis et l'itemset contenant tous les items de R est à la base du treillis. Nous verrons, lorsque nous traiterons des treillis de concepts formels, la similitude entre ces deux treillis.

2.2 Fonctions d'évaluation et itemsets fréquents

Une fonction d'évaluation, dans le contexte du forage de données et de la découverte de connaissances, indique la qualité d'un motif[14] ou d'un groupe de motifs examiné à partir d'un ensemble de données. Ci-après deux fonctions de ce genre sont définies.

Définition 2. 4 - Support

Supposons R un ensemble d'items, r une BDB sur R et $Y \subseteq R$ un itemset. On note $M(r, Y) = \{t \in r \mid Y \subseteq t\}$, le multiset de transactions qui contiennent Y. Le support de Y sur M est noté $support(M, Y) = |M(r, Y)|$ et signifie le nombre de transactions dans M correspondant à Y. Le support peut être représenté de manière absolue, relative ou en pourcentage.

Exemple 2.4

Soient R et r tel que définis dans exemple 2.1. Si Y = FPV alors $M(r, Y) = \{1, 2\}$ et le support de Y est $support(M, Y) = |M(r, Y)| = 2$. Si Y = T, on a $M(r, Y) = \{0, 3, 6, 8\}$ et le support sera 4. Supposons maintenant que Y = BV. $M(r, Y) = \varnothing$ car aucune des transactions ne contient le couple B et V. Le support sera alors 0.

Itemset	Notation Absolue	Notation Relative	Notation Pourcentage
FPV	2	2/10 ou 0.20	20%
T	4	4/10 ou 0.40	40%
BV	0	0/10 ou 0.00	0%

❑

[14] Dans notre cas, le motif est un itemset.

Définition 2. 5- Fréquence

Supposons R un ensemble d'items, r une BDB sur R et $Y \subseteq R$ un itemset. La fréquence de Y est définie de la façon suivante : $f(r, Y) = support(r, Y)/|r|$. En d'autres termes, la fréquence n'est autre que le support en notation relative.

Il est courant d'ajouter le support ou la fréquence aux itemsets du treillis des itemsets. La figure suivante est basée sur l'exemple 2.3.

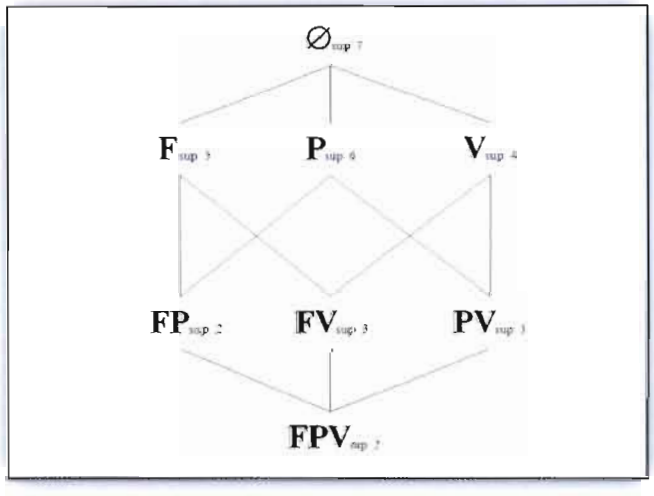


Figure 2.4 – Treillis d'itemsets et support

Notons, à la figure précédente, que l'itemset ayant la valeur ∅ a un support de 7. Selon la théorie des ensembles, l'ensemble vide est inclus dans tout ensemble. Cette notion s'applique également à la collection des itemsets et à la collection des transactions d'une BDB. Puisque cette dernière compte 7 transactions (figure 2.2), l'itemset ∅ aura un support valant 7. De façon générale, le support de l'itemset ∅ pour toute BDB est égal à $|r|$, où r est une BDB.

Autre fait important, illustré par la figure précédente, est le nombre d'itemsets que comporte le treillis. Se basant sur la théorie des ensembles, le nombre d'itemsets basé sur R, un ensemble d'items, est $2^{|R|}$. Dans l'exemple 2.3, on a $|R| = 3$ produisant 8 itemsets. Par contre, dans le cas de l'exemple 2.1, l'ensemble R compte 9 items, le nombre d'itemsets sera de 512 ou 2^9. La définition suivante généralise ce qui vient d'être illustré.

Propriété 2 1 – Taille du treillis des itemsets

Supposons R un ensemble d'items. La taille d'un treillis complet d'itemsets est $2^{|R|}$.

La propriété précédente implique des conséquences que les petits exemples, illustrés plus haut, ne reflètent pas. On n'a qu'à penser aux magasins à grande surface possédant des milliers de produits. Prenons par exemple, une grande surface offrant à sa clientèle 5000 produits différents. Le nombre d'itemsets sera donc 2^{5000}. Il et clair que la complexité spatiale du treillis d'itemsets est exponentielle. Du point de vue d'un gestionnaire de produits, on peut se poser la question quant à l'utilité d'avoir à sa disposition comme information tous les itemsets possibles et, disons, leurs supports respectifs. Règle générale, un gestionnaire de produits recherche des produits ayant ou rencontrant des critères spécifiques. Dans les termes de ce mémoire, on transposera ces critères spécifiques en fonctions d'évaluation. Plus loin dans ce mémoire on les rencontre comme contraintes. Le prochain exemple illustre ce qui vient d'être dit.

Exemple 2.5

Un gestionnaire de produit est intéressé à connaître quels sont les mix de produits les plus populaires auprès des consommateurs. Les mix des produits peuvent être représentés par des itemsets et seuls ceux qui sont populaires, c'est à dire qui ont un support minimal choisi par le gestionnaire, seront retenus.

❑

Ce qui vient d'être illustré est ce que l'on nomme les itemsets fréquents. Les prochaines définitions nous les présentent.

Définition 2. 6 – Itemsets fréquents

Supposons R un ensemble d'items, r une BDB sur R et $Y \subseteq R$ un itemset. On note par σ un seuil de support minimum défini par un utilisateur. On définit un ensemble d'itemsets σ-fréquents comme suit : $Freq(r, \sigma) = \{Y \mid Y \subseteq R \wedge support(r, Y) \geq \sigma\}$.

Dans la même lignée que la définition précédente, une fonction d'évaluation complémentaire est proposée dans (Boulicaut, Bykowski, & Rigotti, 2003).

Définition 2.7 – Itemsets fréquents et support associé

Soient R, r, Y et σ tels que définis dans la définition 2.7. On définit l'ensemble des paires d'itemsets fréquents et leurs supports associés comme suit, $FreqSup(r, \sigma) = \{<Y, support(r, Y)> \mid Y \subseteq R \wedge support(r, Y) \geq \sigma\}$.

Exemple 2.6

Soient R, r et Y définis tel que présentés dans l'exemple 2.3. Supposons que le support minimum choisi σ est 3. $Freq(r, \sigma)$ = {∅, F, P, V, FV, PV} et $FreqSup(r, \sigma)$ = {(∅, 7), (F, 3) , (P, 6), (V, 4), (FV, 3), (PV, 3)}

❑

Référons-nous à la figure 2.4 et traversons le treillis d'itemsets en profondeur en débutant par l'itemset au sommet du treillis, on notera une caractéristique intéressante. Tout d'abord, on traite l'itemset ayant la valeur à ∅, puisque son support est plus grand que 3, cet

itemset est fréquent et ces enfants seront examinés. Le prochain itemset a la valeur F. Son support est exactement 3, il est donc accepté comme itemset fréquent. On passe alors au prochain itemset, soit FP l'enfant de F. Cet itemset a un support de 2 et est plus petit que le support minimum spécifié (3). Il est alors est élagué. Puisqu'il est élagué, on n'examine pas ses enfants. Pour s'assurer qu'il n'y aucune perte d'information utile lors de cet élagage, notons dans ladite figure que le support de l'enfant de FP, soit FPV, est 2. Il est également plus petit que le σ choisi.

On appelle ce principe, l'anti-monotonicité[15]. Suivant ce principe, en traversant en profondeur le treillis, si un itemset est infréquent[16] alors tous ses enfants le seront. Dès lors, plutôt que de conserver un itemset infréquent, on l'élague et par le fait même ses enfants sont élagués. Ce principe se nomme la réduction de l'espace de recherche. On dit de l'anti-monotonicité que c'est une contrainte.

2.3 Algorithme d'extraction des itemsets fréquents

Parmi les premiers algorithmes extrayant les itemsets fréquents on retrouve l'algorithme APRIORI (Agrawal & Srikant, 1994). Entre autres qualités, cet algorithme utilise la propriété d'anti-monotonicité permettant ainsi d'élaguer les itemsets infréquents. Le but de cet algorithme est d'extraire les itemsets fréquents pour ensuite extraire les règles d'association à l'aide d'un autre algorithme. La figure suivante conceptualise le processus.

---

[15] Le mot anti-monotonicité est maintenant utilisé en français en ce qui a trait au treillis d'itemsets ou de concepts formels.

[16] Le mot *infréquent* est utilisé en français dans la littérature scientifique.

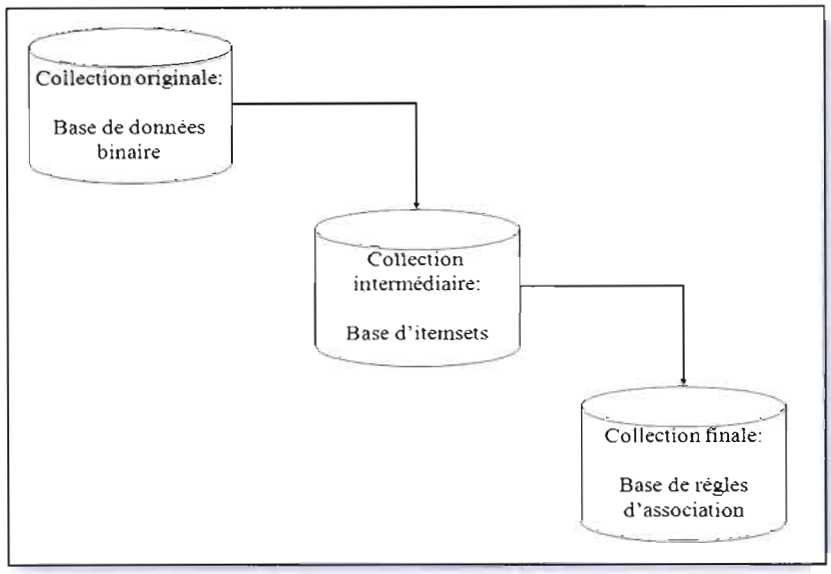


Figure 2.5 – De la BDB à la base d'itemsets à la base de règles

Les deux tableaux suivants présentent respectivement l'algorithme APRIORI et l'algorithme APRIORI_GEN.

Tableau 2.1 – Algorithme APRIORI

▷ Fonction: APRIORI
▷ Paramètres: r une BDB, *minsupport* un support minimum défini pas l'utilisateur.
▷ Retourne: L'ensemble des itemsets fréquents.

FONCTION APRIORI (r, *minsupport*)

(01) $F_1 \leftarrow$ {1-itemsets fréquents}

(02) POUR $k \leftarrow 2$; $F_{k-1} \neq \emptyset$; k++ FAIRE

(03) $C_k \leftarrow$ *APRIORI_GEN* (F_{k-1})

(04) POUR CHAQUE $t \in r$ FAIRE

(05) SI C_k.candidat $\subseteq t$ ALORS candidat.support++

(06) POUR CHAQUE candidat DANS C_k FAIRE

(07) SI candidat.support $\geq$ minsupport ALORS $F_k \leftarrow$ candidat $\cup F_k$

(08) RETOURNER $\cup_k F_k$

Tableau 2.2 – Algorithme APIORI_GEN

▷ Fonction: APRIORI_GEN
▷ Paramètres: F l'ensemble des itemsets fréquents de l'itération précédente calculée par APRIORI.
▷ Retourne: L'ensemble des itemsets candidats pour l'itération en cours dans APRIORI .

FONCTION APRIORI_GEN (F)
(01) POUR CHAQUE paire d'itemsets I_1, I_2 ayant seulement un item différent FAIRE
(02) candidat $\leftarrow I_1 \cup I_2$
(03) $C_k \leftarrow C_k \cup$ candidat si les sous-ensembles du candidat sont fréquents dans F
(04) RETOURNER C_k

Afin de faciliter l'appariement des items et ainsi former des itemsets candidats, on classe les items en ordre lexicographique. L'appariement des items se fait en utilisant le préfixe d'un itemset. La définition suivante explique le principe dudit appariement utilisant le préfixe et le suffixe d'un itemset.

Définition 2.8 – Préfixe et suffixe

Le préfixe d'un itemset fréquent est la partie qui sera évaluée (entre deux itemsets fréquents déjà extraits) afin de créer un itemset candidat qui sera évalué. La longueur du préfixe est la longueur de l'itemset moins un caractère. Le suffixe est la partie restante de l'itemset excluant le préfixe.

Une trace partielle d'APRIORI est proposée ci-après afin d'examiner son comportement[17]. Le jeu de données utilisé est celui de l'exemple 2.1 et le support minimum σ choisi este 2.

A la ligne 01 de APRIORI, on calcule les itemsets 2-fréquents et créer des itemsets de longueur 1 qui seront fréquents. Une technique possible est de balayer les items de l'ensemble R et pour chacun de ces items de calculer leur support en balayant la BDB r. On

[17] Selon (Godin, 2006) chapitre 19

remarque que l'item A possède un support de 1 puisqu'il n'apparait qu'une seule fois dans la BDB *r*. Dès lors, il est élagué. La collection temporaire résultante (couple itemsets fréquents et support) est : {<B, 3>, <C, 2>, <F, 2>, <H, 2>, <P, 6>, <S, 2>, <T, 4>, <V, 4>}.

On passe à la prochaine étape en entrant dans la boucle d'APRIORI. Le comportement d'APRIORI_GEN sera analysé plus loin dans ce texte. Pour le moment on se contente d'accepter le fait que les itemsets candidats produit par APRIORI_GEN à la ligne 03 de l'algorithme APRIORI donnent le résultat illustré à la figure suivante en (a). Aux lignes 04 et 05 d'APRIORI, on balaie chacun des candidats pour calculer le support (en (b)). Et les lignes 06 et 07 procèdent à l'élagage des itemsets candidats infréquents (en (c)).

Itemset Candidat (C_2)
BC
BF
BH
BP
BS
BT
BV
CF
CH
CP
CS
CT
CV
FH
FP
FS
FT
FV
HP
HS
HT
HV
PS
PT
PV
ST
SV
TV

(a)

balayage →

Itemset Candidat (C_2)	Support
BC	2
BF	0
BH	2
BP	0
BS	2
BT	0
BV	0
CF	0
CH	1
CP	0
CS	2
CT	0
CV	0
FH	0
FP	2
FS	0
FT	0
FV	2
HP	0
HS	1
HT	0
HV	0
PS	0
PT	3
PV	4
ST	0
SV	0
TV	2

(b)

Élagage →

Itemset Candidat (C_2)	Support
BC	2
~~BF~~	~~0~~
BH	2
~~BP~~	~~0~~
BS	2
~~BT~~	~~0~~
~~BV~~	~~0~~
~~CF~~	~~0~~
~~CH~~	~~1~~
~~CP~~	~~0~~
CS	2
~~CT~~	~~0~~
~~CV~~	~~0~~
~~FH~~	~~0~~
FP	2
~~FS~~	~~0~~
~~FT~~	~~0~~
FV	2
~~HP~~	~~0~~
~~HS~~	~~1~~
~~HT~~	~~0~~
~~HV~~	~~0~~
~~PS~~	~~0~~
PT	3
PV	4
~~ST~~	~~0~~
~~SV~~	~~0~~
TV	2

(c)

Figure 2.6 – Première itération dans APRIORI

Le résultat de cette itération produit la collection suivante : {<BC, 2>, <BH, 2>, <BS, 2>, <CS, 2>, <FP, 2>, <FV, 2>, <PT, 3>, <PV, 4>, <TV, 2}. Ce sont tous des itemsets 2-fréquents et ils sont ajoutés à la collection temporaire résultante. Passons maintenant à la prochaine itération.

Dans cette itération le comportement d'APRIORI_GEN sera analysé. La figure suivante propose une trace partielle. Tel qu'indiqué dans la l'instruction 01 de APRIORI_GEN, l'algorithme fait l'appariement des itemsets fréquents déjà calculés. Pour chacun des itemsets, l'algorithme balaie cette collection et apparie les 2-itemsets pour produire des 3-itemsets candidats. Cet appariement se fait selon la définition 2.9. Par exemple, les itemsets BC et BH sont analysés. Le préfixe étant le même, un 3-itemset candidat (BCH) est produit (ligne 02 de APRIORI_GEN).

Comme contre-exemple, l'appariement de BC et CS est analysé. Puisque les préfixes des deux itemsets fréquents ne sont pas les mêmes, en dépit du fait que les deux itemsets ont un item en commun, APRIORI_GEN ne produira pas BCS en vertu de la définition 2.9. BCS étant un candidat fréquent, il sera tout de même généré par l'appariement de BC et BS.

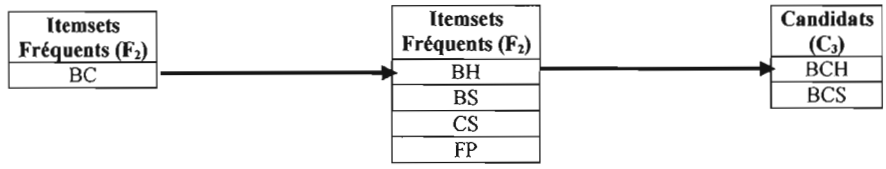


Figure 2.7 – Trace partiel d'APRIORI_GEN

APRIORI_GEN calculera la collection suivante de candidats : {BCH, BCS, BHS, FPV, PTV} et la retournera à APRIORI pour le traitement des 3-itemsets 2-fréquents. Une fois l'exécution de APRIORI terminée, la collection suivante des itemsets 2-fréquents sera : {B, C, F, H, P, S, T, V, BC, BH, BS, CS, FP, FV, PT, PV, TV, BCS, FPV, PTV}.

Ceci conclue la présentation de l'algorithme APRIORI. La section suivante aborde les règles d'association.

2.4 Règles d'association

Un problème bien connu dans le monde des ventes et du marketing et celui de l'analyse du panier à provisions du consommateur. L'extraction des itemsets fréquents permet de trouver les mix de produits les plus populaires. Dans cette section, l'analyse du panier à provisions est poussée un peu plus loin à l'aide des règles d'association. Pour ce faire débutons par un exemple.

Exemple 2.7

Le gestionnaire de produit de l'exemple 2.5 est intéressé non seulement au mix de produits les plus populaires mais de connaitre dans quelle mesure l'achat d'un produit implique automatiquement l'achat d'un autre produit.

❑

L'exemple précédent illustre bien le besoin du gestionnaire et le rôle que jouent les règles d'association pour résoudre ce problème. Ci-après différentes définitions relatives aux règles d'association sont proposées.

Définition 2.9 – Règle d'association

Soient R un ensemble d'items et, A_i et $C_j \in R$. Une règle d'association basée sur R est une expression de la forme $\{A_1, A_2, \ldots, A_n\} \Rightarrow \{C_1, C_2, \ldots, C_n\}$, où $\{A_1, A_2, \ldots, A_n\}$ et $\{C_1, C_2, \ldots, C_n\}$ sont des ensembles disjoints d'éléments. Par disjoint, on entend que $\{A_1, A_2, \ldots, A_n\} \cap \{C_1, C_2, \ldots, C_n\} = \varnothing$. Une règle d'association se note comme suit, $X \Rightarrow Y$, où $X = \{A_1, A_2, \ldots, A_n\}$ et $Y = \{C_1, C_2, \ldots, C_n\}$. Dans le cas d'une règle d'association, X est appelé l'antécédent et Y le conséquent. Une règle d'association, pour être valide, doit avoir $Y \neq \varnothing$.

Exemple 2.8

Soient R et r tel que définis dans l'exemple 2.1. L'énoncé A $\Rightarrow$ BCHS est une règle d'association valide car A $\cap$ BCHS = $\varnothing$. Par contre la règle A $\Rightarrow$ AB n'est pas valide puisque A $\cap$ AB = A $\neq \varnothing$.

❑

Définition 2.10 - support

Soient R un ensemble d'items, r une BDB basé sur R, X et $Y \subseteq R$ respectivement l'antécédent et le conséquent d'une règle d'association. Le support d'une règle d'association $X \Rightarrow Y$ se définit comme suit : $support(r, X \Rightarrow Y) = support(r, X \cup Y)$.

Définition 2.11 – confiance

Soient R, r, X et Y tels que définis précédemment. La confiance représente le ratio du nombre de transactions supportant la règle (antécédent et conséquent) sur le nombre de transactions supportant l'antécédent. On note comme suit : $confiance(r, X \Rightarrow Y) = support(r, X \Rightarrow Y) / support(r, X)$.

Exemple 2.9

Soient R et r tel que définis dans l'exemple 2.1 et les itemsets fréquents et leurs supports respectifs tels que présenté dans le tableau suivant. Calculons la confiance de certaines règles.

Itemset Fréquent	**Support**	**Itemset Fréquent**	**Support**
B	3	BS	2
C	2	CS	2
F	2	FP	2
H	2	FV	2
P	6	PT	3
S	2	PV	4
T	4	TV	2
V	4	BCS	2
BC	2	FPV	2
BH	2	PTV	2

confiance(A $\Rightarrow$ BCHS) = support(A $\cup$ BCHS)/support(A) = 1/1 = 100%
confiance(ABCH$\Rightarrow$S) = support(ABCH $\cup$ S)/support(ABCH) = 1/1 = 100%
confiance(P$\Rightarrow$V) = support(P $\cup$ V)/support(P) = 4/6 = 67%
confiance(V$\Rightarrow$P) = support(V $\cup$ P)/support(V) = 4/4 = 100%

❑

Un même itemset peut générer plus d'une règle d'association (voir exemple précédent). De plus, ces règles peuvent ne pas avoir la même confiance comme en témoigne les règles issues de l'itemset PV. Passons maintenant à quelques définitions.

Définition 2.12 – Règles d'association exactes et approximatives

Si une règle d'association à une confiance de 100%, elle est dite exacte sinon elle est dite approximative.

Définition 2.13 – Règles fortes

Soient σ un seuil de support minimal et κ et un seuil de confiance minimal, tous deux définis par l'utilisateur. Une règle d'association est dite forte si le support et la confiance sont plus grands ou égaux à leurs seuils respectifs.

Une fois les itemsets fréquents extraits, on peut extraire les règles d'association répondant à la contrainte de seuil de confiance minimal. GEN_RULES, illustré au tableau suivant, est un exemple d'un tel algorithme.

Tableau 2.3 – Algorithme GEN_RULES

▷ Fonction: GEN_RULES
▷ Paramètres: *F* l'ensemble de tous les itemsets fréquents, *minconf* le seuil de confiance minimum défini par l'utilisateur.
▷ Retourne: L'ensemble de règles satisfaisant le seuil de confiance minimum

```
FONCTION GEN_RULES (F, minconf)
(01) Règles ← ∅
(02) POUR CHAQUE itemset fréquent Fl dans F FAIRE
(03)     POUR CHAQUE sous-ensemble IS de l'itemset Fl LOOP
(04)         Générer règle IS ⇒ Fl \ IS
(05)         règle.support(IS ⇒ Fl \ IS) ← support(Fl)
(06)         règle.confiance(IS ⇒ Fl \ IS) ← support(Fl) / support(IS)
(07)         SI confiance (IS ⇒ Fl \ IS) ≥ minconf ALORS Règles ← Règles ∪ règle
(08) RETOURNER Règles
```

La boucle externe traite les itemsets individuellement. La boucle interne (ligne 03) traite chaque sous-ensemble possible d'un itemset. À la ligne 04, une règle d'association est générée. Par exemple, si l'itemset BC est traité, quatre sous-ensembles seront générés soit : {∅, B, C, BC}. Afin de simplifier, admettons que seuls B et C sont des sous-ensembles dignes d'intérêt pour construire des règles d'association. Dès lors, deux règles d'association sont générées soit B ⇒ C et C ⇒ B. Supposons que le seuil de support est 2 et que le seuil de

la confiance est 100%[18]. Les deux règles passent le test du support minimal. Pour ce qui est de la confiance la règle B ⇒ C nous donne 66% alors que la règle C ⇒ B nous donne 100%. Puisque le seuil de confiance minimal choisi est de 100%, seul C ⇒ B est retenu (ligne 07). Le tableau suivant présente toutes les règles d'association extraite par GEN_RULES et rencontrant les seuils minimaux.

Itemset Fréquent	*Règle*	*Support*	*Confiance*
BC	C⇒B	2/10	2/2
BH	H⇒B	2/10	2/2
BS	S⇒B	2/10	2/2
CS	C⇒S S⇒C	2/10	2/2 2/2
FP	F⇒P	2/10	2/2
FV	F⇒V	2/10	2/2
PV	V⇒P	4/10	4/4
BCS	C⇒BS S⇒BC BC⇒S BS⇒C CS⇒B	2/10	2/2 2/2 2/2 2/2 2/2
FPV	F⇒PV FP⇒V FV⇒P	2/10	2/2 2/2 2/2
PTV	TV⇒P	2/10	2/2

Figure 2.8 – Règles exactes

Ceci termine la section dédiée aux règles d'association.

2.5 Discussion

Dans ce chapitre, les itemsets, le treillis d'itemsets et des fonctions d'évaluation utiles pour l'extraction des itemsets ont été présentés. Le treillis des itemsets nous démontre que sa complexité spatiale et sa complexité temporelle sont de l'ordre de l'exponentiel si nous extrayons tous les itemsets. Un moyen pour contourner ce problème est de se limiter aux itemsets ayant un support égal ou plus grand que le seuil de support choisi par

[18] Autrement dit, on veut uniquement les règles exactes.

l'utilisateur. Ainsi, la collection extraite n'aura que des itemsets fréquents. Les règles d'association et l'algorithme GEN_RULES ont été également abordés. Nous avons exposé la hiérarchisation de l'extraction des règles d'association par le processus illustré à la figure 2.5. À partir, d'une collection initiale (une BDB) l'extraction des itemsets (fréquents ou non) est produite (i.e. la collection intermédiaire). De cette dernière, on extrait les règles d'association dépassant les seuils de support et de confiance.

Se référant à ladite figure, le sujet de notre mémoire portera sur l'extraction de la collection intermédiaire. Attardons-nous un moment sur l'extraction des itemsets et des itemsets fréquents et identifions certains problèmes connus.

Il est connu que l'algorithme APRIORI extrait les itemsets fréquents en fonction du seuil de support minimal choisi par un utilisateur. Si ce dernier choisit un seuil de 0, on retourne donc à la case de départ et tous les itemsets seront extraits. Produire ces derniers sur des BDB de grande taille nous amènera à réaliser la difficulté et même l'impossibilité de les extraire. Du point de vue de la réalisation de APRIORI, il suffirait de valider que le seuil choisi ne soit pas 0. Cependant, si le seuil choisi est trop petit et que la BDB est de très grande taille, il est probable que l'extraction des itemsets fréquents soit impossible. Étant donnée des BDB dont les tailles ne cessent de croître, il devient souvent difficile, et dans certains cas impossible, d'extraire les itemsets fréquents.

Le deuxième problème est au niveau de l'algorithme APRIORI lui-même. Bien que cet algorithme permette l'élagage des itemsets infréquents en utilisant la propriété d'anti-monotonicité, il n'en demeure pas moins qu'APRIORI balaie la BDB pour chaque itemset afin de calculer le support. Ces lectures et relectures affectent grandement sa performance. Plusieurs chercheurs ont proposé des algorithmes permettant de réduire le nombre de balayage.

Notre contribution se situe au niveau du deuxième problème. Depuis quelques années déjà, on cherche à condenser la représentation intermédiaire par des représentations condensées telles que la famille des itemsets fermés, la famille des générateurs minimaux et celles des concepts formels. Des deux premières familles, les représentations peuvent être encore plus condensées[19] grâce à l'utilisation du seuil de support minimal mais aussi en incluant une ou des marges d'erreur permettant de réduire le nombre des fermés ou des générateurs[20]. Dans le cas des treillis de concepts, l'utilisation du seuil de support minimal a donné lieu à un semi-treillis de concepts appelé l'iceberg de concepts formels (Stumme, Taouil, Bastide, Pasquier, & Lakhal, 2002). Plus récemment, certains centres de recherche se penchent sur la possibilité de fusionner des concepts formels afin d'en réduire leur nombre. Évidemment, les concepts formels fusionnés incluent des erreurs bornées par un expert du domaine et par le fait même ces concepts ne sont plus formels. Dans certains milieux, on les appelle concepts tolérant aux exceptions ou concepts denses et pertinents[21]. Notre contribution s'inscrit dans cette voie.

Dans cette discussion nous avons effleuré l'expression "représentation condensée". Un formalisme a été proposé pour les représentations condensées. Ce sera le sujet du prochain chapitre.

---

[19] Certains les appellent des représentations concises.

[20] Les ensembles fermés seront couverts dans le prochain chapitre et les générateurs dans le chapitre 6 de ce mémoire.

[21] Nous les couvrirons dans le chapitre sur l'état de l'art.

CHAPITRE III
REPRÉSENTATIONS CONDENSÉES

Ce chapitre examine la théorie sur les représentations condensées. On doit à Mannila et Toivonen la formalisation de cette théorie (Mannila & Toivonen, 1996). En effet, le principe des représentations condensées existait bien avant cette formalisation. On n'a qu'à penser aux cubes de données, aux itemsets fréquents et aux treillis de concepts formels pour s'en convaincre. Cependant, l'avènement de ce formalisme a l'avantage d'offrir un cadre générique pour tout motif de données et un gabarit pour mesurer la pertinence et l'exactitude ou la conformité d'une représentation dite condensée.

La première section de ce chapitre débute par une illustration du problème des représentations non condensées en utilisant le problème décrit dans la discussion du chapitre précédent. La section 3.2 présente de façon plus formelle l'énoncé du problème et le cadre dans lequel il se situe. La section 3.3 présente le formalisme des représentations condensées suggéré dans (Mannila & Toivonen, 1996). Ce chapitre se termine par une discussion.

3.1 Illustration du problème

Reprenons l'exemple de l'épicerie DuCoin (Godin, 2006) vu au chapitre 2 de ce mémoire à l'exemple 2.1. Le calcul de tous les itemsets possibles[22] de r produit 2^9 ou 512 itemsets. Plusieurs de ces derniers ont un support de 0. Dans les faits, il n'y a qu'une trentaine d'itemsets qui ont support plus grand que 0 dans *r*. Le chapitre précédent illustre le

[22] Ou produire le treillis des itemsets

comportement d'APRIORI avec un seuil de support à 2. Vingt itemsets 2-fréquents ont été trouvé, c'est à dire une réduction de 96% par rapport à la collection originale. Les auteurs de (Mannila & Toivonen, 1996) affirment donc qu'une "collection d'ensembles fréquents peut servir de représentation condensée en entrée pour une base de règles utilisant le OU et/ou des négations.[23]"

Une réduction de 96% sur la collection d'itemsets fréquents, en utilisant un seuil de support minimal de 2, n'est cependant pas possible si un magasin à grande surface a 5000 produits disponibles en tablettes. D'où l'importance de bien choisir les seuils minimaux. Cependant, en dépit d'un choix judicieux des seuils minimaux, les auteurs de (Boulicaut, Bykowski, & Rigotti, 2003) ont souligné que dans le contexte où les BDB sont denses, on assiste à une explosion combinatoire d'itemsets fréquents. En général, la densité de la plupart des BDB des grands détaillants va de creuse à semi-dense. Mais tel n'est pas le cas pour tous les secteurs d'activités où les bases de données et les outils OLAP sont utilisés. De ce nombre, on retrouve les bases de données sur l'expressivité des gènes, la journalisation des entrées du Web, les données amassées lors des collisions proton-proton en physique nucléaire. Outre le problème de densité, les auteurs de (Boulicaut, Bykowski, & Rigotti, 2003) notent les problèmes d'intégrité des données. Ce problème est causé par des entrées de données erronées soit par l'être humain ou par des instruments de mesure ou encore des capteurs.

La prochaine section énonce le problème et son cadre tel que présentés dans (Mannila & Toivonen, 1997).

[23] Notre traduction de "collection of frequent sets can actually serve as a condensed representation of the input data for rules with disjunction or negation"

3.2 Cadre conceptuel et énoncé du problème

3.2.1 Cadre conceptuel

La définition suivante propose le cadre conceptuel de la théorie des représentations condensées.

Définition 3.1 – Cadre conceptuel

Le cadre conceptuel de la théorie des représentations condensées repose sur les éléments suivants.

Soient R un ensemble d'items, r une base de données sur R, ℓ un langage de représentations des propriétés ou de regroupement de données et q un prédicat de sélection. Le prédicat q évalue si une phrase $\varphi \in \ell$ définit d'intéressantes sous-classes dans r. L'objectif est de trouver la théorie de r en fonction de ℓ et q, i.e., l'ensemble $Th(\ell, r, q) = \{ \varphi \in \ell \mid q(\varphi, r)$ est vrai$\}$.

Tel que noté dans (Mannila & Toivonen, 1996) si ℓ est infini et que $q(\varphi, r)$ est satisfait pour une infinité de solutions, la théorie Th ne peut être calculée. Il faut ajouter, tel qu'observé dans le cas des itemsets extraits de BDB de grande taille, que Th ne peut être calculée dans certains cas finis où la complexité temporelle et/ou spatiale est de l'ordre de l'exponentiel. L'exemple suivant illustre la définition précédente.

Exemple 3.1

- Soient R l'ensemble d'items proposé r une BDB sur R tel que proposé dans l'exemple 2.1 du chapitre précédent.
- Soit $Y \subseteq R$ un itemset
- Soit ℓ la collection de tous les itemsets possibles sur R
- Soit σ un seuil de support minimal défini par un utilisateur
- Soit $q(r, Y)$ la fonction d'évaluation $support(r, Y) \geq \sigma$

On aura $T = Th(\ell, r, q)$, l'ensemble des itemsets fréquents en fonction de r et $q(r, Y)$.

❑

L'exemple précédent utilise la fonction d'évaluation *support* et un seuil de support pour produire les itemsets fréquents. Dans le cadre conceptuel, une expression plus large est utilisée afin de généraliser le plus possible cette définition, ce qu'on appelle les motifs intéressants.

Définition 3.2 – Motifs intéressants

Les motifs intéressants forment une collection de tout motif sélectionné par un prédicat et rencontrant les besoins d'un utilisateur.

3.2.2 Énoncé du problème

On recense au moins 3 types de problèmes pouvant rendre l'extraction d'une théorie *Th* difficile et quelquefois impossible.

- Taille de la collection initiale trop grande ou nombre trop grand d'items

Il est possible qu'une base de données transactionnelle (collection initiale) ait des centaines de millions d'enregistrements[24]. L'extraction de ces données pour créer une collection intermédiaire peut prendre un temps machine considérable. Ce temps machine n'est pas toujours disponible pour une société ou entreprise puisque les ressources informatiques doivent servir à pleine capacité en période de pointe aux systèmes transactionnels. De plus, avec la mondialisation grandissante, les périodes hors-pointes se font de plus en plus rares. En d'autres mots, le temps nécessaire pour extraire une théorie *Th,* quand c'est possible de le faire, n'est pas toujours disponible.

Le nombre d'items ou d'éléments peut aussi être trop grand. L'exemple du magasin à grande surface ayant 5000 produits différents sur les tablettes en fait foi. Si le choix est la collection des itemsets comme collection intermédiaire et qu'un algorithme quelconque requiert un treillis d'itemsets comptant toutes les valeurs possibles d'itemset en fonction de l'ensemble d'items, on est confronté à une complexité spatiale de l'ordre de l'exponentiel et peut être une complexité temporelle du même ordre.

- Prédicats de sélection coûteux

Ce problème a été traité dans le précédent chapitre. Il suffit de se rappeler que si un prédicat demande plusieurs relectures de la collection initiale afin de produire une collection intermédiaire et que la collection initiale est de très grand volume, il sera possible que l'extraction soit coûteuse à produire voire impossible. Également mentionné, une valeur de seuil trop faible pourrait produire une explosion

[24] Certains systèmes transactionnels des géants de la télécommunication produisent des centaines de millions de transactions en un mois seulement. Pour extraire les données pour fin d'analyse, ils ont déployé des systèmes complexes et coûteux qui ne sont pas à la portée de toutes les bourses.

combinatoire des phrases du langage. (e. g. explosion combinatoire des itemsets fréquents) avec les mêmes conséquences que précédemment citées.

- Collections intermédiaires de trop grande taille

Supposons une collection initiale de très grande taille pour laquelle une extraction d'une collection intermédiaire de très grande taille est possible. Il est probable que la taille de cette collection intermédiaire soit si importante qu'elle devient trop grande pour être utile. À titre d'exemple, l'extraction de tous les itemsets et du support associé de la BDB de l'exemple 2.1 au chapitre 2 produit 512 itemsets et ce, sachant qu'il n'y a qu'une trentaine d'itemsets offrant une information utile. Quel est l'utilité d'avoir 512 itemsets à prospecter si une trentaine d'entre eux ont de l'information pertinente ?

3.3 Représentation condensée – un cadre formel

Il existe deux types de représentation condensée :

- La représentation condensée exacte et,
- La représentation condensée approximative.

3.3.1 Représentation condensée approximative

Afin de définir les représentations condensées quelles soient approximatives ou exactes certaines définitions de base doivent être présentées. Commençons par les classes de structures et les classes de requêtes.

Définition 3.3 – Classes de structures

Soit S une classe de structures, celle-ci se définit comme suit. $S = \{ s_i \mid i \in I \}$ où I est un ensemble d'indices.

Définition 3.4 – Classes de requêtes

Soient S une classe de structures tel que défini précédemment et Q une classe une classe de requêtes sur S. Q est un ensemble fini de requêtes de type $\{Q_1, Q_2, \ldots, Q_n\}$. La valeur de $Q \in Q$ sur une classe $s \in S$, noté $Q(s)$, est habituellement mais pas exclusivement, un nombre réel dans l'intervalle [0, 1].

Exemple 3.2

Soit R un ensemble d'items présenté à l'exemple 2.1 du chapitre 2 de ce mémoire. Une classe de structures S sur R comprend toutes les BDB possibles sur R. Ainsi la BDB r de la figure 2.1 (b) est l'une des instances possibles sur l'ensemble d'items R.

Soient Q_R l'ensemble des requêtes de fréquence[25], $Y \subseteq R$ un itemset, $Q_R = \{Q_Y \mid Y \subseteq R\}$ et σ un seuil de fréquence minimal défini par l'utilisateur. L'énoncé suivant permet d'extraire les itemsets fréquents et leurs supports respectifs. $Q_Y = FreqSup(r, \sigma)$.

❑

La définition et l'exemple suivants sont inspirés des textes de (Mannila & Toivonen, 1996) et (Boulicaut, Bykowski, & Rigotti, 2003).

[25] Utilisant par exemple, les fonctions d'évaluation *support*, $\int$ pour la fréquence, *Freq* et *FreqSup*.

Définition 3.5 – Représentation condensée ε-adéquate

Soient S une classe de structures, Q_S une classe de requêtes sur S et ε une valeur d'adéquation définie par un utilisateur. On définit une représentation condensée ε-adéquate sur S en fonction de Q_S avec les éléments suivants.

C est une classe de représentations condensées et $c \in C$ une représentation condensée si :

Soit $rep : S \rightarrow C$, une correspondance entre $s \in S$ et $c \in C$. Soit $m : Q \times C \rightarrow [0, 1]$ une évaluation de la requête[26]. Une représentation condensée est dite ε-adéquate si l'énoncé suivant est satisfait.

$\forall Q \in Q, \forall s \in S$ et $|Q(s) - m(Q, rep(s))| \leq \varepsilon$. Où $Q(s) - m(Q, rep(s))$ signifie la mesure de Q sur s – la mesure de Q sur la représentation de s.

Exemple 3.3

Soient ε = 2, R un ensemble d'items, r une BDB sur R tel que définie dans l'exemple 2.1 et $Y \subseteq R$ un itemset. On note que $r \in S$ puisque r est une instance possible de S. Soient Q_R une classe de requêtes calculant le support et $Q_Y \in Q_R$.

Soit $c \in C$ une représentation condensée calculé à l'aide de la fonction d'évaluation $FreqSup(r, \sigma)$ où σ est 2. La représentation condensée c est illustrée par le tableau ci-dessous. Soit le mappage $rep(r) = c$.

Soient $X \subseteq R$, $X \in c$ et $X = Y$ un itemset. Soient Q_c une classe de requêtes calculant le support et $Q_X \in Q_c$ une requête de support sur X définie comme suit : SI $X \in c$ ALORS RETOURNER le support associé à X SINON RETOURNER 0

[26] L'intervalle [0, 1] est dans la définition originale. Notons que c'est l'intervalle habituel mais non exclusif.

Soit $m : Q \times C \rightarrow [0, |R|]$ l'évaluation de la requête retournant un entier.

Itemset Fréquent	**Support**	**Itemset Fréquent**	**Support**
B	3	BS	2
C	2	CS	2
F	2	FP	2
H	2	FV	2
P	6	PT	3
S	2	PV	4
T	4	TV	2
V	4	BCS	2
BC	2	FPV	2
BH	2	PTV	2

L'énoncé suivant, dans le contexte de cet exemple est toujours vrai. $\forall Q_Y \in Q_S$, $\forall s \in S$, $| Q_Y(r) - m(Q_X, rep(r)) | \leq \varepsilon$. En effet, pour tous les itemsets ayant comme support 0 dans r, la requête $Q_Y(r)$ retourne 0 et $m(Q_X, rep(r))$ retourne 0 également en vertu de la définition de Q_X. Donc $|0 - 0| \leq 2$ est vrai. Pour tous les itemsets Y de r dont le support ≥ 2, $m(Q_X, \text{rep}(r))$ retourne la même valeur que $Q_Y(r)$ et leur soustraction donnera toujours $0 \leq 2$. Enfin, dans le cas où $Q_Y(r)$ retourne un support de 1, $m(Q_X, rep(r))$ retourne 0 également en vertu de la définition de Q_X. Nous aurons alors $|1 - 0| \leq 2$ est vrai. On conclut donc que la représentation condensée $c \in C$ est une représentation condensée 2-adéquate pour r.

❑

3.3.2 Représentation condensée exacte

Une représentation condensée exacte suit la même définition que la représentation condensée approximative mais avec les particularités suivantes.

- La valeur d'adéquation ε est 0 et,
- Idéalement mais pas obligatoirement, il existe une fonction, que nous appellerons fonction d'inférence, permettant de recréer en partie ou en totalité les motifs de la

représentation condensée exacte en motifs de la collection intermédiaire. Cette fonction doit être suffisamment performante pour que les complexités temporelle et spatiale de la construction de la représentation condensée exacte et de la fonction de création des motifs originaux soient égales ou moindre que lesdites complexités associées à la génération de la collection intermédiaire.

Dans la première particularité, le paramètre d'adéquation étant 0 signifie qu'il n'y a aucune perte d'information. Ce qui rend cette représentation séduisante.

Il est à noter de la dernière particularité qu'elle n'est pas toujours obligatoire. En effet, elle dépend de certaines caractéristiques. D'une part, elle peut être non nécessaire, si l'utilisateur ne la juge pas appropriée. D'autre part, les motifs de certaines de ces représentations condensées exactes sont autonomes[27]. Par là, on entend que la génération des motifs de la collection intermédiaire à partir de la représentation condensée exacte n'ajoute pas, pour l'utilisateur, de l'information utile ou pertinente. L'information donnée par les motifs de la représentation condensée exacte est suffisante à elle seule.

L'une de ces représentations condensées exactes les plus connues est la collection des itemsets fermées et elle est présentée à la sous-section suivante.

3.3.2.1 Les itemsets fermés et la représentation condensée exacte

Définition 3. 6 – Opérateurs φ et ψ

Soient R un ensemble d'items, T ensemble de transactions et r une BDB. L'opérateur φ est utile pour calculer l'ensemble des items communs aux itemsets de T, noté $\varphi(r, T)$.

L'opérateur ψ est la duale de φ.

[27] De l'anglais, self-contained.

Exemple 3. 4

Soit r une BDB tel qu'illustrée à l'exemple 2.1.

(1) On veut calculer l'ensemble des items communs de la transaction 9. $\varphi(r, 9)$ = ABCHS.

(2) On veut calculer l'ensemble des transactions communes de l'attribut A. $\psi(r, A) = 9$.

(3) On veut calculer l'ensemble des transactions communes à l'itemset ABCS. $\psi(r, ABCHS) = 9$.

❑

Définition 3.7 – Fermeture

Soient R un ensemble d'items, r une BDB sur R et $Y \subseteq R$ un itemset. La fermeture de l'itemset Y, notée $fermeture(r, Y) = \varphi(\psi(r, Y)$, est le surensemble de Y ayant la plus grande taille et la même fréquence.

Propriété 3.1

Soient r une BDB et I un itemset fermé. L'opérateur de fermeture $fermeture(r, I)$ doit satisfaire aux propriétés suivantes.

1. $I \subseteq fermeture\ (r, I)$
2. $I_1 \subseteq I_2 \Rightarrow fermeture(r, I_1) \subseteq fermeture\ (r, I_2)$
3. $fermeture\ (fermeture(r, I)) = fermeture\ (r, I)$

Exemple 3.5

Soit r une BDB tel qu'illustrée à l'exemple 2.1. Supposons qu'un utilisateur veuille trouver la fermeture de l'itemset de valeur B. La collection des itemsets 2-fréquents est présentée dans le tableau de l'exemple 3.3. De ce tableau on trouve que le support de B est 3 et que B n'a pas de surensemble ayant le même support. Dès lors, $fermeture(r, B)$ = B.

Supposons qu'un utilisateur veuille trouver la fermeture de l'itemset C. À partir du tableau de l'exemple 3.3, on trouve que le support de C est 2. Dès lors, *fermeture*(*r*, C) = BCS. Ce dernier est le surensemble de l'itemset C ayant la plus grande taille et le même support.

❑

La figure suivante présente le treillis des itemsets fermés basé sur la BDB *r* de l'exemple 2.1. Cette collection d'itemsets fermés est un bon exemple pour illustrer l'utilité ou non d'une fonction de génération des itemsets de la collection intermédiaire.

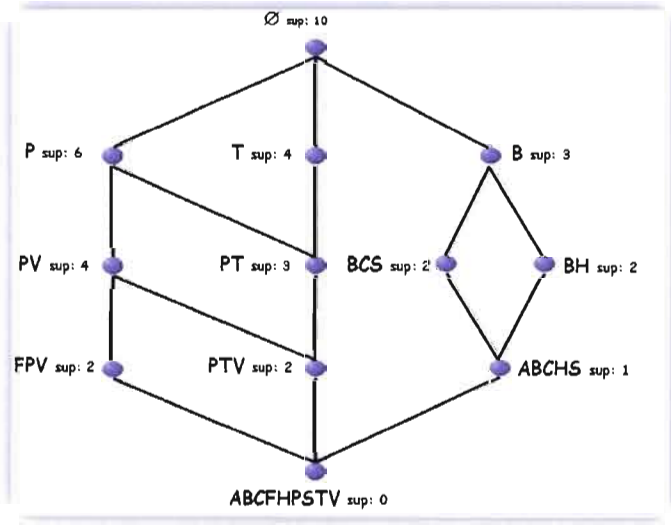


Figure 3.1 – Treillis d'itemsets fermés (avec support)

Par exemple, un analyste de produits peut considérer que la collection des itemsets fermés lui fournit suffisamment d'information. La génération de la collection intermédiaire des itemsets lui serait inutile puisqu'il sait qu'un itemset fermé est un surensemble des itemsets de même support. Il sait également que ce ne sont pas tous les sous-ensembles de l'itemset fermé qui on le même support. Cependant, un regard sur le treillis lui permet de savoir quels sont les sous-ensembles de l'itemset fermé qui n'ont pas le même support.

Exemple 3.6

Un analyste de données tient à savoir quels sont le sous-ensemble d'itemsets provenant de l'itemset fermé FPV qui n'ayant pas le même support que FPV. En examinant le treillis des itemsets fermés (figure 3.1), il se rend compte que seuls V et PV avec un support de 4 ainsi que P avec un support de 6 ne font pas partie des sous-ensembles de l'itemset fermé FPV ayant un support de 2. Les autres itemsets[28] de l'itemset fermé FPV, soient F, FV, FP et FPV auront un support de 2.

❑

Pour ce qui est de savoir si le treillis des itemsets fermés est bien une représentation condensée exacte, il suffit d'appliquer la définition 3.5 mais avec $\varepsilon = 0$.

3.4 Discussion

Ce chapitre s'est penché sur les problèmes que peuvent causer la génération d'une collection intermédiaire par une collection initiale de très grande taille et d'une collection intermédiaire de trop grande taille pour être utile. On a présenté le cadre formel des représentations condensées proposé par Mannila et Toivonen. On trouve qu'il existe deux types de représentations condensées : l'approximative et l'exacte. Cette section illustre le cadre dans lequel une représentation condensée peut être utilisée.

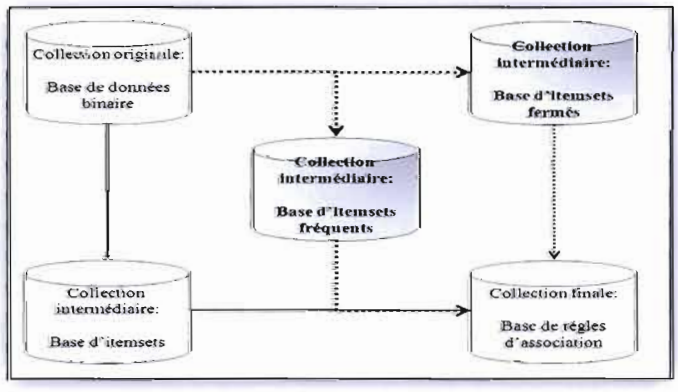


Figure 3.2 – représentations alternatives de la collection intermédiaire

[28] Dans cet exemple, l'itemset ayant la valeur ∅ est exclu de cet ensemble.

La figure ci-dessus représente ce qui a été vu, au chapitre précédent, sur le concept de l'extraction des itemsets comme collection intermédiaire et, de cette dernière, l'extraction des règles d'association. On sait que dans certains cas, il est difficile et même impossible d'extraire la collection intermédiaire. L'idée des représentations condensées est la suivante. Si une représentation condensée est ε-adéquate pour une collection intermédiaire difficile d'extraction alors pourquoi ne pas l'utiliser comme collection intermédiaire pour générer les règles d'association.

Dans les cas de la figure 3.2, il est possible de remplacer la collection de tous les itemsets par une collection d'itemsets 2-fréquents. Si l'utilisateur accepte que la représentation soit 2-adéquate alors l'extraction de la collection intermédiaire devient possible. Si l'utilisateur tient à une représentation condensée exacte avec une fonction quelconque de génération de certains itemsets au besoin alors la représentation condensée des itemsets fermés peut faire office de collection intermédiaire. Naturellement, dans ces deux exemples, on suppose que les deux représentations condensées sont facile d'extraction[29].

Plusieurs notions abordées dans (Mannila & Toivonen, 1996) et (Mannila & Toivonen, 1997) n'ont pas été abordées dans ce chapitre. L'une des plus intéressantes est la théorie sur les frontières[30] positive et négative. Le lecteur intéressé pourra consulter ces documents pour en savoir plus. L'intérêt principal des frontières est donné dans (Mannila & Toivonen, 1997) où les auteurs prouvent que la bordure négative joue un rôle dans le calcul de la complexité temporelle de l'algorithme APRIORI.

Ceci conclut ce chapitre. Le chapitre 5 de ce mémoire examinera une autre représentation condensée de choix : les treillis de concepts formels. Le chapitre suivant traitera des concepts mathématiques des ensembles ordonnés et partiellement ordonnés ainsi que des treillis.

---

[29] Ou au moins plus facile ou possible comparativement à une représentation intermédiaire d'origine

[30] Ou bordure. De l'anglais: border

CHAPITRE IV
RELATION D'ORDRE ET TREILLIS

Ce chapitre examine la théorie des relations d'ordre et des treillis. Le principal objectif de ce chapitre est d'alléger le chapitre suivant sur les treillis de concepts formels. Dès lors, ce chapitre se limite à que ce qui est utile en fonction de la théorie des treillis de concepts formels et du but de ce mémoire. Le format de cette section, étant purement théorique, est de présenter en rafale les définitions jugées importantes ainsi que des exemples pertinents. Sauf indication contraire, le matériel de ce chapitre est inspiré de ces livres suivants : (Carpineto & Romano, 2004), (Lipschutz, 1976) et (Lipschutz & Lipson, 1992).

4.1 Ensembles ordonnées et partiellement ordonnées.

Définition 4.1 – Théorie des relations d'ordre

La théorie des relations d'ordre est une des branches de mathématiques qui étudie différentes relations binaires ainsi que la notion intuitive de comparaison et d'ordonnancement.

Définition 4. 2 – Relation d'ordre (partiel) et propriétés

Soient Σ un ensemble et a, b, et c $\in \Sigma$ et $\leq$ une relation binaire sur Σ. Une relation d'ordre (ou d'ordre partiel[31]) est une relation binaire si cette relation est :

(1) Réflexive e.g. $a \leq a \ \forall a \in \Sigma$
(2) Antisymétrique e.g. si $a \leq b$ et $b \leq a$ alors $a = b$
(3) Transitive e.g. si $a \leq b$ et $b \leq c$ alors $a \leq c$

Si ces propriétés sont satisfaites Σ un ensemble partiellement ordonné (EPO).

Définition 4.3 – Ensembles ordonnés et partiellement ordonnés

Un ensemble ordonné (ou partiellement ordonné[32]) est un ensemble muni d'une relation d'ordre (ou ordre partiel).

Définition 4.4 – Relation de précédence

Une relation de précédence (un ordre partiel), notée $\leq$, est une expression de la forme $a \leq b$ signifiant que a précède b ou b succède a. Une relation de précédence, notée $<$, est une expression de la forme $a < b$ signifiant a précède strictement b et $a \neq b$. Par strictement, on entend qu'il n'existe pas d'élément x pour l'expression $a < x < b = a < b$ et $a \neq b$.

Définition 4.5 - Comparabilité

Soient P un EPO, $a, b \in P$ et une relation d'ordre $\leq$. On dit que a et b sont comparables si l'expression suivante $a \leq b$ ou $b \leq a$ est vrai. Si l'expression est fausse alors on dit des éléments a et b qu'ils sont incomparables.

---

[31] Ci après, ROP

[32] Ci après, EPO est notre traduction de l'anglais POSET

En ce qui a trait à la notation d'un EPO dans une définition ou une illustration, on peut exprimer cette notation de deux manières. Si la relation d'ordre est intrinsèque à l'EPO ou connu du lecteur, elle est exprimée de la façon suivante : « Soit P un EOP... » Si la relation d'ordre sur un EPO n'est pas évidente à discerner ou que l'auteur tient à spécifier cette relation, on utilise alors le couple EPO et sa relation d'ordre, e.g. « Soit $(P, \leq)$ un EPO... ». Il peut être utile d'illustrer un EPO ordonné par une ROP. On utilise habituellement un diagramme de lignes ou diagramme de Hasse.

Exemple 4. 1

Soit $Z = (1, 2, 3, 4, 6, 8, 9, 12, 18, 36)$ un EPO ordonné par une ROP définie comme suit : a divise b et la division est entière. Une illustration possible de cet EPO est donnée à la figure suivante.

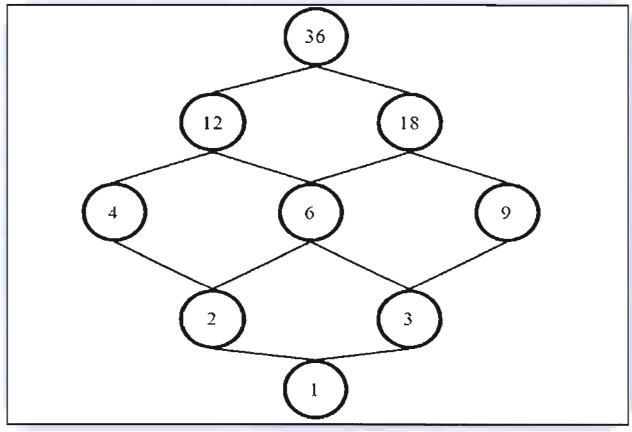


Figure 4.1 – Diagramme de Hasse de la relation de divisibilité

❑

Il existe deux types d'EPO ayant des particularités spécifiques. Elles sont présentées dans la prochaine définition.

Définition 4.6 – Chaine et anti-chaîne

Soit P un EPO. Si chaque paire d'éléments de P est comparable alors on dit que P est une chaîne. Soit Q un EPO. Si aucune paire d'éléments de Q n'est comparable alors on dit que Q est une anti-chaîne.

Exemple 4.2

Soit P un EPO et sa relation d'ordre tel que définie à la figure 4.2(a). On dit de P qu'il est une chaîne puisque chaque paire de P est comparable.

Soit Q un EPO et sa relation d'ordre tel que définie à la figure 4.2 (b). On dit de Q qu'il est une anti-chaîne puisque toutes ses paires d'éléments sont incomparables.

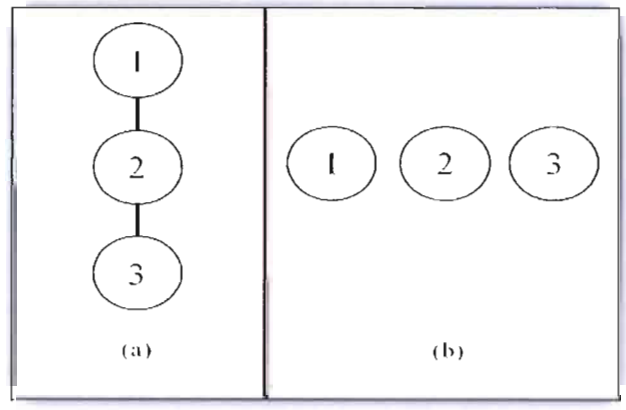


Figure 4.2 – Chaîne et anti-chaîne

❑

Les prochaines définitions seront illustrées à partir de la figure suivante. Les exemples sont tirés de (Lipschutz & Lipson, 1992).

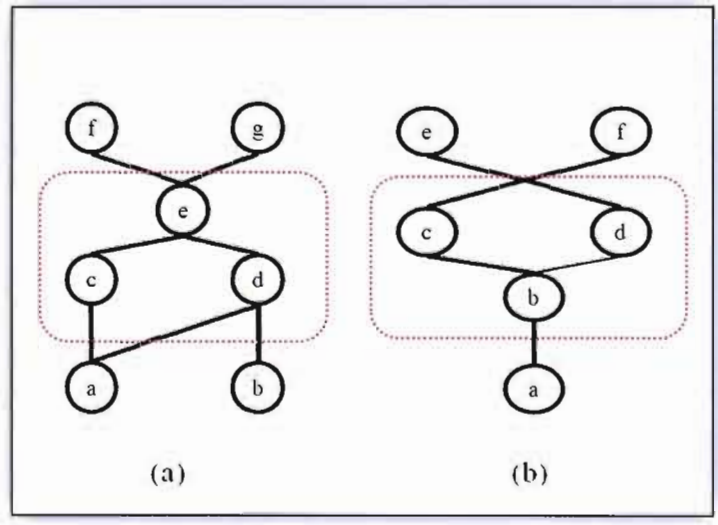


Figure 4.3 – Deux EPO

Définition 4.7 – Élément maximal/minimal

Soit $(P, \leq)$ un EPO, $S \subset P$ et $a, b \in S$. On dit de a qu'il est un élément maximal de S si $a \leq b$ signifie $a = b$. La définition de l'élément minimal est la duale de l'élément maximal. Il est possible qu'un EPO ait plus d'un élément maximal (resp. élément minimal).

Définition 4.8 – Borne supérieure/inférieure

Soient S un sous-ensemble de P un EPO et $p \in P$. L'élément p est appelé borne supérieure de S si $\forall s \in S$ on a $s \leq p$. La borne inférieure est définie comme la duale de la borne supérieure.

Définition 4.9 – Supremum/Infimum

Soient S et P tels que définis dans la définition 4.8. Si une borne supérieure de S précède toutes les autres bornes supérieures de S, cette borne est nommée supremum de S et est notée, sup(S). Si une borne inférieure de S succède à toutes les autres bornes inférieures de S, cette borne est nommée infimum et est notée, inf(S).

Exemple 4.3 – Éléments maximaux/minimaux

(1) Soit P un EPO tel qu'illustré à la figure 4.3(a). Les éléments maximaux de cet EPO sont f et g. Les éléments minimaux de cet EPO sont a et b.

(2) Soient $\mathbb{N}$ l'ensemble des entiers positifs un EPO et $\leq$ une ROP. L'élément minimal est 0 mais il n'y a pas d'élément maximal puisque $\mathbb{N}$ est infini.

❑

Exemple 4.4 – Borne supérieure/inférieure

Soient P = {a, b, c, d, e, f, g} un EPO, $S \subset P$ = {c, d, e} et une ROP sur P tel qu'illustré à la figure 4.3(a). Les éléments 'e', 'f' et 'g' succèdent tous les éléments de S. Ils sont donc des bornes supérieures de S. L'élément 'a' précède tous les éléments de S. Dès lors, 'a' est la borne inférieure de S. Notons que 'b' n'est pas une borne inférieure puisque b $\leq$ c est faux.

❑

Exemple 4.5 – Supremum/infimum

(1) Soient S, P et la ROP tels que définis dans l'exemple précédent. L'élément 'e' de S est le supremum puisqu'il précède toutes les bornes supérieures de S. L'infimum de S est l'élément 'a' puisqu'il succède à toutes les bornes inférieures de S.

(2) Soient P = {a, b, c, d, e, f}, $S \subset P$ = {b, c, d} et une ROP tels que définis à la figure 4.3(b). L'infimum de S est 'b' puisqu'il succède à toutes les bornes inférieures de S. Par contre, il n'y a pas de supremum pour S parce qu'il n'y a aucune borne supérieure et, par conséquent, il ne peut y avoir de supremum.

❑

4.2 Treillis

Définition 4.10 – Les fonctions supremum et infimum

Soient P un EPO et x, y $\in P$. La fonction supremum sur x et y, noté sup(x, y) ou x $\vee$ y, calcule l'élément supremum de x et y dans P. La fonction infimum sur x et y, noté inf(x, y) ou x $\wedge$ y, calcule l'élément infimum de x et y dans P.

Définition 4.11 – Treillis

Soit L un EPO. L est un treillis, quelquefois noté (L, $\wedge$, $\vee$), si pour tous les éléments x et y de L, le couple (x, y) a un supremum et un infimum dans L.

Selon la définition ci-dessus, l'ensemble partiellement ordonné de la figure 4.1 est un treillis. L'exemple suivant illustrera, à partir de ladite figure, les fonctions supremum et infimum.

Exemple 4.6

Soit (L, $\wedge$, $\vee$) un treillis représentant la relation de divisibilité du nombre 36 tel qu'illustré à la figure 4.1.

(1) sup(2,3) = 6. La figure 4.4(a) illustre le calcul.
(2) inf(9, 12) = 3. La figure 4.4(b) illustre le calcul.

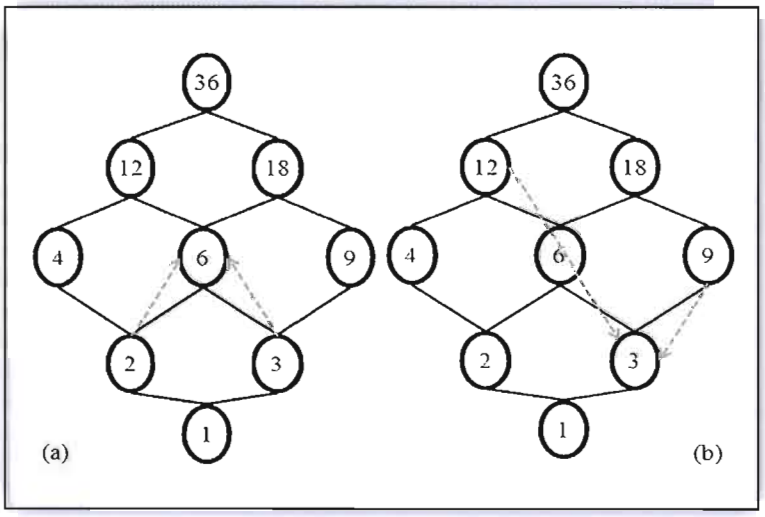


Figure 4.4 – Fonctions supremum et infimum

❑

Définition 4.12 – Semi-treillis-supremum et semi-treillis-infimum

Soit (P, ≤) un EPO. (P, ≤, ∨) est un semi-treillis-supremum si ∀x, ∀y ∈ P, x ∨ y existe.

Soit (P, ≤) un EPO. (P, ≤, ∧) est un semi-treillis-infimum si ∀x, ∀y ∈ P, x ∧ y existe.

Définition 4.13 – Treillis complet

Soit L un treillis. L est un treillis complet si pour tout sous-ensemble S de L on a un supremum et un infimum.

L'illustration suivante présente des treillis et semi-treillis. Respectivement en (a) et (b) sont un semi-treillis-infimum et un semi-treillis-supremum. Les treillis en (c), (d) et (e) sont tous des treillis complets selon la définition 4.13.

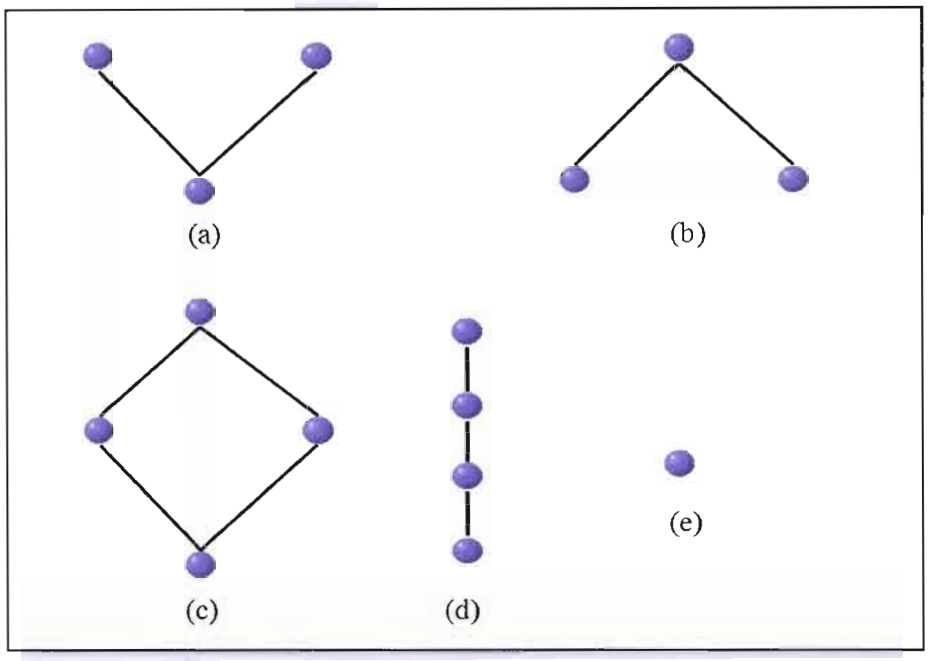


Figure 4.5 – Treillis et semi-treillis

Définition 4.14 – Sous-treillis

Soient $(L, \wedge, \vee)$ un treillis et $L' \subseteq L$ et $S \neq \varnothing$. L' est un sous-treillis de L ssi pour toutes les paires x, y de L' on a $x \vee y$ et $x \wedge y$ dans L'.

Exemple 4. 7

Soit L un treillis tel que présenté à la Figure 4. 6.

(1) Soit L_1 = {a, b, c, h}. Est-ce que L_1 est un sous-treillis de L? Non, car $b \wedge c = e$ alors que l'infimum de L est h.

(2) Soit L_2 = {a, b, g, h}. Est-ce que L_2 est un sous-treillis de L? Oui, car $b \wedge g = h$ et $b \vee g = a$.

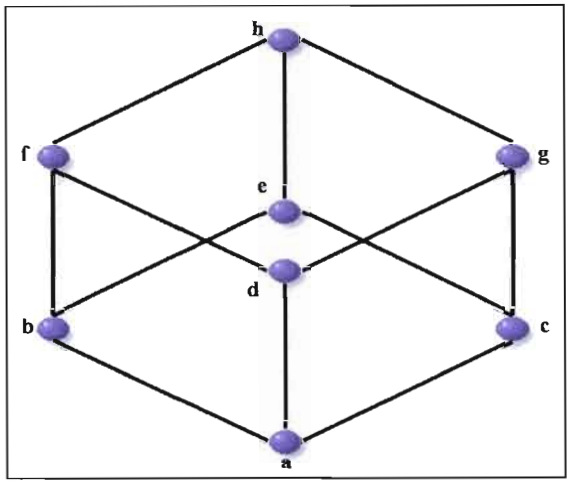


Figure 4. 6 - Treillis

❑

4.3 Discussion

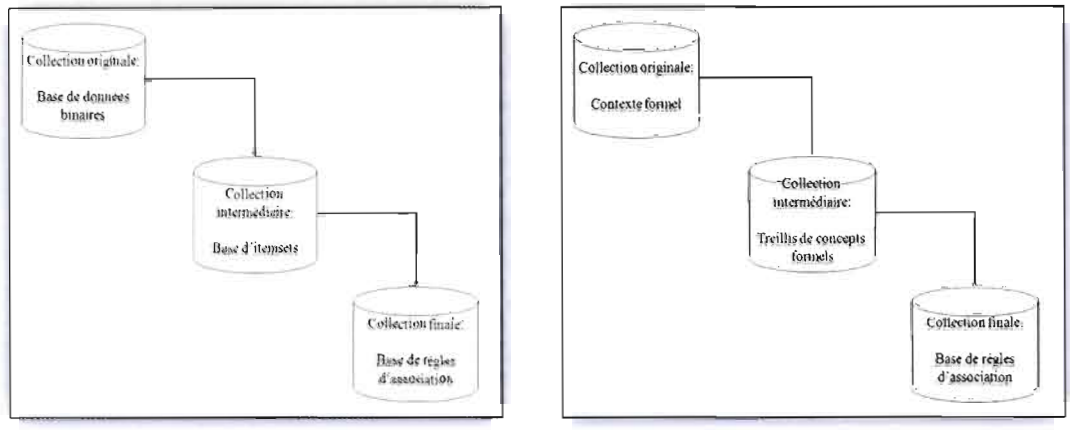
Ceci termine ce chapitre. Les définitions et exemples de ce chapitre ne sont malheureusement pas complets. Par exemple, les éléments irréductibles du treillis, de l'algèbre relatif aux EPO et des morphismes des EPO n'ont pas étés couverts. Cependant, ce qui a été donné dans ce chapitre est la base de tout EPO et, des treillis de concepts formels qui seront vus au prochain chapitre.

CHAPITRE V
TREILLIS DE CONCEPTS FORMELS – NOTIONS DE BASE

Ce chapitre présente les notions de base de l'analyse formelle de concepts (AFC). Les définitions de base de l'AFC sont formulées à la section 5.1. La section 5.2 présente l'iceberg de concepts formels (IFC). La section 5.3 présente l'algorithme MagaliceA. Cet algorithme sera le point de départ pour le projet de ce mémoire. À moins d'indication contraire, les exemples et définitions sont inspirés du chapitre 19 de (Godin, 2006) et du chapitre 1 et 2 de (Carpineto & Romano, 2004).

5.1 Définitions de base

Il a été dit au chapitre 2 qu'afin d'extraire les règles d'association d'une BDB, il faut extraire préalablement la collection des itemsets. L'extraction de cette collection est connue pour être difficile. La figure suivante propose une solution alternative à la collection des itemsets. À partir d'une collection initiale, nommée contexte formel, on extrait la collection des concepts formels, dont la morphologie est représentée par un treillis. Cette représentation condensée permettra l'extraction les règles d'association.



(a) (b)

Figure 5.1 – Processus d'extraction des collections intermédiaire et finale

Parmi les différences entre les deux représentations on remarque la nomenclature des deux premières collections. Ainsi, la collection originale, dans le contexte de l'AFC, se nomme contexte formel (figure 5.1(b)).

Définition 5.1 – Contexte formel

Un contexte formel est un triplet $K = (O, A, I)$ où O est un ensemble d'objets, A un ensemble d'attributs et I une relation d'incidence entre O et A.

La différence majeure entre le contexte formel et la BDB est la suivante. Habituellement, la BDB prend en compte que les items des transactions alors que la définition du contexte formel prends en compte les items (ou attributs) et les transactions (ou objets). Autrement dit, la définition du contexte formel prend en compte les deux dimensions

d'un tableau binaire alors que la définition de la BDB est basée sur une seule dimension, celle des colonnes (ou items)[33].

Exemple 5.1

Soient O un ensemble d'objets tel que défini par la figure 5.2(a) et A un ensemble d'attributs tel que défini par la figure 5.2(b). Le contexte formel $K = (O, A, I)$ est présenté à la figure 5.2(c)[34]. (Godin, 2006)

O, ensemble d'objets

Objet	Description
0	Transaction 0
1	Transaction 1
2	Transaction 2
3	Transaction 3
4	Transaction 4
5	Transaction 5
6	Transaction 6
7	Transaction 7
8	Transaction 8
9	Transaction 9

(a)

A, ensemble d'attributs

Attribut	Description
A	Antiacide
B	Bière
C	Croustille
F	Fromage
H	Huitres
P	Pain
S	Salsa
T	Paté
V	Vin

(b)

K = (*O*, *A*, *I*), contexte formel

	Attribut								
Objet	A	B	C	F	H	P	S	T	V
0						✖		✖	
1				✖		✖			✖
2				✖		✖			✖
3						✖		✖	✖
4						✖			
5		✖	✖				✖		
6						✖		✖	✖
7		✖			✖				
8								✖	
9	✖	✖	✖		✖		✖		

(c)

Figure 5.2 – Contexte formel

❑

[33] Ceci étant dit, cela ne signifie que selon la définition d'une BDB, il est impossible de concevoir ou d'extraire des représentations à partir d'une des deux dimensions, voire les deux. La différence soulevée dans le texte est par rapport à la définition du contexte formel. Cette définition rend explicites les deux dimensions, soient les rangées et les colonnes du contexte formel. De plus, cette définition permet d'inclure la relation d'incidence qui est intrinsèque au contexte formel.

[34] Notons que le contexte formel de la figure 5.2(c) utilise un ✖ pour signifier une relation entre un objet et un attribut. Une case vide signifie qu'il n'y a pas de relation entre un objet et un attribut. Dans certains papiers scientifiques, on utilise la notation binaire 1/0 ou 1 indique une relation entre un objet et un attribut et 0 indique qu'il n'y aucune relation entre l'objet et l'attribut.

Puisque la définition inclue les deux dimensions dans son formalisme, on peut dès lors définir les représentations de ces dimensions.

Définition 5. 2 – Représentation horizontale et verticale sur le contexte formel

Soit *K* un contexte formel tel que défini dans la définition 5.1. La représentation horizontale prend en compte chaque rangée (objet) du contexte associée de ses attributs (colonnes) induits par la relation d'incidence. La représentation verticale prend en compte chaque colonne (attribut) du contexte associée de ses objets (rangées) induits par la relation d'incidence.

Certains algorithmes d'extraction des concepts formels utilisent la représentation horizontale en entrée. Elle nous est naturelle car les rangées sont traitées une à la fois et dans l'ordre d'apparition. D'autres algorithmes utilisent la représentation verticale du contexte en entrée. C'est le cas de l'algorithme qui sera présenté dans ce chapitre. L'atout majeur de cette représentation est la disponibilité immédiate du support pour un attribut donné. Si ce support est plus petit que le seuil de support minimum choisi, on peut élaguer l'attribut dès le début. Ceci implique, dans certains cas, une grande économie de temps machine et d'espace mémoire. Une difficulté existe avec la représentation verticale. Si le format du contexte formel est stocké dans un fichier texte ou binaire dans sa représentation horizontale, une conversion devient nécessaire. Dans le cas, où le contexte formel est stocké dans une base de données relationnelle, la sélection peut se faire sur les deux dimensions de difficulté. Enfin, certains algorithmes utilisent les deux représentations. L'un de ces algorithmes sera examiné aux chapitres 7 et 9 de ce mémoire.

5.1.2 Connexion de Galois et opérateurs de dérivation

À la base de la théorie de l'analyse formelle de concepts, on trouve la connexion de Galois et ses opérateurs φ et ψ (Carpineto & Romano, 2004). À partir de ces fonctions ϕ et ψ des opérateurs de dérivation ont été spécifiés dans le contexte de l'AFC.

Définition 5. 3 – Opérateurs de dérivation

Soient O, A, I, K tel que décrits à la définition 5.1 et, $X \subseteq O$. On définit l'opérateur de dérivation sur X comme suit. $X' = \{a \in A \mid oIa \ \forall o \in X\}$. De même, on définit l'opérateur de dérivation sur $Y \subseteq A$ comme suit. $Y' = \{o \in O \mid oIa \ \forall a \in Y\}$.

Exemple 5.2

Supposons O, A et K tels que décrits dans l'exemple 5.1 et, $X \subseteq O$ et $Y \subseteq A$.

(1) Disons $X = 12$. $X' = 12' = FPV$
(2) Disons $Y = FPV$. $Y' = FPV' = 12$
(3) Disons $Y = ABC$. $Y' = ABC' = 9$
(4) Disons $X = 9$. $X' = 9' = ABCHS$
(5) Disons $Y = ABCHS$, $Y' = ABCHS' = 9$

❑

Les trois dernières lignes de l'exemple précédent indiquent un fait intéressant. Le calcul de la dérivation de $Y = ABC$ produit un $X = 9$. Le calcul de cette dérivation a produit, un ensemble fermé X dans O. De l'application de la dérivation sur $X = 9$ on obtient $Y = ABCHS$. Cette fois-ci, le calcul de la dérivation sur X a produit un ensemble fermé $Y \subseteq A$. La dérivation de ce dernier a produit la même valeur $X = 9$ que calculé dans la ligne (3) de l'exemple. Cette observation permet de définir la notion de concept formel dans l'AFC.

Définition 5.4 – Concept formel

Soient O, A, K tel que décrits à la définition 5.1, $X \subseteq O$ et $Y \subseteq A$. Un concept formel est une paire (X, Y) ayant $X' = Y$ et $Y' = X$. Dans ce cas, X est appelé l'extension et Y est appelé l'intension.

Exemple 5. 3

Supposons O, A et K tels que décrits dans l'exemple 5.1 et, $X \subseteq O$ et $Y \subseteq A$.

(1) La paire (12, FPV) est-elle un concept formel ? Le calcul des dérivations donne : $X' = \text{FPV}$ et $Y' = 12$. Dès lors, (12, FPV) est un concept formel.

(2) La paire (9, ABC) est-elle un concept formel ? Le calcul des dérivations donne : $X' = \text{ABCHS}$ et $Y' = 9$. Dès lors, (9, ABC) n'est pas un concept formel. En revanche, la paire (9, ABCHS) est un concept formel.

❑

Définition 5.5 – Relation d'ordre entre concepts formels issus d'un contexte formel

Soient O, A, K tel que décrits à la définition 5.1 et $\boldsymbol{C}(O, A, K)$ l'ensemble des concepts formels. Supposons $X_1, X_2 \subseteq O$ et $Y_1, Y_2 \subseteq A$. Supposons également que (X_1, Y_1) et $(X_2, Y_2) \in \boldsymbol{C}(O, A, K)$.

On dit de (X_1, Y_1) qu'il est un sous-concept de (X_2, Y_2)[35], noté $(X_1, Y_1) \leq (X_2, Y_2)$, si $X_1 \subseteq X_2$ (ou de façon équivalente $Y_2 \subseteq Y_1$). On peut noter la collection des concepts formels en incluant la relation d'ordre comme suit : $\boldsymbol{C}(O, A, K, \leq)$.

La notion de sous-concept est aussi appelée la couverture. Dans ce cas on dit que (X_1, Y_1) est couvert par (X_2, Y_2) ou, inversement, que (X_2, Y_2) couvre (X_1, Y_1)[36].

[35] Ou inversement, que (X_2, Y_2) est un super-concept de (X_1, Y_1).

[36] Pour ce qui de la représentation symbolique de la couverture, on la note habituellement comme suit : $(X_1, Y_1) \prec (X_2, Y_2)$ signifiant que (X_1, Y_1) est couvert par (X_2, Y_2).

Cette relation d'ordre donne lieu à une représentation géométrique particulière à la collection des concepts formels soit, le diagramme du treillis de concepts formels[37]. La figure suivante présente le TCF extrait du contexte formel de l'exemple 5.1.

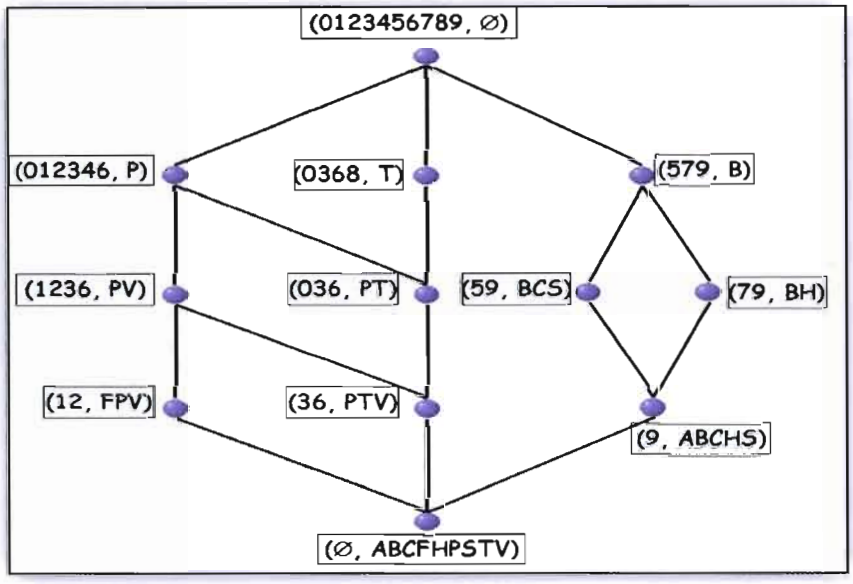


Figure 5.3 – Treillis de concepts formels basé sur le contexte formel DuCoin

Définition 5.6 – Notation pour le sommet et la base du TCF

On note par $\perp$ la base du TCF et par $\top$ le sommet du TCF ou de l'ICF.

Un fait intéressant à noter quant au treillis de concepts présenté ci-haut et le treillis d'itemsets fermés vu 3 à la figure 3.1 du chapitre. On remarque la similitude morphologique

[37] Ou habituellement le treillis de concepts. Dans ce mémoire nous seront explicites quant à la nomenclature du treillis ou de la collection des concepts. Dans ce chapitre et le prochain nous parlons uniquement des concepts formels. Par contre à partir du chapitre 7 de ce mémoire, nous présenterons les concepts denses qui partagerons certaines notions de l'AFC et possèdent leur treillis. Lorsque les expressions collection de concepts ou treillis de concepts seront rencontrées, cela signifiera que la collection ou le treillis est mixte quant à la caractérisation du concept (formel ou dense) ou bien que le sujet traité s'appliquera aux deux types de concepts

des deux treillis. Si le treillis de concepts était dépouillé d'extensions, on obtient le même treillis que celui du treillis d'itemsets fermés. En effet, il y a correspondance parfaite entre les intensions des concepts formels du TCF et les itemsets fermés du treillis d'itemsets fermés (TIFe) issu de la même BDB ou contexte formel. Ce n'est pas sans surprise car l'ensemble des intensions du treillis est l'ensemble des Y fermés $\subseteq A$. Il a été dit au chapitre 3 de ce mémoire que la collection ou treillis des itemsets fermés est une représentation condensée ε-adéquate de la collection ou du treillis des itemsets. Dès lors, de part la complète similitude des intensions des concepts formels et des itemsets fermés on peut dire du TCF qu'il est une représentation condensée 0-adéquate du treillis d'itemsets et ce, au même titre que les treillis d'itemsets fermés.

Admettons, pour le moment, que le treillis de concepts formels joue le même rôle que le treillis d'itemsets dans les exemples évoqués aux chapitres 2 et 3 de ce mémoire, c'est à dire que le TCF est un outil d'analyse pour trouver les mix de produits et leur fréquence. Dès lors, quel rôle ont les extensions dans le TCF ? C'est, entre autres, ce que sera examiné dans la section suivante.

5.2 Fonctions d'évaluation et Iceberg de concepts formels

Deux fonctions d'évaluation sur les TCF, soient le support et la fréquence, sont définies ci-dessous.

Définition 5.7 - Support

Soient $\boldsymbol{C}(O, A, K)$ l'ensemble des concepts formels et $c \in \boldsymbol{C}(O, A, K)$. On définit le support d'un concept formel c, noté $support(c)$, comme étant la cardinalité de l'extension de c ou $support(c)$ = |extension de c|.

Définition 5.8 - Fréquence

Soient $\boldsymbol{C}(O, A, K)$ et c tels que définis précédemment. On définit la fréquence d'un concept formel c comme suit : $\nexists(c) = \text{support}(c)/|O|$.

On se rappelle que la fonction APRIORI doit relire la BDB à chaque fois pour connaître le support d'un itemset candidat. Dans le cas des concepts formels, le support est intrinsèque au concept formel puisque calculé à même l'extension. On admet qu'il existe des algorithmes qui extraient les concepts formels en une passe tout en calculant le support ou en conservant le support dans une forme ou une autre, l'économie en temps d'exécution saute aux yeux. Un tel algorithme sera présenté dans la prochaine section de ce chapitre.

Malgré le fait que le TCF est une représentation condensée, lors de l'extraction des concepts formels des contextes formels de grande taille, il a été noté qu'il peut y avoir explosion combinatoire du nombre de concepts [38] (Besson, 2006). Dès lors, afin de réduire le nombre de concepts extraits, il serait avantageux de choisir ce qui pourrait être intéressant à conserver versus ce qui peut être élagué. Se reportant à la situation de l'analyste de produits qui s'intéresse seulement aux mix de produits ayant un support égal ou supérieur à un seuil de support minimum pour le treillis d'itemset, on pourrait dès lors appliquer le même principe sur le TCF. D'une part, on a une représentation condensée due à la nature même du TCF et d'autre part, on peut ''condenser cette condensation'' afin de ne conserver que les concepts formels intéressants[39].

[38] Ce qui est également vrai pour le treillis d'itemsets fermés.

[39] Il est à noter, bien que non mentionné aux chapitres 2 et 3, que le motif de l'itemset fermé fréquent et son semi-treillis existent et ont le même objectif que les concepts formels fréquent et le semi-treillis des concepts fréquents.

Exemple 5.4

L'analyste de produit ayant examiné le TCF du mix de produits considère qu'il y a trop d'information pour que cette représentation soit utile. Il décide donc de choisir un seuil de support minimal de 2.

❑

Le résultat de l'action de l'analyste produit le semi-treillis de concepts formels aussi appelé l'iceberg de concepts formels[40] illustré à la figure 5.4. Naturellement, il y a peu de changements entre le TCF et l'ICF. Par contre, dans un contexte d'une base de données de grande taille et en choisissant judicieusement le seuil de support, la réduction peut être beaucoup plus considérable que la figure suivante ne laisse supposer.

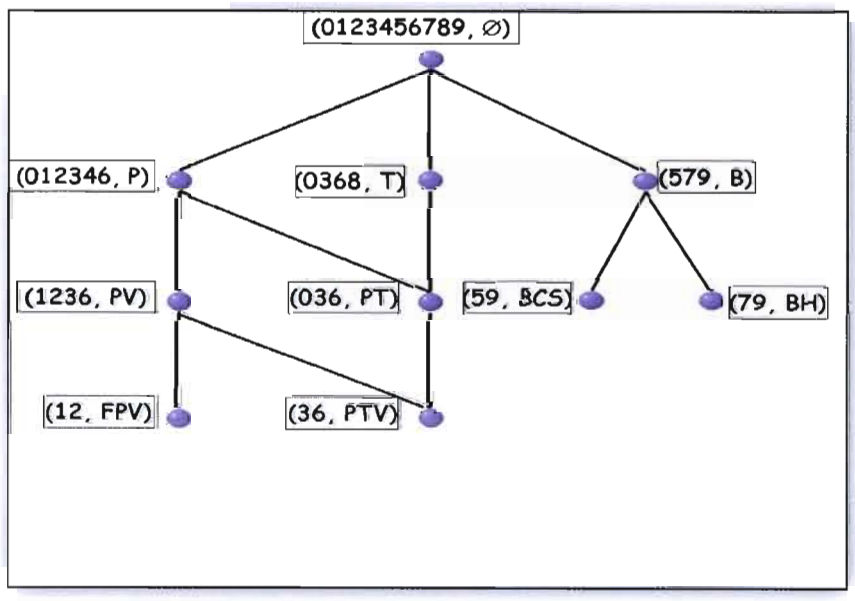


Figure 5.4 – Iceberg de concepts formels

[40] Ci-après ICF.

5.3 Algorithme incrémental d'extraction des concepts formels

Il existe deux types d'algorithmes extrayant les concepts formels : les algorithmes de traitement en lots[41] et les algorithmes incrémentaux. Le premier type extrait du contexte formel les concepts formels. Si le contexte formel est augmenté d'autres transactions, le TCF préalablement extrait doit être calculé à nouveau en fonction du contexte augmenté de ses nouvelles transactions. Le second type calcule également les concepts formels à partir d'un contexte formel. Cependant, si après l'extraction des concepts, le contexte formel est augmenté de nouvelles transactions, on n'a pas besoin de refaire le calcul sur le nouveau contexte formel ; seules les nouvelles transactions seront utilisées pour modifier les concepts formels existants ou, au besoin, créer de nouveaux concepts.

Bien que les algorithmes incrémentaux semblent, apriori, être une meilleure solution que les algorithmes de traitement en lot, chacun a ses avantages. D'une part, si un algorithme de type 'lot' est plus efficace qu'un algorithme incrémental et que l'extraction des concepts est finale[42] alors l'algorithme de type 'lot' sera choisi. Si par contre, des transactions sont constamment ajoutées dans le contexte formel et qu'elles doivent être reflétées dans le TCF alors l'algorithme incrémental est l'outil de choix. Notons, au passage, qu'un survol des plusieurs algorithmes extrayant les concepts formels est donné dans (Carpineto & Romano, 2004).

Dans le cadre de ce mémoire et de ce projet, l'algorithme choisi est l'algorithme incrémental par cardinalité de (Godin, Missaoui, & Alaoui, 1995) et son descendant nommé MagaliceA de la famille des Magalice (Valtchev, Grosser, Roume, & Hacenum, 2003)[43]. Cet algorithme permet l'extraction des concepts formels et de la relation d'ordre. Il prend en

---

[41] De l'anglais : batch

[42] De l'anglais : one shot deal

[43] MagaliceA est un algorithme de la famille Magalice du projet Galicia. Le code de MagaliceA peut être téléchargé du site de sourceforge à l'adresse suivante : http://galicia.cvs.sourceforge.net/galicia/.

compte le seuil de support s'il y a lieu, donc pouvant construire un TCF ou un ICF. Il est à noter que MagaliceA utilise la représentation verticale, ce qui permet l'élagage de certaines transactions du contexte formel si la taille de l'extension, i.e. le support d'un concept à créer ou à mettre à jour, ne satisfait pas le seuil de support minimal choisi. De plus, il peut extraire les générateurs minimaux des concepts[44].

Le tableau suivant présente une version abrégée de l'algorithme incrémental par cardinalité basé sur (Godin, Missaoui, & Alaoui, 1995).

[44] Les générateurs minimaux seront présentés au prochain chapitre de ce mémoire.

Tableau 5.1 – Algorithme MagaliceA

```
▷ PROCÉDURE MagaliceA – calcule le treillis de concepts
▷ Paramètres
▷ L un treillis (E/S)
▷ X l'ensemble des objets associé à l'attribut (E)
▷ a l'attribut (E)
▷ minsup, le support minimum défini par l'usager (E)

PROCÉDURE MagaliceA(L, X, a, minsup)

    (01) compartiments ← ∅
    (02) SI (|X| < minsup) ALORS
    (03)     RETOURNER
    ▷ si le treillis est vide créer le top
    (04) SI (L = ∅) ALORS
    (05)     SI (X = O) ALORS
    (06)         L ← L ∪ (X, a)
    (07)     SINON
    (08)         L ← L ∪ (O, ∅) ∪ (X, a)
    (09)     FIN SI
    (10)     RETOURNER
    (11) FIN SI

    (12) TRIER L sur la taille des intensions en ordre non décroissant
    (13) POUR chaque Concept c de L FAIRE
    (14)     SI c.extension ⊆ X ALORS
    (15)         c.intension ← c.intension ∪ {a} ▷ mise à jour concept existant
    (16)         compartiments ← compartiments ∪ c
    (17)         SI c.extension = X ALORS RETOURNER
    (18)     SINON ▷ créer concept(s)
    (19)         SI (|c.extension ∩ X| ≥ minsup) ALORS
    (20)             intersect ← c.extension ∩ X
    (21)             SI pas de (X1, Y1)∈ compartiments t.q. Y1 = intersect ALORS
    (22)                 L ← L ∪ (intersect, c.intension ∪ {a})
    (23)                 compartiments ← compartiments ∪ (intersect, c.intension ∪ {a})
    (24)                 Calculer les liens
    (25)             FIN SI
    (26)         FIN SI
    (27)     FIN SI
    (28) FIN POUR
FIN MagaliceA
```

La figure suivante présente une trace partielle de l'algorithme MagaliceA basée sur la représentation verticale du contexte formel vu à la section 5.1 de ce chapitre et le seuil de support minimal est 0.

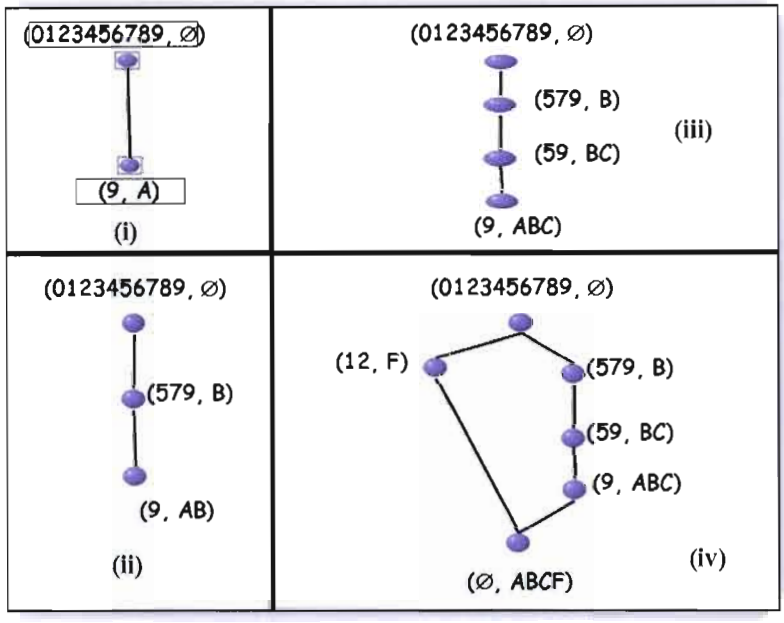


Figure 5.5 – Trace partiel de l'algorithme MagaliceA

Tout d'abord, on insère l'attribut A et son objet associé 9. Puisque le treillis est vide, on crée le concept $\top$ ayant tous les objets comme extension et l'ensemble vide comme intension. Ensuite, le concept (9, A) est créé telle que la figure ci-dessus en (i) l'illustre. Notons au passage que le concept (9, A) fait temporairement office de concept $\perp$. La prochaine entrée est l'attribut B et ses objets 579. Le nouveau concept est inséré soit (579, B) et le concept à la base du treillis est modifié soit (9, AB) tel qu'illustré en (ii). On traite l'attribut C et ses objets associés 59. À partir du concept (579, B) on crée le concept (59, BC). Le concept (9, AB) est ajusté à (9, ABC) tel qu'illustré en (iii). Le dernier exemple de cette trace partielle traite de l'insertion de l'attribut F et de ses objets associés 12. Le concept (12, F) est ajouté au treillis. Par contre, le concept (9, ABC) n'est plus le $\perp$ légitime du treillis car il ne contient pas l'attribut F (dans le pseudo-code l'intersection de 9 et 12 = ∅). Un

nouveau concept $\perp$ est alors créé (∅, ABCF) incluant tous les attributs traités jusqu'ici comme intension et l'extension étant l'ensemble vide tel que montré en (iv). Ceci termine la trace partielle de l'algorithme MagaliceA.

Si le seuil de support minimal avait été autre que 0, il aurait fallu en tenir compte. Prenons, pour le même contexte formel, un seuil de support minimal à 3. La section (i) de la figure 5.5 serait vide car le support de l'attribut A, calculé à partir de ses objets associés est 1. L'attribut A et son objet 9 sont élagués. La figure 5.5 (ii) donnerait le concept $\top$ (0123456789, ∅) et le concept (579, B) lors du traitement de l'attribut B et de ses objets associés 579. La figure 5.5 (iii) et (iv) auraient également les concepts (0123456789, ∅) et (579, B) puisque les supports des attributs C et F sont 2 et donc plus petit que le support minimal choisi. Ce fait est important pour l'algorithme MagaliceA utilisant le support minimal. Puisque l'attribut A possède un support de 1, tout surensemble (intension) contenant A ne peut pas avoir un support plus grand que 1. Il en va de même pour les attributs C et F avec leur support de 2. C'est le principe d'anti-monotonicité discuté au chapitre 2 de ce mémoire et utilisé par APRIORI. MagaliceA utilisant la représentation verticale pour ses entrées, tire avantage de l'anti-monotonicité avec la fonction d'évaluation de support. D'une part, le treillis contient moins de concepts alors la complexité spatiale est réduite. D'autre part, puisqu'il y a élagage des entrées non intéressantes et des concepts non intéressants, la complexité temporelle en est réduite également. Au chapitre 2 de (Carpineto & Romano, 2004), il est dit que la complexité temporelle de l'algorithme incrémental par cardinalité est $\boldsymbol{O}(|\boldsymbol{C}|^2\ |A|^2)$, pour $\boldsymbol{C}$ étant l'ensemble des concepts formels et A celui des attributs. Cependant, il a été noté dans (Godin, Missaoui, & Alaoui, 1995) qu'en pratique[45] la croissance est de l'ordre du linéaire plutôt que de la quadratique[46].

[45] Par de nombreuses expériences empiriques.

[46] Également noté dans (Carpineto & Romano, 2004).

Avant de clore, parlons de la caractérisation de l'iceberg de concepts formels fréquents[47] (et son homologue l'iceberg d'itemsets fermés fréquents) comme représentation condensée. Il a été dit au chapitre 3, que le treillis d'itemsets fermés est une représentation condensée exacte. C'est à dire qu'il n'y a aucune perte d'information. Moyennant une fonction d'inférence, il est possible à partir d'un TCF de recréer, au besoin, une partie ou la totalité des informations que le treillis condense[48]. Dans le cas de l'ICF, bien que les concepts formels soient de par leur nature, des entités condensant exactement l'information qu'elles représentent, l'ICF ne donne que les concepts ayant un seuil de fréquence minimal. Ainsi, l'ICF n'est pas une représentation condensée 0-adéquate. Le degré d'adéquation sera basé sur le seuil de fréquence ou de support choisi. Cependant, il demeure que les intensions des concepts formels représentent des itemsets fermés. Si l'utilisateur désire générer tous les itemsets à partir des intensions des concepts formels d'un iceberg, il sera obligé d'utiliser une fonction d'inférence.

5.4 Discussion

Ce chapitre a présenté les bases de l'AFC et la collection de concepts formels et le TCF. Il a été dit que la collection de concepts formels est une représentation condensée exacte de la collection des itemsets. La section 5.2 fut dédiée à certaines fonctions d'évaluation et à l'ICF. L'avantage de l'ICF tire avantage de l'utilisation de la fonction de support (ou de fréquence) ce qui permet de réduire les complexités temporelle et spatiale et de ne conserver que les concepts intéressants, i.e. satisfaisant le support minimal. La collection des représentations condensées se trouvent enrichie de nouvelles collections pouvant jouer le rôle de représentations condensées ε-adéquate pour la collection intermédiaire (figure ci-dessous).

---

[47] Ci-après ICF.

[48] Dans le cas d'un TCF de grande taille, on n'utiliserait pas la fonction d'inférence pour recréer le treillis complet d'itemsets. La fonction d'inférence serait utile pour recréer des itemsets à partir des concepts jugés intéressants par l'utilisateur.

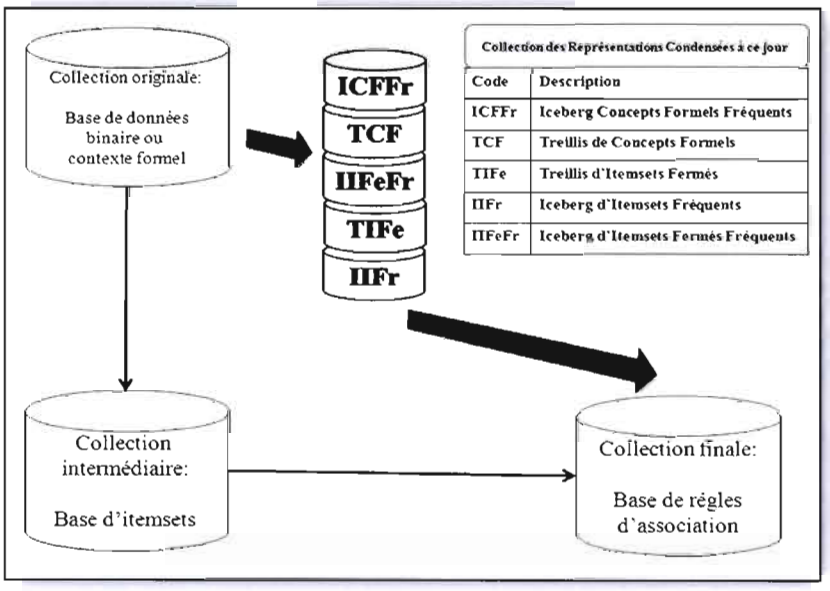


Figure 5.6 – Collection des représentations condensées

L'algorithme présenté dans ce chapitre est MagaliceA qui est un algorithme incrémental. Son implémentation autonome, outre le fait de tenir compte du support minimal et, de créer les concepts formels et la relation d'ordre, elle génère en plus les générateurs minimaux. Ces derniers seront le sujet d'une section du chapitre suivant. La première section du prochain chapitre portera sur la caractérisation des concepts formels dans le treillis.

CHAPITRE VI
CARACTÉRISERATION DES CONCEPTS FORMELS ET GÉNÉRATEURS MINIMAUX

Ce chapitre présente la caractérisation des concepts formels sous extraction utilisant l'algorithme MagaliceA ainsi qu'une autre représentation condensée, les générateurs minimaux.

6.1 Caractérisation des concepts formels sous extraction

6.1.1 Caractérisation informelle

L'algorithme MagaliceA, présenté au chapitre précédent, permet de mettre à jour un TCF sans avoir à recalculer le treillis au complet pour de nouvelles transactions du contexte formel. De plus, il calcule la relation d'ordre, utile pour illustrer le diagramme de Hasse du TCF. On trouve dans (Godin, Missaoui, & Alaoui, 1995) une description de la caractérisation des concepts formels sous extraction. Cette description a servi de préambule à la présentation de l'algorithme incrémental dans le document précédemment cité.

Afin d'illustrer informellement la caractérisation des concepts sous extraction, le jeu de données présenté à l'exemple 5.1 du chapitre précédent sera utilisé. Après la quatrième insertion, le TCF a l'apparence suivante (figure suivante).

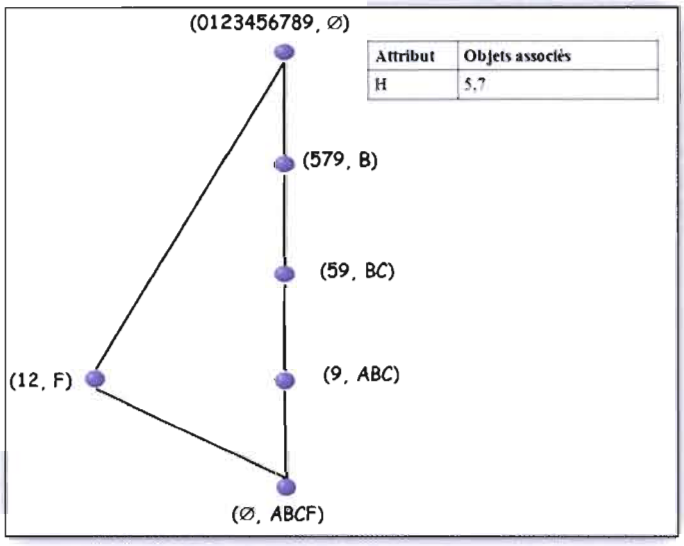


Figure 6.1 – TCF après la 4e insertion

L'attribut H et ses objets associés 5 et 7 sont traités et le résultat de cette insertion est le TCF présenté ci-dessous.

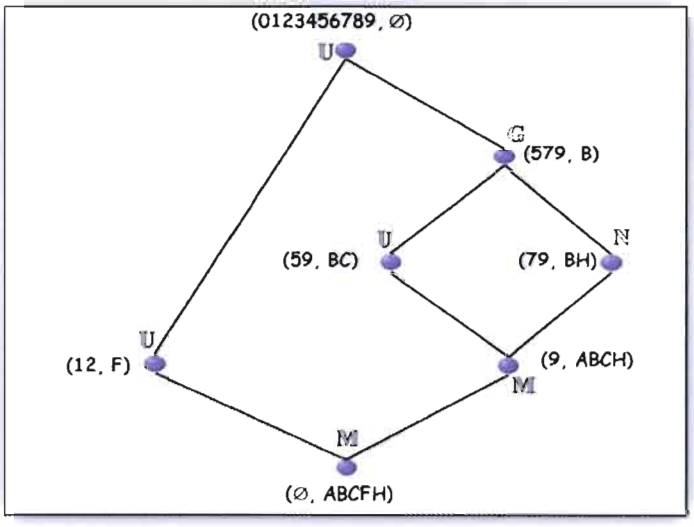


Figure 6.2 – TCF après la 5e insertion

On note les changements suivants. Tout d'abord, à partir du concept (579, B), le concept (79, BH) est créé. Les concepts (9, ABC) et (∅, ABCF) sont modifiés pour obtenir (9, ABCH) et (∅, ABCFH) respectivement. Les autres concepts formels demeurent inchangés. Pour ce qui est de la relation d'ordre, seuls deux liens ont été créés soient, le lien entre (579, B) et (79, BH) et, le lien entre (9, ABCH) et (79, BH).

Après la huitième itération, le TCF a la forme tel que présenté à la figure suivante. Le prochain ajout est le dernier couple de la représentation verticale soient l'attribut V et ses objets associés 1, 2, 3, et 6.

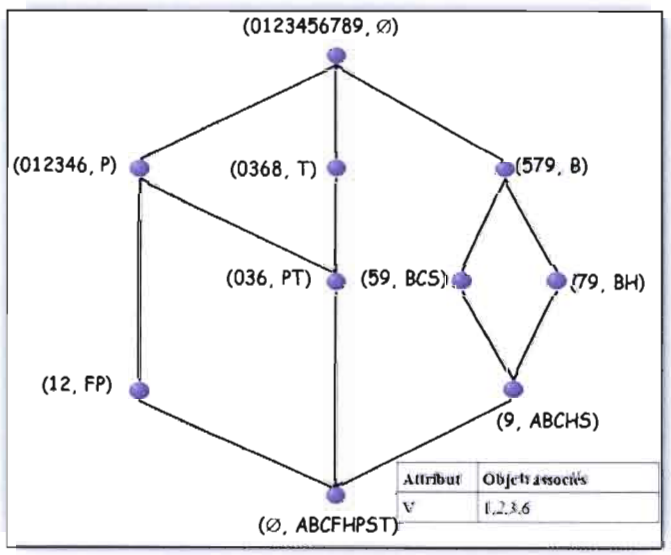


Figure 6.3 – TCF après la 8e itération

Une fois l'insertion terminée, le TCF a la forme telle que présentée à la figure 6.4. On note que le concept (1236, PV) a été créé à partir du concept (012346, P) et que le concept (36, PTV) a été créé à partir du concept (036, PT). Les concepts (12, FP) et (∅, ABCFHPST) ont été modifiés pour obtenir (12, FPV) et (∅, ABCFHPSTV) respectivement. Les liens de précédence ont été ajustés en conséquence.

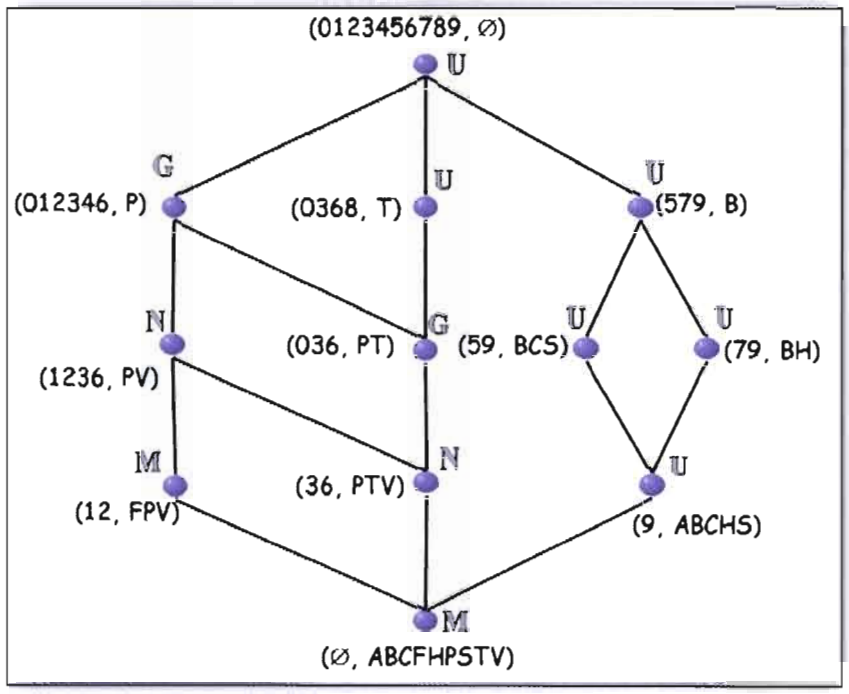


Figure 6.4 – TCF final

Selon le scénario partiel de l'extraction exposé ci-dessus, on trouve des concepts qui demeurent inchangés, d'autres qui sont modifiés, d'autres qui sont nouveaux et enfin, des concepts qui aident à la génération de nouveaux concepts. De ces derniers, on les nomme les géniteurs[49]. Le tableau suivant, tiré de (Godin, Missaoui, & Alaoui, 1995), résume les classes ou catégories de concepts sous extraction.

[49] Autrefois appelés des générateurs de concepts formels.

Tableau 6.1 – Caractérisation des concepts formels sous extraction

Code	Type de concept	Extension	Intension	Parents du concept	Enfants du Concept
𝕌	Concept qui n'a pas été touché par la mise à jour et qui n'est pas un géniteur d'un autre concept[50]	N/A	N/A	N/A	N/A
𝕄	Concept modifié	N/A	On ajoute l'item $a \in A$	N/A	Dans certains cas, des nouveaux concepts peuvent être créés dans la même itération. Dès lors, il faut créer les relations entre le géniteur et le nouveau concept et, le nouveau concept et le concept modifié.
𝔾	Géniteur	N/A	N/A	Ce concept sert à créer un nouveau concept. Il faut supprimer les liens entre le géniteur et un concept déjà existant. On crée le lien entre le géniteur et le nouveau concept.	N/A
ℕ	Nouveau concept	$X \cap a'$	$Y \cup a$	Les parents peuvent être des géniteurs et des nouveaux concepts.	Ils peuvent être des nouveaux concepts et des concepts modifiés.

6.1.2 Caractérisation formelle

De ce qui vient d'être dit, une caractérisation formelle des concepts sous extraction a été proposée. Cette sous-section présente certains éléments de ce formalisme tirés de (Nehmé, Valtchev, Rouane, & Godin, 2005). Ce texte fut choisi puisqu'il couvre la modification et l'insertion de concepts formels utilisant la représentation verticale du contexte formel. L'un des textes fondamentaux de ce formalisme est (Valtchev, Missaoui, Godin, & Meridji, 2002). Il sera utilisé afin de complémenter certaines des notions abordées dans (Nehmé, Valtchev, Rouane, & Godin, 2005).

[50] Le sigle 𝕌 est utilisé et vient de l'anglais unchanged.

Définition 6.1 – Définitions de base

Soient K et K^+ deux contextes formels divergeant d'un seul attribut décrits comme suit : $K = (O, A, I)$ et $K^+ = (O, A^+, I^+)$ avec $A^+ = A \cup \{a\}$ et $I^+ = I \cup \{a\} \times a'$. Pour ce qui est de la représentation graphique du treillis, on suppose $\mathcal{L}$ et $\mathcal{L}^+$ des TCF pour K et K^+.

Exemple 6.1

Contexte K

Objet	A	B	C	F	H	P	S	T	V
0									
1				✖					
2				✖					
3									
4									
5		✖	✖						
6									
7		✖							
8									
9	✖	✖	✖						

(a)

Contexte K^+

Objet	A	B	C	F	H	P	S	T	V
0									
1				✖					
2				✖					
3									
4									
5		✖	✖						
6									
7		✖			✖				
8									
9	✖	✖	✖		✖				

(b)

Figure 6.5 – Contextes K et K^+

Le tableau de gauche de la figure ci-haut représente le contexte K à un moment donné. On ajoute un attribut a et ses objets associés a' à K et K^+ (illustré au tableau de droite) est obtenu. Les TCFs respectifs à K et K^+ sont $\mathcal{L}$ et $\mathcal{L}^+$ illustrés à la figure suivante.

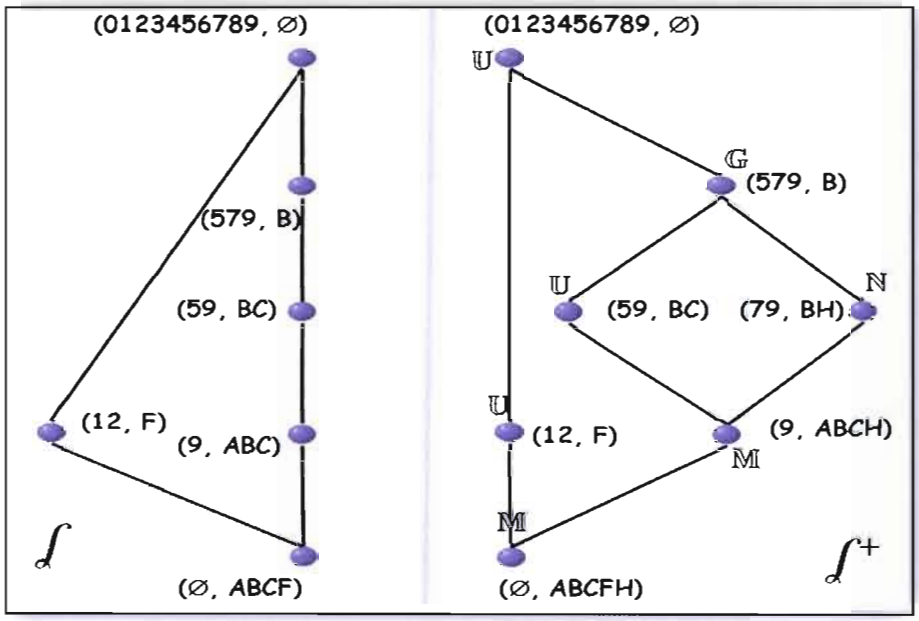


Figure 6.6 – Treillis $\mathcal{L}$ et $\mathcal{L}^+$

❑

Deux fonctions de correspondance entre $\mathcal{L}$ et $\mathcal{L}^+$ sont définies ci-dessous.

Définition 6.2 – Mappage σ

Soient C et C^+ deux collections de concepts formels et leurs TCFs $\mathcal{L}$ et $\mathcal{L}^+$ respectifs. Le mappage σ relie un concept c de $\mathcal{L}$ au concept de $\mathcal{L}^+$ ayant la même extension. Formellement on écrit :

$\sigma : C \rightarrow C^+$ et $\sigma(X, Y) = (X, X')$. Le calcul de X' se fait sur K^+.

Définition 6.3 – Mappage γ

Soient C et C^+ tels que définis précédemment et, $a \in A$ un attribut. Le mappage γ relie chaque concept c de $\mathcal{L}$ vers le concept, dans $\mathcal{L}$, dont la valeur de l'intension est c modulo a. Formellement on peut aussi écrire :

$\gamma : C^+ \rightarrow C$ et $\gamma(X, Y) = (\bar{Y}', \bar{Y})$ où $\bar{Y} = Y - \{a\}$.

Exemple 6.2

Soient K, K^+, $\mathcal{L}$ et $\mathcal{L}^+$ tels que définis dans l'exemple 6.1. Soit a = H et a' = {7, 9}. Les mappages σ et γ sont illustrés par les 2 exemples suivants. La figure 6.7 illustre les deux fonctions de mappage entre les treillis $\mathcal{L}$ et $\mathcal{L}^+$.

(1)On calcule $\sigma(X, Y) = \sigma$ (9, ABC) où (9, ABC) $\in \mathcal{L}$. Afin de trouver le concept dans $\mathcal{L}^+$, il faut calculer (9, 9') où le calcul de 9' se fait dans K^+. Ce qui donne (9, ABCH)

(2)On désire calculer $\gamma(X, Y) = \gamma$(9, ABCH) où (9, ABCH) $\in \mathcal{L}^+$. Afin de trouver le concept dans $\mathcal{L}$, il faut d'abord calculer $\bar{Y}$= Y – {a} soit : ABCH – H = ABC. On a alors ((ABCH - H)', (ABCH - H)) = (ABC', ABC) = (9, ABC).

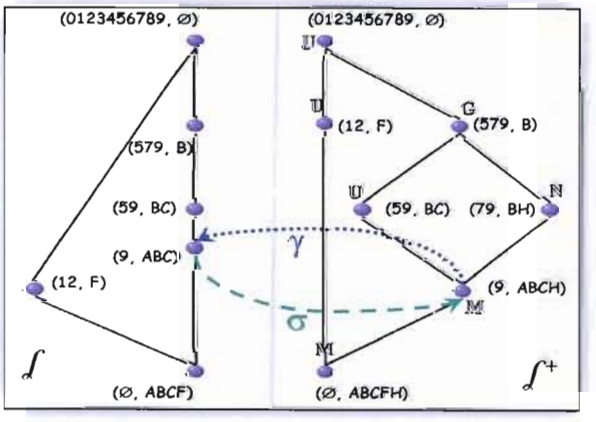


Figure 6.7 – Mappage σ et γ

❑

Les prochaines définitions caractérisent formellement les nouveaux concepts, les concepts modifiées et les concepts géniteurs sous extraction.

Définition 6.4 – Nouveaux concepts dans $\mathcal{L}^+$

Soient C et C^+ et a tels définis dans la Définition 6.3. L'ensemble des nouveaux concepts dans $\mathcal{L}^+$ put se définir comme suit.

$$\mathbb{N}(a) = \{c = (X, Y) \mid c \in C^+; a \in Y; (Y - \{a\})'' = Y - \{a\}\}.$$

Définition 6. 5 – Concepts modifiés dans $\mathcal{L}^+$ et $\mathcal{L}$

Soient C et C^+ et a tels définis dans la Définition 6.3. L'ensemble de concepts modifiés dans $\mathcal{L}^+$ et $\mathcal{L}$ sont respectivement :

$$\mathbb{M}^+(a) = \{c = (X, Y) \mid c \in C^+; a \in Y; (Y - \{a\})' = X\}$$

$$\mathbb{M}(a) = \{c = (X, Y) \mid c \in C; \exists \bar{c} \in \mathbb{M}^+(a), c = \gamma(\bar{c})\}$$

Définition 6.6 – Géniteurs dans $\mathcal{L}^+$ et $\mathcal{L}$

$\mathbb{G}^+(a) = \{c = (X, Y) \mid a \notin Y; (Y \cup \{a\})'' = Y \cup \{a\}\}$

$\mathbb{G}(a) = \{ c = (X, Y) \mid X \not\subseteq \{a\}' ; (Y \cup \{a\})'' = Y \cup \{a\}\}$. Notez que l'expression $(Y \cup \{a\})''$ est calculé dans K^+.

Exemple 6.3

Soient les contextes formels K et K^+ illustrés à la figure 6.5. Soient C et C^+ des collections de concepts formels et leurs TCF respectifs $\mathcal{L}$ et $\mathcal{L}^+$ illustrés à la figure 6.7. Notons que C = {(0123456789, ∅), (579, B), (59, BC), (9, ABC), (12, F), (∅, ABCF)} et C^+ = {(0123456789, ∅), (579, B), (59, BC), (79, BH), (9, ABCH), (12, F), (∅, ABCFH)}. Notons également que $\mathcal{L}$ (et C) représente le TCF (la

collection) avant le traitement par MagaliceA de l'attribut H et ses objets associés notons que $\mathcal{f}^+$ (ou C^+) représente le TCF (la collection) après le traitement par MagaliceA de l'attribut H et ses objets associés.

(1) Nouveaux concepts ($\mathbb{N}$) selon la définition 6.4

Est-ce que le concept formel c = (79, BH) fait parti de l'ensemble $\mathbb{N}$(H) pour $\mathcal{f}^+$? Oui! Notons que l'énoncé $c \in C^+$ est vrai, i.e. que le concept (79, BH) est un concept formel de l'ensemble C^+. L'attribut H est un élément de l'intension BH du concept (79, BH). Finalement le calcul de la fermeture (BH-H)'' produit le même résultat que BH-H soit B.

(2) Concepts modifiés ($\mathbb{M}$ et $\mathbb{M}^+$) selon la définition 6.5

Soit $\mathbb{M}^+$(H) = {(9, ABCH), (∅,ABCFH)}. On trouve que les deux concepts formels de $\mathbb{M}^+$(H) sont des concepts formels de C^+. L'attribut H fait parti de l'intension des 2 concepts. La dernière étape est la vérification de l'énoncé $(Y - \{a\})' = X$. Pour le concept (9, ABCH), le calcul (ABCH-H)' = ABC' produit bel et bien l'extension 9 et pour le concept ((∅,ABCFH), le calcul (ABCFH - H)' = ABCF' produit également l'extension ∅.

Soit $\mathbb{M}$(H) = {(9, ABC), (∅,ABCF)}. On trouve que les deux concepts formels de $\mathbb{M}$(H) sont des concepts formels de C. Il faut maintenant calculer pour chaque $\bar{c} \in \mathbb{M}^+$(H) que le mappage $\gamma(\bar{c})$ produit un concept $c \in C$. Le calcul de γ(9, ABCH) produira le concept (9, ABC) et γ(∅, ABCFH) produira le concept (∅, ABCF). Ces 2 concepts sont effectivement des concepts formels de l'ensemble C et du treillis $\mathcal{f}$.

(3) Concepts géniteurs ($\mathbb{G}$ et $\mathbb{G}^+$) selon la définition 6.6

Est-ce que le concept (579,B) est un concept des ensembles $\mathbb{G}$(H) et $\mathbb{G}^+$(H)? En ce qui a trait à $\mathbb{G}^+$(H), on vérifie tout d'abord que H ne soit pas un élément de l'intension du concept (579, B); ce qui est le cas. Ensuite, on applique l'union de l'attribut H à l'intension de (579, B) c qui donne l'intension BH qui est l'intension du concept (79, BH) le nouveau concept discuté en (1). On peut conclure que (579, B) est un concept géniteur.

Dans le cas de $\mathbb{G}$(H), on vérifie que l'extension du concept géniteur n'est pas un sous-ensemble de H''. Ce qui est le cas, i.e. 579 $\notin$ H'' = 79. Le calcul de $(Y \cup \{a\})''$, i.e. (B $\cup$ {H}) base sur K^+ produit l'intension BH, dès lors le concept (579, B) est un concept de l'ensemble $\mathbb{G}$(H).

❑

Dans (Valtchev, Missaoui, Godin, & Meridji, 2002) on trouve une généralisation de l'intersection de l'attribut a par rapport à l'ensemble des concepts C.

Définition 6.7 – Généralisation de l'intersection et classe d'équivalence

Le mappage $R: C \rightarrow 2^O$ calcule $R(c) = X \cap \{a'\}$. Ce mappage induit une relation d'équivalence sur l'ensemble C et ainsi, une classe d'équivalence sur un concept c, noté $[c]_R$.

Soit $C_{/R}$ l'ensemble des classes d'équivalence de C et, $c_1\ c_2 \in C$. On note la relation d'ordre suivante : $[c_1]_R \leq_{/R} [c_2] \Leftrightarrow R(c_2) \subseteq R(c_1)$.

Une propriété importante est présentée dans (Nehmé, Valtchev, Rouane, & Godin, 2005).

Propriété 6.1 – Classes d'équivalence des nouveaux concepts et des concepts modifiés

Les classes d'équivalence des nouveaux concepts et des concepts modifiés de $\mathcal{L}^+$ sont composées comme suit.

(1) $\forall c \in \mathbb{N}(a)$, $[c]_R^+ = [\gamma(c)]_R \cup c$
(2) $\forall c \in \mathbb{M}^+(a)$, $[c]_R^+ = [\gamma(c)]_R$ (où c est remplacé par $[\gamma(c)]$)

Il serait utile de pouvoir extraire de ces classes d'équivalence, présenté à la définition 6.7, l'élément minimal. Le mappage suivant, noté χ^+, retourne cet élément minimal.

Définition 6.8 – Mappage χ^+

Le mappage χ^+ retourne, pour un concept c de $\mathcal{L}$, l'élément minimal de la classe d'équivalence $[]_R^+$ pour son équivalent $\sigma(c)$ dans $\mathcal{L}^*$. On peut ainsi écrire :

$\chi^+ : C \rightarrow C^+$, $\chi^+(X, Y) = (\bar{X}, \bar{X}^{+\prime})$ où $\bar{X} = X \cap a^{+\prime}$.

Les deux dernières définitions sont importantes pour le prochain sujet soit, les générateurs minimaux extraits par l'algorithme incrémental. Il s'avérera que la notion d'élément minimal (ou concept minimal) sera également importante pour l'extraction des concepts denses dans le projet de ce mémoire.

6.2 Générateurs minimaux

6.2.1 Définitions de base

On retrouve dans la littérature scientifique plusieurs expressions pour désigner les générateurs minimaux, dont celui des ensembles libres (Boulicaut, Bykowski, & Rigotti, 2003). Certaines définitions et exemples de cette sous-section proviennent de la thèse de doctorat de (Jeudy, 2002).

Définition 6. 9 – Relation d'équivalence

Soient R et r tels que définis dans la définition 6.9 et X, $Y \subseteq R$. On dit de X et Y qu'ils sont équivalents dans la BDB r si leur fermeture est la même. On note cette relation $X \sim_{\int} Y$ ou $X \equiv Y$.

Définition 6. 10– Générateur minimal

Soient R un ensemble d'items et r une BDB sur R. Un itemset I est dit un générateur minimal si c'est un élément minimal de la classe d'équivalence de la relation $\sim_f$.

Exemple 6.4

Cet exemple est tiré de (Jeudy, 2002). Soient une ensemble d'items R = {A, B, C, D, E} et r une BDB sur R. Le treillis d'itemsets résultant de r, incluant les supports, est présenté à la figure suivante. Il est à noter que les relations d'ordre ne sont pas exprimées explicitement dans la figure. Les classes d'équivalence sont basées sur la fermeture. On note que les itemsets fermés sont au sommet de leur classe d'équivalence respective. Quant aux générateurs minimaux, ils sont à la base de leur classe d'équivalence. Par exemple, ABCD est l'itemset fermé de sa classe d'équivalence. Dans cette même classe, BD et BC sont les générateurs minimaux. L'itemset A est à la fois un itemset fermé et un générateur minimal puisqu'il est l'unique itemset de sa classe d'équivalence.

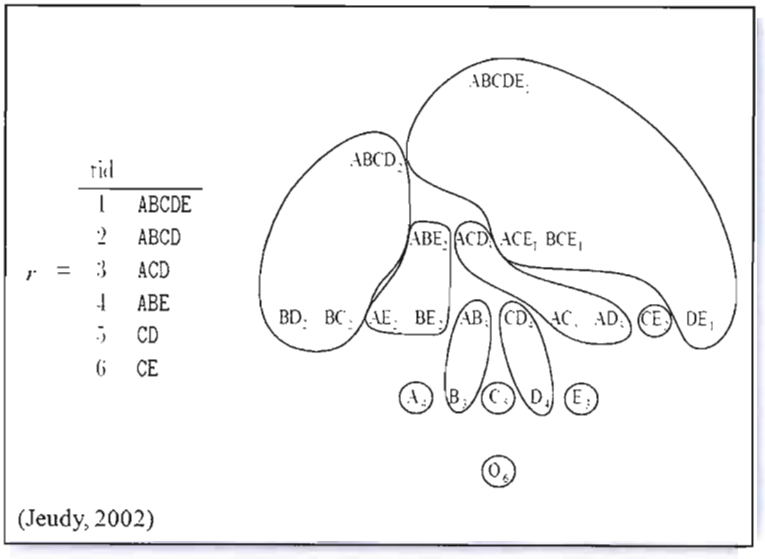


Figure 6.8 – Treillis d'itemsets et classes d'équivalence

❑

Les générateurs minimaux peuvent être définis de plusieurs manières comme en font foi les définitions suivantes. Soient R et r tels que définis précédemment et X, $Y \in R$. Les définitions suivantes sont équivalentes

(1) Un itemset X qu'il est un générateur minimal s'il répond à la Définition 6. 10.
(2) La fréquence de X est différentes de celle de ses sous-ensembles stricts.
(3) X n'est pas inclus dans la fermeture de l'un de ses sous-ensembles stricts.
(4) Soient S, X, $Y \subseteq R$. Il n'existe pas de règles exactes $X \rightarrow Y$ où $Y \neq \varnothing$ t. q. $(X \cup Y) \subseteq S$.

Il n'est pas rare de présenter les générateurs minimaux (voir (Ben Yahia & Nguifo, 2004)) sous la forme d'un treillis tel qu'illustré à la figure suivante.

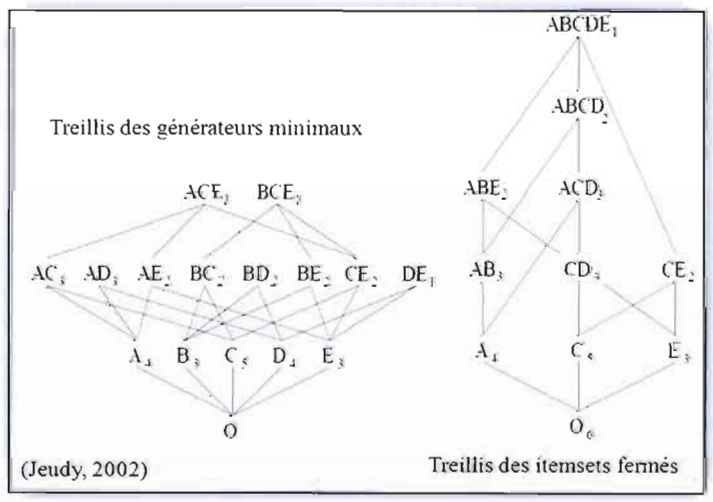


Figure 6.9 – Treillis de générateurs minimaux et d'itemsets fermés

Il est également possible, tout comme l'extraction des itemsets fermés et des concepts formels, que l'extraction de tous les générateurs minimaux d'une BDB soit difficile. On peut alors n'extraire que les générateurs minimaux intéressants. Par exemple, un utilisateur peut choisir de n'extraire que les générateurs minimaux ayant un seuil de fréquence minimal. Ce faisant, un iceberg ou un semi-treillis de générateurs minimaux

fréquents[51] sera produit. Règle générale, on remarque que la collection des générateurs minimaux ne produit pas un treillis complet[52]. Notons, en dernier lieu, que par sa nature la collection des générateurs minimaux est une représentation condensée de la collection des itemsets.

6.2.2 Algorithme d'extraction des générateurs minimaux INCA-GEN

Plusieurs algorithmes d'extraction des générateurs minimaux existent. Parmi ceux-ci, on retrouve Titanic (Stumme, Taouil, Bastide, Pasquier, & Lakhal, 2002) et MIN-EX (Boulicaut, Bykowski, & Rigotti, 2003). La particularité de ce dernier est qu'il extrait non seulement les libres et les libres fréquents mais également les δ-libres et δ-libres fréquents. De façon générale, on dit qu'un itemset X est δ-libre (et fréquent selon le cas) s'il n'existe pas de règles fortes basées sur X dans une BDB. Le δ est un autre paramètre qui permet une condensation plus grande mais avec un degré d'incertitude (autre que le seuil de fréquence minimal) choisi par l'utilisateur. Dès lors, on parle d'une représentation condensée approximative.

Cette sous-section traite de l'algorithme d'extraction des générateurs minimaux à partir d'un TCF sous extraction par MagaliceA présenté dans (Nehmé, Valtchev, Rouane, & Godin, 2005). Cet algorithme, nommé INCA-GEN, est inclus dans l'implémentation autonome MagaliceA. L'originalité de cet algorithme consiste dans l'extraction d'un ou des générateurs minimaux à partir de l'intension du concept formel en construction. Par conséquent, MagaliceA extraie deux représentations condensées soient la collection des concepts formels et la collection des générateurs minimaux. Une définition alternative des générateurs minimaux est présentée ci-dessous.

---

[51] Dans (Jeudy, 2002), on les appelle les libres fréquents.

[52] On peut cependant terminer le treillis en ajoutant un $\top$ au sommet du treillis.

Définition 6.11 – Générateur minimal

Soient K un contexte formel, A l'ensemble des attributs, $Y \subseteq A$ une intension d'un concept formel et $G \subseteq A$ un générateur minimal. On dit de G qu'il est un générateur minimal de l'ensemble fermé Y ssi G est un sous-ensemble minimal de Y t. q. $G'' = Y$.

À cette définition s'ajoute la fonction *gen* associée aux générateurs minimaux.

Définition 6.12 – Fonction *gen*

Soient K, A, Y et G tel que définis précédemment, $F \subset G$, C la collection des concepts formels et $c \in C$. La fonction associant les concepts formels à l'ensemble de ses générateurs minimaux, noté $gen(c) : C \rightarrow 2^A$ est définie comme suit. $gen(Y', Y) = \{G \subseteq Y \mid G'' = Y$ et $\forall F \subset G, F'' \subset Y\}$

L'exemple suivant illustre la définition précédente.

Exemple 6.5

Soient K, A, O tels que définie dans l'exemple 5.1 du chapitre précédent et un seuil de support minimum de 1. Le TCF et les générateurs minimaux sont présentés à la figure suivante.

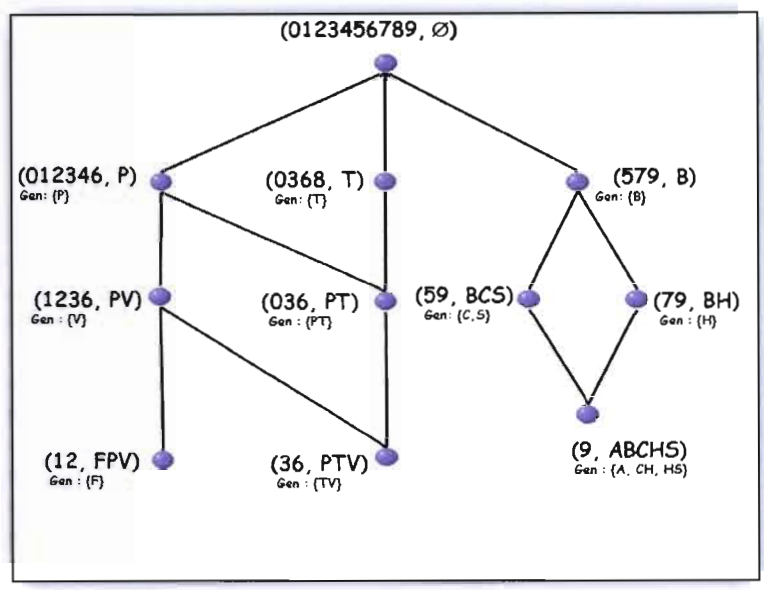


Figure 6.10 Iceberg de concepts formels et générateurs minimaux associés

Selon la figure ci-dessus, le générateur minimal du concept (012346, P) est P. Examinons si P est un générateur minimal de (012346, P) avec la fonction *gen*(012346, P). On a P'' = P et il n'y a pas de sous-ensembles stricts, sauf pour l'ensemble vide, donc P est un générateur minimal et un itemset fermé, i.e., l'intension du concept (012346, P).

Le concept (9, ABCHS) sera maintenant examiné afin d'extraire les générateurs minimaux. La figure 6.10 illustre que de ce concept on peut extraire trois générateurs minimaux soient {A, CH, HS}. Examinons le premier générateur minimal. On note que A'' = ABCHS donc il est un générateur minimal puisque également son seul sous-ensemble strict est l'ensemble vide.

De même on a CH'' = ABCHS. On extrait les deux sous-ensembles stricts de CH soient C et H. On calcule C'' = BCS et BCS $\subset$ ABCHS. On calcule H'' = BH et BH $\subset$ ABCHS. Dès lors, on arrive à la conclusion que l'itemset CH est un générateur minimal de ABCHS. Le principe est le même pour le générateur minimal HS.

❑

Les deux tableaux suivants présentent les algorithmes servant à l'extraction des générateurs minimaux à même le concept formel en construction. Le premier algorithme extrait les classes d'équivalence[53] tel que définies dans la propriété 6.1.

Tableau 6.2 – Algorithme COMPUTE_CLASSES

```
▷ PROCÉDURE COMPUTE-CLASSES
▷ Paramètres
▷ C (E/S) un ensemble de concepts
▷ a (E) un attribut
▷ Note
▷ c (GLOBAL) concept en cours

(01) PROCÉDURE COMPUTE-CLASSES(C, a)

(02)POUR ∀c ∈ C FAIRE
(03)      E ← c.extension ∩ a+,
(04)      θ ← chercher(Classes, E)
(05)      SI (θ = NUL) ALORS
(06)            CRÉER θ
(07)            θ.min-concept ← c
(08)            AJOUTER(Classes, θ, E)
(09)      SI (θ.min-concept < c) ALORS
(10)            θ.min-concept ← c
(11)      θ.min-gen ← MIN(θ.min-gen ∪ gen(c))
(12)FIN POUR
(13)FIN COMPUTE-CLASSES
```

[53] Notons la différence entre les classes d'équivalence dans le treillis d'itemsets et dans le treillis de concepts formel. En effet, les classes d'équivalence pour une collection de motifs quelconques sont des partitions de la collection basées sur un ordre et une évaluation des éléments de la partition. Cet ordre et cette évaluation sont décidés par l'expert du domaine ou par des principes inhérents au domaine.

Tableau 6.3 – Algorithme INCA-GEN

```
▷ PROCÉDURE INCA-GEN
▷ Paramètres
▷ 𝓛=⟨C, ≼⟩ (E/S) un treillis de concepts
▷ a (E) un attribut
▷ Note
▷ c (GLOBAL) le concept courant
(01) PROCÉDURE INCA-GEN(𝓛, a)
(02)      COMPUTE_CLASSES(C, a)
(03)      POUR ∀θ ∈ Classes FAIRE
(04)           c ← θ.min_concept
(05)           SI |R(c)| = c.extension ALORS
(06)                c.intension ← c.intension ∪ {a} ▷ concept existant qui est modifié
(07)           SINON▷ c est géniteur
(08)                ĉ ← nouveauConcept(R(c), c.intension ∪ {a})
(09)                𝓛=𝓛∪ {ĉ}
(10)                miseAjour deOrdre(c, ĉ)
(11)                gen(ĉ) ← ∅
(12)                θ.min_concept ← ĉ
(13)           FIN SINON
(14)           c ← θ.min_concept
(15)           gen(c) ← gen(c) ∪ θ.min-gen {a}
(16)      FIN POUR
(17)FIN INCA-GEN
```

Le deuxième algorithme calcule à partir des classes d'équivalence le ou les générateurs minimaux d'un concept modifié (lignes 06, 14 et 15 du tableau ci-dessus) ou d'un nouveau concept à partir d'un géniteur (ligne 02 – 12 et 14-15 du même tableau). Ces algorithmes tirent avantage du formalisme tel que défini dans la section 6.1.

6.2.2.1 Trace partielle sur l'algorithme INCA-GEN

Soient un ensemble d'objets O, un ensemble d'attributs A et un contexte formel $K = (O, A, I)$ tels que définis à la figure 5.2 du chapitre précédent. Supposons $\int$ une fréquence minimale de 0.20. L'algorithme d'extraction des concepts formels sera MagaliceA augmenté de l'algorithme INCA-GEN pour l'extraction des générateurs minimaux.

La figure 6.11(a) illustre l'ICF après la première itération. Le $\top$ a été créé et le concept #2 (579, B) a été ajouté et son générateur minimal a été calculé soit B. La prochaine

itération traitera la transaction C et ses objets associés 5 et 9. Cette itération illustrera le calcul de la classe d'équivalence, la création d'un nouveau concept et la génération du générateur minimal. Le calcul de la classe d'équivalence produira la collection de classes suivante $\{c_{\#1}, c_{\#2}\}$. Le concept $c_{\#3}$ = (59, BC) est créé à partir de son géniteur, i.e. le concept minimal de la classe. Le concept $c_{\#3}$ est ajouté à la classe d'équivalence (ainsi que dans l'ICF) et devient le nouveau concept minimal de cette classe (ligne 07 à 13 de INCA-GEN). À ce stade, le concept $c_{\#3}$ n'a pas de générateur minimal (ligne 11 de INCA-GEN). À la ligne 15 de l'algorithme INCA-GEN, le générateur minimal C est calculé pour le concept $c_{\#3}$.

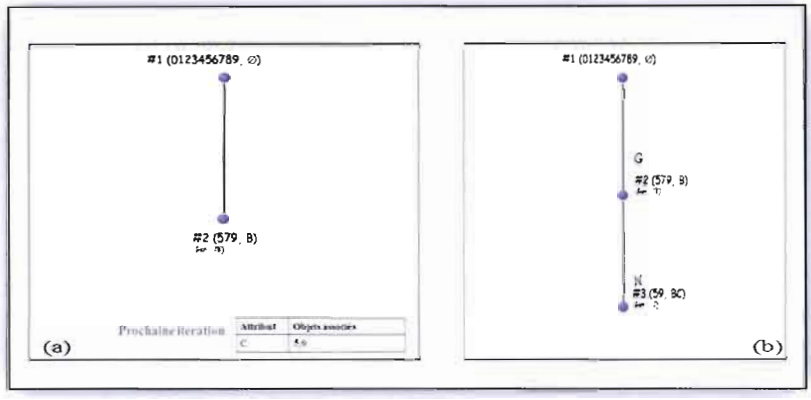


Figure 6.11 – Calcul du générateur pour le concept #3

La figure suivante présente le traitement de l'attribut S et de ses objets associés (figure 6.12 (a)). Le calcul de la classe d'équivalence pour l'attribut S produit la collection de concepts suivante : $\{c_{\#1}, c_{\#2}, c_{\#3}\}$. La transaction (S, 59) ne produira pas de nouveau concept mais le concept $c_{\#3}$ sera mise à jour et le concept minimal de la classe sera le concept $c_{\#3}$ = (59, BC). Ce dernier sera mis à jour, i.e. (59, BCS) selon les lignes 05 à 07 de INCA-GEN. À la ligne 15 le générateur minimal S sera calculé et ajouté à la collection de générateurs minimaux du concept $c_{\#3}$.

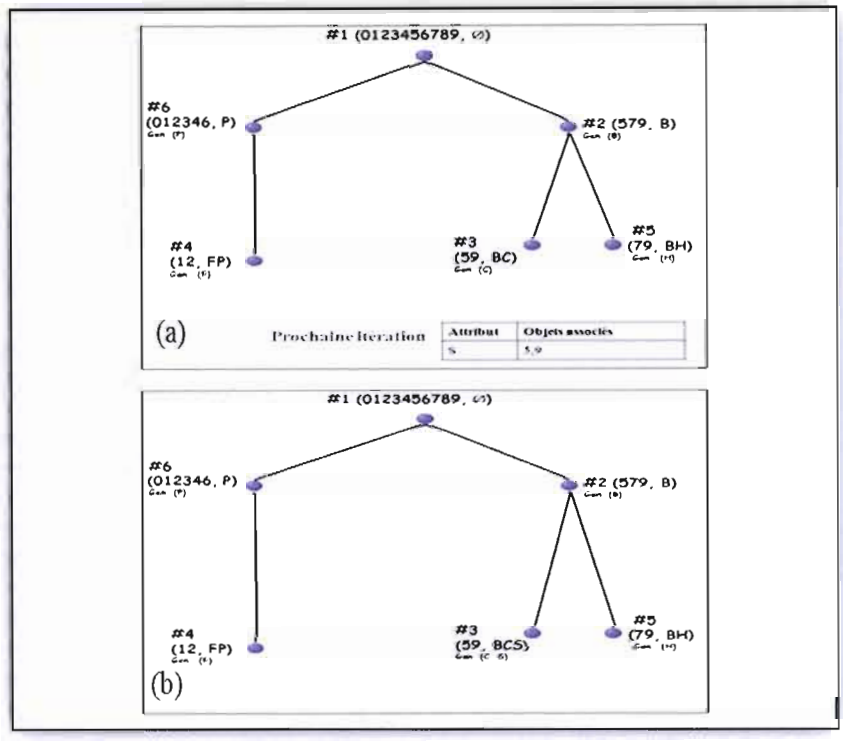


Figure 6.12 – Traitement de l'attribut S

Ceci termine la trace partielle de l'algorithme INCA_GEN.

6.3 De l'utilité de produire le TCF avec la collection de générateurs minimaux

L'algorithme MagaliceA augmenté de l'algorithme INCA-GEN permet de calculer la collection de concepts formels ainsi que la collection de générateurs minimaux. Mais, en quoi l'extraction de deux représentations condensées peut-elle être utile ? Revenons un moment sur les règles (exactes) d'association que nous avons extraites dans le chapitre II de ce mémoire (figure 2.8).

À la figure 2.8, il y a 3 règles exactes pour l'itemset FPV. Ces règles n'ajoutent aucune information pertinente. En revanche, elles ajoutent à la taille de la base de règles. On remarque à la figure 6.10 que le générateur minimal de FPV est F. Le générateur minimal

peut être utilisé comme antécédent de la règle exacte sans l'extraire de l'intension du concept formel. Pour le conséquent, on retire F de l'intension FPV. On obtient donc la règle F $\Rightarrow$ PV. On se limitera donc à une seule règle plutôt que trois. Le lecteur intéressé par ce sujet est invité à consulter le chapitre 19 de (Godin, 2006) sur les règles informatives maximales.

6.4 Discussion

Dans ce chapitre nous avons examiné la caractérisation des concepts formels sous extraction ainsi qu'un nouveau motif soit le générateur minimal (ensemble libre).

La collection de représentations condensées, qui sont des alternatives à la collection complète (ou treillis complet) d'itemsets, est augmentée de nouvelles collections, tel qu'en fait foi le tableau 6.4. Peut-on penser qu'une de ces collections est meilleure que les autres ? Dans les faits, cette question est sans réponse car, il en va d'une part de la complexité temporelle et spatiale des représentations et d'autre part, de l'utilité précise d'une représentation par rapport à une autre et, comme mentionné au chapitre 1 de ce mémoire, du type de problème à résoudre. Par exemple, si l'utilisateur désire des itemsets fermés sans le support, le treillis d'itemsets fermés est une bonne solution. Par contre, si la valeur du support es importante pour l'utilisateur, alors le TCF, de part la structure du concept formel, possède intrinsèquement le support calculable rapidement à même la cardinalité de l'extension associé à son intension sans relecture du contexte formel[54].

[54] Notons que ce commentaire n'est pas vrai dans le cas de l'ICF. Ce dernier étant un sous-treillis, il n'est pas une représentation condensée de tous les itemsets possibles d'une BDB.

Tableau 6.4 – Représentations condensées, alternatives au treillis d'itemsets

Représentation sur une dimension – Itemsets		
Acronyme	**Représentation**	**Exacte/ Approximative**
TIFe	Treillis d'itemsets fermés	E
IIFr	Iceberg d'itemsets fréquents	A
IIFeFr	Iceberg d'itemsets fermés et fréquents	A
Représentation sur deux dimensions – Concepts formels		
Acronyme	**Représentation**	**Exacte/ Approximative**
TCF	Treillis de concepts formels	E
ICF	Iceberg de concepts formels fréquents	A
Représentation sur une dimension – Générateurs minimaux		
Acronyme	**Représentation**	**Exacte/ Approximative**
TGM	Treillis de générateurs minimaux	E
IGMFr	Iceberg de générateurs minimaux fréquents	A
δ-libre	Collection des δ-libres	A
δ-libre-Fréquents	Collection des δ-libres et fréquents	A

Ceci étant dit, il serait possible de matérialiser plus d'une représentation condensée et de les utiliser en fonction des besoins de l'usager et des forces ou lacunes desdites représentations. Par exemple, supposons que l'iceberg des itemsets fréquents et le treillis des itemsets fermés sont matérialisés. Il peut arriver qu'une application ait besoin de l'information des itemsets fermés fréquents. Une solution serait de les extraire de la collection initiale soit la BDB ou encore de créer une vue logique[55] sur l'IIFr et le TIFe, autrement dit, IIFeFr = IIFr ∩ TIFe.

Puisque le sujet de la matérialisation des représentations condensées a été avancé continuons dans ce schème de pensée. À supposer qu'à partir d'un contexte formel de très grande taille, le TCF/TGM a été extrait et matérialisé. Cette matérialisation permet alors aux usagers d'extraire la collection des concepts fréquents ou des itemsets fermés fréquents

[55] À condition que la taille (en mémoire centrale) le permet.

(intension d'un concept) pour un seuil de fréquence minimal. Ou alors, on peut faire une requête pour ne considérer que les générateurs minimaux ou seulement les générateurs minimaux fréquents. Ces requêtes faites à partir du TCF/TGM peuvent être matérialisées à leur tour ou non. En supposant leur matérialisation, on peut les entreposer dans des "magasins de données[56]".

Somme toute, il y a des avantages pour les usagers tout comme les développeurs d'applications à avoir sous la main le choix entre plusieurs représentations condensées pour une collection de motifs particuliers.

Le but de la présentation de la caractérisation des concepts formels et de la collection de générateurs minimaux est que le projet de ce mémoire, l'extraction des concepts denses, pourrait bien avoir un impact sur la caractérisation des concepts. Ou encore, il est possible que la caractérisation des concepts permette de trouver un moyen efficace pour calculer la collection des concepts denses. Il faut ajouter que l'extraction des concepts denses pourrait avoir des conséquences sur l'extraction des générateurs minimaux tels qu'ils sont présentés dans ce chapitre. Le chapitre traitant des concepts denses y reviendra. Le sujet du prochain chapitre portera sur la définition du problème menant à une solution d'extraction des concepts denses suivant des principes de l'AFC ainsi qu'une présentation l'état de l'art dans le domaine des représentations condensées sur deux dimensions.

[56] Entrepôt de données spécialisé généralement relié à un secteur d'activité particulier d'une organisation. De l'anglais : datamart.

CHAPITRE VII
ÉTAT DE L'ART

La première section de ce chapitre présente le problème relatif à l'extraction de la collection de concepts formels et illustre une solution possible. La seconde section passe en revue certaines propositions en ce qui a trait à l'extraction des concepts avec exceptions bornés par des seuils choisis par un utilisateur.

7.1 Description du problème

Les chapitres 5 et 6 de ce mémoire ont identifié certains problèmes relatifs à l'extraction ou l'utilisation de la collection des concepts formels. Résumons ces problèmes ci-dessous.

- Il peut arriver que l'extraction de la collection de concepts formels ou de concepts formels fréquents soit difficile.
- Le contexte formel peut contenir des erreurs, i.e., des données bruitées. L'extraction à partir d'un tel contexte produit un nombre indu de concepts formels.
- Un trop grand nombre de concepts pourrait ne pas servir les besoins des usagers.
- L'ICF, tout en ayant moins de concepts que le TCF, pourrait ne pas servir les besoins des usagers.
- La précision des concepts formels est trop fine pour être utile. Un concept ayant des exceptions mais présentant une certaine régularité pourrait être plus utile.

Par conséquent, il est clair qu'une relaxation du concept formel serait utile, i.e. l'acceptation d'exceptions dans le concept formel. Ce dernier ne serait plus formel par définition. Nous avons emprunté l'expression *concept dense* de (Besson, Robardet, & Boulicaut, 2006)[57] qui exprime bien l'idée. C'est un concept qui contient plus de '1' que de '0'. Le nombre d'exceptions devrait être borné par un seuil choisi par l'utilisateur. L'exemple suivant exprime l'idée. Le jeu de données de cet exemple provient du chapitre 19 de (Godin, 2006).

Exemple 7.1

Soient O l'ensemble des objets tel qu'illustré à la figure 7.1(a), A l'ensemble des attributs (figure 7.1(b)) et K le contexte formel (figure 7.1(c)).

Objet	Description
0	Transaction 0
1	Transaction 1
2	Transaction 2
3	Transaction 3
4	Transaction 4
5	Transaction 5
6	Transaction 6
7	Transaction 7
8	Transaction 8
9	Transaction 9

(a)

Attribut	Description
A	Antiacide
B	Bière
C	Croustille
F	Fromage
H	Huitres
P	Pain
S	Salsa
T	Pâté
V	Vin

(b)

	Attribut							
Objet	F	P	V	T	C	S	B	H	A
1	✖	✖	✖						
2	✖	✖	✖						
3		✖	✖	✖					
6		✖	✖	✖					
0		✖		✖					
4		✖							
8				✖					
9					✖	✖	✖	✖	✖
5					✖	✖	✖		
7							✖	✖	

(c)

Figure 7.1 – Exemple d'un contexte formel bruité

Ce contexte formel contient douze concepts formels[58], incluant les concepts $\top$ et $\bot$. Si des exceptions sur les objets et les attributs sont acceptées le nombre de concepts pourrait réduire

[57] Dans les faits, les auteurs appellent leur motif, un bi-ensemble dense et pertinent. Un bi-ensemble formel est un concept formel. Dès lors, un bi-ensemble dense sera pour nous un concept dense.
[58] Voir chapitre 5 de ce mémoire.

considérablement. La figure 7.1(c) illustre ce fait. On trouve quatre concepts dont deux denses, (0123468, FPTV) et (579, ABCHS) et, $\top$ et $\bot$. La différence est illustrée dans les treillis ci-dessous.

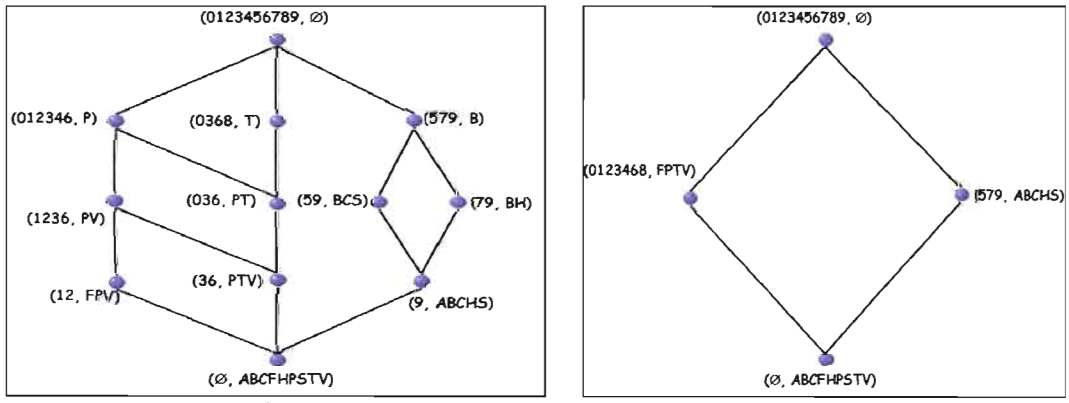


Figure 7.2 – Treillis de concepts formels (a) et treillis de concepts denses (b)

❑

L'objectif du projet de ce mémoire est d'extraire les concepts denses, à partir d'un contexte formel, en utilisant des seuils d'exception sur les lignes (objets) et les colonnes (attributs) choisis par l'utilisateur.

7.2 État de l'art

La plupart des rubriques de cet état de l'art ont été colligées dans l'état de l'art présenté dans (Besson, 2005) et (Besson, Robardet, & Boulicaut, 2006). Les auteurs desdits textes ont remarqué le peu de travaux sur les concepts denses, i.e. les concepts formels avec erreurs. Dès lors, ils ont ajouté certains travaux de condensation avec exceptions qui se basent sur une seule dimension de la matrice binaire.

7.2.1 Les motifs ETI

Dans (Yang, Fayyad, & Bradley, 2001), les auteurs présentent un "nouveau" type de motif appelé ETI[59] qui est une généralisation des itemsets fréquents avec exceptions bornées.

Définition 7.1 - ETI

Soient I un ensemble d'items, *bd* une BDB ayant n transactions basée sur I, $E \subseteq I$ un itemset, un seuil de fréquence minimal σ et un nombre d'erreurs ε. On dit que E est un ETI ayant au plus ε erreurs et un seuil minimal de fréquence σ dans *bd* s'il existe σn transactions dans lesquelles il y a au moins $1 - \varepsilon$ items présents.

Les auteurs continuent en établissant deux classes de ETI soient : les faibles et les fortes. La différence réside dans la probabilité d'apparition des '1' dans un ensemble de transactions. Ils ont proposé un algorithme dit exhaustif pouvant extraire tous les ETI. Cependant sa complexité temporelle est de l'ordre de l'exponentiel. Les auteurs se sont rendus compte qu'en utilisant d'une part des heuristiques et d'autre part un algorithme glouton, on peut abaisser la complexité temporelle. L'envers de la médaille est que la collection extraite est incomplète.

Exemple 7.2 (Yang, Fayyad, & Bradley, 2001)

Soient $I = \{P_1, P_2, P_3, P_4, P_5, \ldots\}$ un ensemble d'items, *bd* une BDB sur I ayant $n = 10000$ transactions, un seuil de fréquence minimal σ de 5.7% et un seuil d'exception maximal ε de 20%. Soit $E \subseteq I$ et $E = \{P_1, P_2, P_3, P_4, P_5\}$. E est un ETI car il a une fréquence de 5.7% (570/10000) et au maximum 20% d'exceptions tel qu'illustré au tableau suivant.

[59] ETI : Error-Tolerant Itemset. Une traduction en français serait itemset tolérant aux fautes.

Nombre de transactions	P_1	P_2	P_3	P_4	P_5	...
100	1	1	1	1	0	...
100	0	1	1	1	1	...
90	1	1	1	0	1	...
80	1	0	1	1	1	...
200	1	1	0	1	1	...
...	...	...	...	...	...	...

❑

7.2.2 Les itemsets denses et les motifs FT

Dans (Seppänen & Mannila, 2004), les auteurs se sont intéressés aux itemsets à densité faible et ont présentés l'algorithme DENSE_SETS. Ce dernier extrait les itemsets à densité faible pour ensuite les passer à un algorithme de générateur de candidats de type APRIORI. DENSE_SETS requiert deux paramètres soient le seuil de fréquence minimal et le seuil de densité faible. Les auteurs font remarquer la difficulté de choisir les deux seuils de façon optimale. Ils proposent donc l'algorithme TOP-k qui extrait un sous-ensemble des k-itemsets d'une BDB avec un seuil désiré. Ce sous-ensemble n'est, bien entendu, qu'une approximation permettant d'évaluer l'extraction par densité faible. Une fois les seuils trouvés grâce à ce sous-ensemble, on peut les utiliser pour extraire les itemsets denses de la collection complète. Les auteurs de (Besson, Robardet, & Boulicaut, 2006) constatent cependant qu'il est difficile d'appliquer les notions proposées basées sur des ensembles à une dimension aux concepts formels qui sont une représentation sur deux dimensions. La définition du calcul de la densité faible ainsi qu'un exemple l'illustrant sont donnés ci-après.

Définition 7.2 – Densité faible

Soient R un ensemble d'items, r une BDB basée sur R et $X \subseteq R$. On définit la densité faible comme suit.

$wdens(X,r) = \frac{\sum_{t \in r} |X \cap t|}{|X| * |r|}$, autrement dit le nombre des 1 divisé par le nombre de case.

Exemple 7.3 (Seppänen & Mannila, 2004)

Soient R = {A, B, C, D} un ensemble d'items, r une BDB sur R tel qu'illustré au tableau suivant et $X \subseteq R$ et X = {ABCD}. Le calcul de la densité faible de X est $wdens(X, r)$ = 12/24 = 0.5.

A	B	C	D
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	1

❑

L'article précédent se base sur la densité relative. On retrouve dans (Pei, Tung, & Han, 2001) un motif semblable au précédent soit le motif FT et son homologue fréquent, le motif fréquent FT[60]. La différence est l'utilisation de la densité absolue plutôt que relative. Notons que le calcul de la densité absolue semble plus simple à appliquer dans le contexte des concepts formels que le calcul de la densité relative[61]. La définition d'un motif FT est donnée ci-dessous. On remarquera la simplicité du calcul.

Définition 7.3 – Motif FT

Soient R un ensemble d'items, r une BDB basée sur R, $X \in r$ une transaction de r, i.e. un itemset de r, δ un seuil de densité absolue et P un itemset. On dit de P qu'il est un motif FT si $|P \cap X| \geq (|P| - \delta)$.

[60] De l'anglais : Fault-tolerant pattern et fault-tolerant frequent pattern.

[61] Cependant le calcul de la densité absolue a ses revers comme en fera état le chapitre 9 et 10 de ce mémoire

7.2.3 Les tuiles géométriques et combinatoires

Dans (Gionis, Mannila, & Seppänen, 2004), on présente un motif qui prend en compte les deux dimensions d'une BDB ou d'un contexte formel. Les auteurs commencent avec le problème des tuiles géométriques. Ce problème consiste à trouver des rectangles maximaux avec exceptions dans un contexte formel. Autrement dit, il existe des transactions dont les objets et les attributs sont déjà dans un ordre approprié facilitant l'extraction des rectangles maximaux avec exceptions. Les auteurs ont implémenté et testé quatre approches.

- Approche descendante : à chaque étape descendante, une tuile est sélectionnée et est considérée come une tuile pour ce niveau ou est fusionnée avec une tuile déjà existante si les critères de fusion sont satisfaits.
- Approche ascendante : elle suit les mêmes principes que la précédente mais dans l'autre sens.
- Approche mixte : une tuile est considérée géométrique si elle ne recouvre pas une autre tuile.
- Approche à niveau unique : seules les tuiles du niveau 0 sont considérées.

Bien sûr, on ne s'attend pas à ce que le contexte formel soit ordonné de telle sorte que les tuiles soient prêtes à la cueillette. Dans la deuxième section de (Gionis, Mannila, & Seppänen, 2004), les auteurs abordent le sujet des tuiles combinatoires. L'idée est de réordonner les lignes et les colonnes afin d'obtenir des tuiles consécutives. Ainsi l'extraction des tuiles géométriques s'en trouve facilitée. Trouver un tel ordre peut être difficile puisqu'un contexte formel peut avoir un nombre combinatoire de tuiles. En dépit de cette observation, les auteurs argumentent qu'il est possible de trouver un tel ordre par une approche "spectrale" d'un contexte formel ou d'une BDB, d'où leur algorithme spectral.

Dans (Besson, Robardet, & Boulicaut, 2006) on argumente que ce ne sont pas tous les contextes formels qui possèdent des tuiles enchâssées "naturellement". De plus, ils notent

que l'algorithme d'optimisation locale n'est pas déterministe et que la qualité globale des tuiles peut en être affectée.

7.2.4 Réduction du treillis de concepts formels

Cette sous-section examine deux propositions dont le but est de réduire la taille du treillis de concepts afin de le rendre plus lisible et possiblement plus utile. Les auteurs des deux propositions argumentent que l'ICF, bien qu'utile, n'est pas toujours adéquat. L'ICF basé sur le seuil de fréquence minimal ne présente que les concepts de la partie supérieure du treillis. Cependant, il peut exister des concepts infréquents dignes d'intérêt[62] et qui ne sont pas présents dans l'ICF.

7.2.4.1 Treillis Alpha Galois

Les treillis Alpha Galois sont présentés dans (Ventos & Soldano, 2005). L'idée générale consiste à réduire le nombre de concepts du treillis en employant un critère local de fréquence et un partitionnement. L'exemple suivant illustre le principe du partitionnement.

Exemple 7.4 (Ventos & Soldano, 2005)

Soit K le contexte formel suivant.

	T1	T2	T3	A3	A4	A5	A6	A7	A8
I1	✖			✖	✖		✖		✖
I2	✖			✖		✖	✖		
I3		✖			✖		✖		✖
I4		✖			✖		✖	✖	
I5		✖		✖			✖		✖
I6			✖	✖			✖		✖
I7			✖	✖			✖		✖
I8			✖	✖		✖	✖		✖

Supposons que les attributs T1, T2 et T3 expriment chacun un type ou une catégorie d'objets. Prises ensembles, ces trois catégories couvrent tous les objets de K. On peut alors créer des classes de

[62] Certains appellent ces concepts, des concepts émergents.

base sur les objets soient : BC1 = {I1, I2}, BC2 = {I3, I4, I5} et BC3 = {I6, I7, I8} dont les intensions sont respectivement, BC1' = {T1, A3, A6}, BC2' = {T2, A6} et BC3' = {T3, A3, A6, A8}.

❑

Définition 7.4 – Satisfaction α

Soient O un ensemble d'objets, A un ensemble d'attributs, $K = (O, A, I)$ un contexte formel et $\alpha = [0, 100]$. Supposons $e \subseteq O$ l'une des classes de base et $t \subseteq A$ un terme. On définit que e α-satisfait t comme suit : $e\ \mathrm{sat}_\alpha\ t$ ssi $|t' \cap e| \geq (|e| \times \alpha)/100$.

Donc, une classe de base e satisfait un terme t pour un pourcentage α défini par un utilisateur si la taille de l'intersection de l'extension du terme t et de la classe de base e est plus grande ou égale la taille de la classe e multipliée par $\alpha/100$.

Puisque le degré de satisfaction α est défini en fonction des objets d'un terme (i.e. t') et d'une classe de base, on utilise donc ce degré pour vérifier si il y a au moins α% des objets d'une classe de base qui satisfait un terme. En ajoutant une contrainte de type *est un*[63], on retrouve l'appartenance classique[64] entre les classes d'équivalence et les termes.

Exemple 7.5 (Ventos & Soldano, 2005)

Soit K le contexte formel défini dans l'exemple précédent. Supposons t = {A6, A8}. Donc t' = {I1, I3, I5, I6, I7, I8}. Est-ce que la classe de base BC2 satisfait avec $\alpha = 60$ le terme t ou autrement dit est que BC2 sat_{60} t est vrai ? $|BC2 \cap t'| = 2$ et $(|BC2| * 60)/100 = 1.8$. Puisque $2 \geq 1.8$ alors BC2 sat_{60} t est vrai.

❑

[63] La contrainte *est un* est la traduction de l'anglais *is a*.

[64] De l'anglais classical membership.

Une caractéristique appréciable du treillis Alpha Galois est que si $\alpha = 0$, l'algorithme extrait tous les concepts formels. Pour des valeurs plus élevées, les concepts auront une granularité moins fine que les concepts formels. Naturellement, ceci veut dire qu'il y a une certaine fusion des concepts en fonction des classes de base. Dans (Besson, Robardet, & Boulicaut, 2006) on déplore cependant que seule une dimension est utilisée alors que dans l'AFC on est habitué à la dualité des fonctions et des mappages. On peut également ajouter que le partitionnement en classes de base d'un contexte formel n'est pas toujours possible. On ne peut prendre en compte que tout contexte formel possède intrinsèquement des partitions "naturelles".

7.2.4.2 Concepts stables

Dans (Kuznetsov, Obiedkov, & Roth, 2007), les auteurs présentent les concepts stables. L'idée est de réduire le nombre de concepts dans le treillis afin d'obtenir une meilleure lisibilité. Le principe de base est de définir un indice de stabilité sur un concept. Si l'indice de stabilité rencontre le seuil minimum défini par un utilisateur, ce concept sera conservé (ou affiché dans le treillis). On définit deux types d'indice de stabilité soient l'un pour les extensions et l'autre pour les intensions d'un concept.

Définition 7.5 – Stabilité intensionnelle

Soient O un ensemble d'objets, A un ensemble d'attributs, $K = (O, A, I)$ un contexte formel, (X, Y) un concept formel de K et, $C \subseteq X$ et $C' = Y$. On définit l'indice de stabilité intensionnel, noté σ, comme suit : $\sigma(X, Y) = \frac{|\{C \subseteq X \mid C' = Y\}|}{2^{|X|}}$.

En d'autres termes, $\sigma(X, Y)$ est égale au nombre de sous-ensembles de X ayant la même intension sur le nombre total de sous-ensembles possibles sur X. Notons que la

définition de l'indice de stabilité extensionnelle est la duale de la définition ci-dessus. De la définition donnée ci-haut, les auteurs énoncent la proposition suivante.

Proposition 7.1

Soient O un ensemble d'objets, A un ensemble d'attributs, $K = (O, A, I)$ un contexte formel, (X, Y) un concept formel de K. Pour un ensemble $H \subseteq O$, soient $I_H = I \cap (H \times A)$ et $K_H = (H, A, I_H)$. On a donc $\sigma(X, Y) = \frac{|\{K_H \mid H \subseteq O \text{ et } Y = Y^I{}^H{}^I{}^H\}|}{2^{|O|}}$.

En d'autre mots, l'indice de stabilité d'un concept est la probabilité d'une intension Y si tous les sous-contextes de K sur A sont équiprobables. La stabilité indique à quel point l'intension d'un concept est dépendante de certains objets de l'extension du même concept. Une intension stable est moins sensible au bruit sur l'extension. De plus, l'extension d'un concept stable est "moins près" des extensions des concepts qui précèdent le concept.

On trouve dans (Roth, Obiedkov, & Kourie, 2008) l'algorithme qui calcule les concepts stables. Cependant, ce calcul est un problème #P-complet. Afin de faciliter le calcul (Kuznetsov, Obiedkov, & Roth, 2007) proposent des heuristiques simples. Dans (Jay, Kohler, & Napoli, 2008) les auteurs présentent une application utilisant le principe des treillis de concepts stables.

Il semble qu'il soit difficile d'utiliser à la fois la stabilité intensionnelle et la stabilité extensionnelle. Bien que ces deux principes sont duaux l'un de l'autre, la complexité temporelle de leur amalgame est un problème #P-complet. Dans le champ d'activité où ce type de treillis est utilisé, les auteurs de (Roth, Obiedkov, & Kourie, 2008) notent qu'il n'est pas nécessaire d'avoir les deux types de stabilité. Il est à noter également que la réduction du treillis de concepts formels en treillis de concepts stables par l'algorithme présenté dans (Roth, Obiedkov, & Kourie, 2008) prend comme entrée le TCF. Si le TCF est difficile à extraire, il faut donc prendre en compte le temps d'exécution de l'extraction du TCF et le

temps d'exécution du treillis de concepts stables. Dans le cas où il est impossible d'extraire le TCF, il est donc impossible de produire un treillis de concepts stables.

7.2.5 Les motifs bi-ensembles

Cette sous-section présente deux nouveaux types de motifs soient : les δ-bi-ensembles et les bi-ensembles à proprement parler.

7.2.5.1 Les δ-bi-ensembles

Dans (Pensa & Boulicaut, 2005) les auteurs présentent les δ-bi-ensembles. Ces derniers sont basés (et calculés) sur les δ-libres et δ-fermés tels que trouvés dans (Boulicaut, Bykowski, & Rigotti, 2003). La définition des δ-bi-ensembles est énoncée ci-dessous.

Définition 7.6 – δ-bi-ensemble

Soient O un ensemble d'objets, A un ensemble d'attributs, $K = (O, A, I)$ un contexte formel et δ un entier. Soient h_δ une fonction qui calcule le δ-fermé et ψ la fonction qui calcule à partir d'une intention l'extension d'un concept. Un δ-bi-ensemble (T, G) de K est construit à partir de chaque δ-libre $Y \subset A$ et T et G sont calculés comme suit $G = h_\delta(Y)$ et $T = \psi(Y)$.

L'algorithme utilisé est une extension de MIN-EX (Boulicaut, Bykowski, & Rigotti, 2003) et l'extraction des δ-bi-ensembles est efficace. Par contre, les auteurs de (Besson, Robardet, & Boulicaut, 2006) remarquent que cette théorie présentée tient compte d'une seule dimension, soit celle des attributs.

Dans (Pensa, Robardet, & Boulicaut, 2005) les auteurs introduisent les bi-clusters. L'idée maitresse est le regroupement de concepts formels ou de δ-bi-ensembles qui se chevauchent sur la base de deux paramètres pour chacune des dimensions. Elle est basée sur

la notion de centroïde[65] calculé par l'algorithme CDK-MEANS. Malheureusement, l'algorithme prend en entrée une collection de concepts formels (ou denses) déjà calculée.

7.2.5.2 Les DRBS

(Besson, Robardet, & Boulicaut, 2006) présentent un nouveau type de motif, le DRBS[66]. De plus, ils ont développé une définition alternative du concept formel basée sur les valeurs '0' plutôt que les valeurs '1'. Tout d'abord, la définition de la fonction calculant le nombre de zéros, noté Z, est énoncée. Ensuite la définition alternative du concept formel est formulée.

Définition 7.7 – Fonctions Z

Soient O un ensemble d'objets, $o \in O$ un objet, A un ensemble d'attributs, $a \in A$ un attribut, $K = (O, A, I)$ un contexte formel et (X, Y) un concept formel sur K. On note $Z_o(x, Y)$ le nombre de zéros d'un objet x sur les attributs de Y comme suit, $Z_o(x, Y) = |\{y \in Y \mid (x, y) \notin I\}|$. De manière duale, on note $Z_a(y, X) = |\{x \in X \mid (x, y) \notin I\}|$ comme étant le nombre de zéros d'un attribut y sur les objets de X.

Définition 7.8– Concept formel, une nouvelle définition

Soient O, A, K et (X, Y) tels que définis précédemment. Un bi-ensemble (X, Y) est un concept formel de K ssi :

(1) $\forall x \in X, Z_o(x, Y) = 0$ ou $\forall y \in Y, Z_a(y, X) = 0$
(2) $\forall x \in O \backslash X, Z_o(x, Y) \geq 1$ et $\forall y \in A \backslash Y, Z_a(y, X) \geq 1$

[65] En physique, un quasi-synonyme de centroïde serait le centre des forces parallèles.

[66] DRBS: Dense and Relevant Bi-Sets. On pourrait traduire par bi-ensembles denses et pertinents.

L'expression (1) de la définition précédente dit qu'il n'existe pas de valeur 0 dans le concept formel. Tandis que l'expression (2) affirme qu'il existe au moins une valeur à 0 en dehors de X et de Y du concept formel, respectivement $O \backslash X$ et $A \backslash Y$. La définition suivante énonce le principe de concept dense basé sur les deux dimensions.

Définition 7. 9 – Concept dense

Soient (X, Y) un concept dense sur K et α, α^* des entiers positifs. α, α^* sont des paramètres choisis par un utilisateur qui déterminent le nombre de '0' sur X et Y respectivement. Le concept (X, Y) est dit dense ssi il satisfait la contrainte dense anti-monotone suivante.

$C_d(\alpha, \alpha^*, (X, Y)) \equiv (\forall x \in X, Z_o(x, Y) \leq \alpha)$ et $(\forall y \in Y, Z_a(y, X) \leq \alpha^*)$

Notons que les valeurs des paramètres α, α^* n'ont pas besoin d'être les mêmes. La définition ci-haut permet, avec $\alpha = \alpha^* = 0$, d'extraire les concepts formels. Si $\alpha > 0$ et $\alpha^* > 0$, on extrait alors des concepts denses. On obtient ici un effet de zoom sur les concepts tel que vu dans (Ventos & Soldano, 2005). Par contre, cet effet de zoom est sur les deux dimensions, la dualité recherchée est donc préservée[67]. La prochaine définition énonce le principe de pertinence des concepts.

Définition 7.10 – Concept pertinent

Soient (X, Y) un concept dense sur K et δ, δ^* des entiers positifs. δ, δ^* sont des paramètres choisis par un utilisateur qui déterminent le nombre de '0' sur X et Y respectivement. Le concept (X, Y) est dit pertinent ssi il satisfait la contrainte suivante.

$C_r(\delta, \delta^*, (X, Y)) \equiv (\forall o \in O \backslash X, \forall x \in X, Z_o(o, Y) \geq Z_o(x, Y) + \delta)$ et
$(\forall a \in A \backslash Y, \forall y \in Y, Z_a(a, X) \geq Z_a(y, X) + \delta^*)$

[67] Le but des auteurs de (Besson, Robardet, & Boulicaut, 2006) est de préserver le plus possible les caractéristiques définies dans l'AFC.

Autrement dit, les objets de X ont une densité de valeurs '1' plus élevés sur les attributs de Y que leurs contreparties $O\backslash X$. La duale s'applique également sur Y et $A\backslash Y$.

L'utilisation des deux contraintes précédemment définies par un algorithme d'extraction produira des concepts denses et pertinents. Dans (Besson, Robardet, & Boulicaut, 2006) , on introduit l'algorithme DR-MINER qui est basé sur DUAL-MINER (Bucila, Gehre, Kifer, & White, 2003). DR-MINER est un algorithme de type diviser-pour-régner. L'algorithme est élégant dans sa conception. Il garde les principes des concepts formels tout en acceptant des exceptions sur les deux dimensions d'un concept. On peut choisir de n'extraire que les concepts denses en initialisant les paramètres $\delta = \delta^* = 1$. La structure de données utilisée est un arbre d'énumération dont les feuilles sont les concepts résultant du calcul de DR-MINER. L'envers de la médaille est que l'extraction des DRBS peut s'avérer extrêmement difficile dans des contextes formels de grande taille, particulièrement lorsque les données sont fortement corrélées (Boulicault & Besson, 2008). Afin de faciliter l'extraction, on peut choisir un seuil de fréquence minimal. Si ce dernier est trop faible, l'extraction demeure difficile. Ce qui facilite l'extraction des DRBS ce sont les seuils sur la pertinence. Plus ils sont élevés, plus on élague des candidats. Par contre on n'obtient qu'un petit ensemble de DRBS. Dans certains domaines, c'est le résultat désiré mais ce n'est pas toujours le cas. Dans (Boulicault & Besson, 2008), on présente différentes contraintes pouvant être utiles tant pour une extraction plus efficace que pour une collection plus utile à l'utilisateur.

Dans les tests dans le cadre du projet de ce mémoire, nous avons utilisé les contraintes de maximalité des concepts. En un mot, un concept (dense ou formel) est conservé s'il a une dimension suffisante sur X et Y. Cette contrainte permet d'élaguer des DRBS candidats non-intéressants et rends l'extraction par DR-MINER un peu plus efficace. Nous déferrons la présentation de l'algorithme au chapitre 9 de ce mémoire.

7.3 Discussion

Notons tout d'abord que le projet issu de ce mémoire se basera sur MagaliceA. Rappelons que cet algorithme extrait les concepts formels et les générateurs minimaux. Il calcule également les relations de précédence et prend en compte le seuil de fréquence minimal lorsque présent. Le projet ajoutera, si c'est possible, l'extraction des concepts denses tout en conservant les principes susmentionnés. Dès lors, les travaux présentés dans la sous-section précédente seront évalués en fonction de ce qui vient d'être émis comme "exigences".

Dans le cadre de ce mémoire, nous abondons dans le sens de l'article de (Besson, Robardet, & Boulicaut, 2006). En effet, il faut conserver le plus possible les principes de l'AFC, principalement la dualité dans les théorèmes et définitions. À cet égard les DRBS et l'algorithme DR-MINER réussissent à remplir cette mission, alors que la plupart des techniques couvertes dans ce chapitre ne proposent pas cette dualité. Par contre, les relations de précédence et les générateurs minimaux ne sont pas calculés. Outre (Ventos & Soldano, 2005) et (Kuznetsov, Obiedkov, & Roth, 2007), aucune des autres propositions n'offre le calcul de la relation d'ordre dans les algorithmes ni de la représentation du treillis[68]. On extrait tout simplement les concepts ou itemsets. De plus, même dans ce cas, (Kuznetsov, Obiedkov, & Roth, 2007) ne promettent pas que le résultat du calcul des concepts par la stabilité intensionnelle (ou extensionnelle) produira nécessairement un treillis.

Dans le cas des générateurs minimaux, il est possible que l'algorithme de (Pensa & Boulicaut, 2005) les produisent. Puisque cet algorithme calcule les δ-bi-ensembles à partir des δ-libres, on peut supposer qu'ils sont conservés[69] dans une structure quelconque. Les autres propositions ne mentionnent pas les générateurs minimaux.

---

[68] Dans ces articles on peut décrire ou présenter la relation d'ordre des concepts formels, voire même une définition de la relation d'ordre des nouveaux concepts mais on ne trouve pas d'algorithme ou de pseudo-code produisant ce calcul.

[69] Ou qu'il y a possibilité de les conserver.

Notre proposition d'algorithme incrémental utilisant la représentation verticale sera présentée dans le prochain chapitre. À la lumière des travaux brièvement examinés dans ce chapitre, et de nos premières esquisses d'algorithme incrémental supportant les concepts denses, nos idées préliminaires coïncident avec les théories présentées dans (Besson, Robardet, & Boulicaut, 2006) tant pour la définition de la densité dans les concepts, que pour la préservation des principes de la dualité dans les TCF.

CHAPITRE VIII
ALGORITHME MAD-G ET LES CONCEPTS DENSES

Ce chapitre explore la possibilité qu'un algorithme incrémental comme MagaliceA puisse être transformé afin d'extraire des concepts denses. La section 8.1 décrit les propriétés recherchées tant pour le motif lui-même que pour l'algorithme d'extraction. La section 8.2 signale deux difficultés auxquelles un algorithme incrémental de type MagaliceA pourrait faire face lors de l'extraction de concepts denses. La section 8.3 présente des contraintes et heuristiques qui seront utile pour l'algorithme d'extraction des concepts denses. Dans la section 8.4, on propose la fonction estDense vérifiant qu'un concept est un candidat valide à la condensation ainsi que l'algorithme MAD-G qui extraie les concepts denses. La section 8.5 présente une trace partielle de l'algorithme MAD-G sur le contexte formel DuCoin. Les sections 8.6 et 8.7 discutent des impacts sur les générateurs minimaux et sur la catégorisation des concepts. La section 8.8 présente la complexité temporelle de l'algorithme MAD-G. La section 8.9 conclu ce chapitre.

8.1 Propriétés recherchées

Concepts formels/denses :

- Les concepts doivent conserver la propriété de maximalité sur les deux dimensions selon les seuils de densité choisis.
- Les seuils de densité minimale devront couvrir les deux dimensions du concept en construction. La valeur minimale légale des seuils de densité est 0. Si cette valeur est 0, alors les concepts seront formels.

Treillis de concepts formels/denses :

- Dans le cas d'un treillis de concepts, le $\perp$ et $\top$ doivent être préservés.
- Dans le cas d'un iceberg de concepts, le $\top$ doit être préservé.
- Les relations de précédence entre les concepts doivent être calculées de telle sorte qu'elles respectent les définitions de l'AFC.
- Un effet de zoom sur l'ensemble des concepts est recherché.

Générateurs minimaux

- Les générateurs minimaux doivent être présents et être calculés selon les principes de l'algorithme INCA-GEN.

Algorithme

- L'algorithme proposé doit prendre en compte le seuil de fréquence minimal s'il est présent et calculer les concepts formels/denses fréquents en conséquence.
- L'algorithme incrémental MagaliceA, utilisant la représentation verticale, sera le point de départ de notre algorithme d'extraction des concepts denses.

8.2 Les concepts denses et MagaliceA, deux difficultés observées

Afin que l'algorithme incrémental basé sur MagaliceA puisse extraire des concepts denses, certaines difficultés devront être surmontées. Le contexte formel DuCoin (chapitre 5, figure 5,2(c)) servira à illustrer deux difficultés majeures dont l'algorithme doit tenir compte. La figure suivante présente la trace complète de la construction du TCF par MagaliceA.

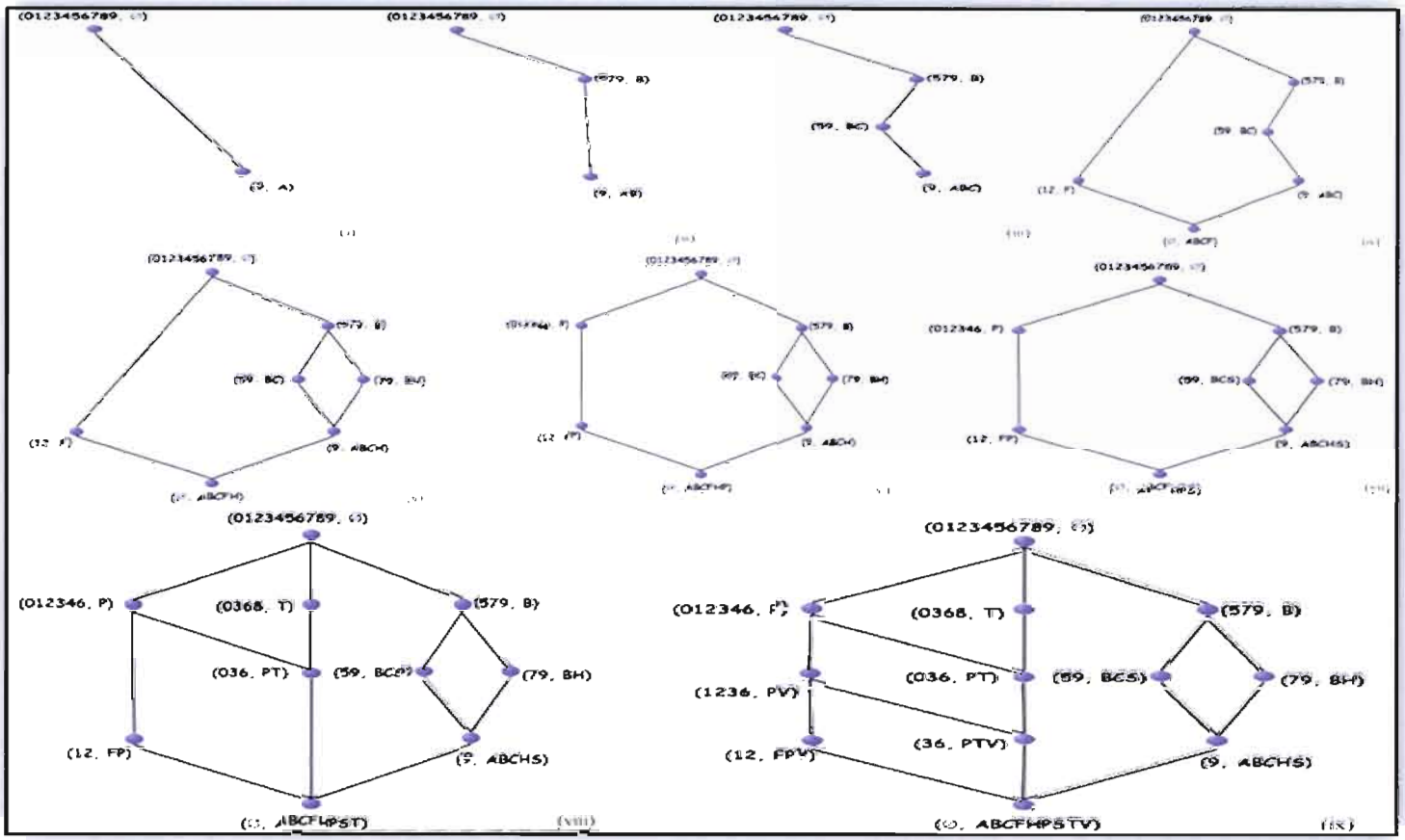


Figure 8.1 – Trace complète de l'algorithme MagaliceA

8.2.1 Protection des concepts $\top$ et $\bot$ pour la préservation du treillis

L'un des tenants des propriétés recherchées afin de conserver l'intégrité du treillis selon les notions de l'AFC est la préservation et le calcul correct des concepts $\bot$ et $\top$. Dès lors, l'algorithme proposé devrait empêcher toute fusion des concepts $\top$ et $\bot$ afin d'éliminer toute possibilité de dégénérescence du treillis (et des concepts denses).

Il est à noter que l'algorithme MagaliceA, lors du traitement de la première transaction (lignes 04 à 11 du tableau 5.1), construit automatiquement l'extension complète du concept $\top$ du treillis. Le fait d'avoir l'une des deux composantes complétée, i.e. l'extension du concept $\top$, est pour le moins rassurant et facilitera la gestion de la "protection" dudit concept.

Le cas du concept $\bot$ est un peu plus compliqué. En ce qui concerne le traitement du contexte formel DuCoin, on ne sait qu'à la fin du traitement dudit contexte que la valeur du concept $\bot$ sera ($\varnothing$, ABCFHPSTV). Toutefois, la figure 8.2 nous révèle que l'intension du concept $\bot$ bouge constamment. Ce n'est qu'une fois le contexte formel est entièrement traité, que l'intension du concept $\bot$ est connu et fixé. Autrement dit, aucune des deux composantes du concept $\bot$ ne sont connues dès le départ. Empêcher la condensation par densité sur le concept $\bot$ en construction pourrait limiter la condensation du treillis de concepts. L'exemple suivant illustre le problème.

Exemple 8.1

Soient α et α^* des paramètres contrôlant respectivement la densité sur les lignes et les colonnes. Soient $\alpha = 2$ et $\alpha^* = 1$. Dans la figure 8.1 (ii), on trouve les concepts en construction (579, B) et (9, AB). Les seuils de densité sont respectés donc techniquement, la fusion de ces deux concepts en construction peut s'effectuer et produira un seul concept (dense) soit, (579, AB). Par conte le concept (9, AB) joue le rôle temporaire de concept $\bot$ du treillis, on ne devrait pas fusionner ces concepts.

❑

8.2.2 Le problème des singletons

L'exemple suivant illustre le problème à définir.

Exemple 8.2

Supposons que la boucle d'exécution de MagaliceA est à l'itération (v) de la figure 8.1, i.e. le traitement de l'attribut H et ses objets associés 5 et 9. Supposons $\alpha = 1$ $\alpha^{*} = 2$. MagaliceA créera un nouveau concept formel (en construction) (79, BH) et fera une modification sur le concept (9, ABC) qui deviendra (9, ABCH). Notons que, selon la valeur des paramètres de condensation, il aurait été possible de fusionner (9, ABCH) et (79, BH). En effet, ces deux concepts ne sont ni $\perp$ ni $\top$et, ils respectent les seuils de condensation. Cependant, il n'y a aucune indication que ces deux concepts sont formels et terminés ou encore en construction par rapport au contexte formel dont ils sont issus. Ces deux concepts ne sont complets et "formels" que pour cette itération uniquement et non pour le contexte formel dans son ensemble. Pour s'en convaincre, que serait-il arrivé lors du traitement de l'attribut S (figure 8.2 (vii)) si les deux concepts précédents avaient été déjà fusionnés?

❑

Étant donné, qu'une des deux composantes du concept en construction n'a qu'un seul élément (un singleton), il est difficile de fusionner ce concept avec tout autre concept. Il est vrai que sur le moment les seuils de densité sont respectés. Cependant, à la fin du traitement complet du contexte formel, on se rendrait compte de l'erreur. Cette erreur a deux "saveurs". La première implique qu'une condensation par densité a été produite trop tôt dans le cycle de traitement de l'algorithme. Il est vrai que le concept est dense, mais une meilleure solution aurait pu être trouvée si le concept ayant un élément singleton comme extension ou intension n'avait pas été condensé. La deuxième "saveur" a une implication plus importante. Il a été noté, durant nos premières expérimentations, qu'il y avait de temps à autre dégénérescence du treillis et/ou des concepts denses. Cette dégénérescence était quelquefois causée par la fusion d'un concept ayant un élément singleton. Il semble donc plus sage de ne pas accepter la condensation d'un concept ayant un singleton comme extension et/ou intension.

8.2.3 La complétude du TCF

Les deux problèmes mentionnés ci-haut trouvent leur source dans le fait que les concepts sont déclarés complets qu'une fois que MagaliceA a terminé toutes les transactions d'un ensemble donné (contexte formel donné). La prochaine définition énonce informellement le principe de la complétude du TCF et de ces concepts.

Proposition 8 1– Complétude du TCF après *n* insertions.

Soit $K = (O, A, I)$ un contexte formel de n colonnes par m lignes. Supposons un algorithme incrémental utilisant la représentation verticale de K qui construit le treillis pour chaque attribut et ses objets associés. Le nombre d'itération dans le contexte formel sera dans l'intervalle $[1, n]$ et les attributs (et ses objets associés) seront traités un à la fois selon l'ordre usuel. On peut alors émettre la proposition suivante. Pour une colonne $i \in [1, n]$, le TCF est complet pour les colonnes de 1 à i après le traitement de l'attribut et de ses objets associés de la colonne i.

Se référant à l'exemple 8.2, il est vrai que la fusion des concepts (9, ABCH) et (79, BH) était possible par rapport au traitement local en cours. Mais la condensation pourrait être fausse ou inadéquate une fois le contexte formel complètement traité. C'est une différence importante entre les algorithmes MagaliceA et DR-MINER. En effet, ce dernier vérifie par la contrainte de pertinence si le candidat au cours de la construction est valide. De plus, DR-MINER traite la création des DRBS un à la fois jusqu'à ce qu'il soit complet. En revanche, MagaliceA n'a qu'une vue partielle (ou locale). Pour une itération i sur le contexte formel, MagaliceA n'a connaissance que des concepts produit par les itérations 1 à $i - 1$ et à la fin du traitement de l'itération i l'algorithme produit (ajout de concepts, mise à jour de concepts existants) les concepts formels pour les colonnes 1 à i du contexte formel. Par contre, il n'y a pas de fonctions qui lui permettent de savoir ce qu'il y a dans les colonnes $i + 1$ à n. C'est naturellement le principe d'un algorithme par incrément par rapport un algorithme de traitement en lot.

8.3 Contraintes, heuristiques et type d'algorithme

Avec ce qui vient d'être dit à la section précédente, il est clair que des choix s'imposent quant à la façon d'extraire les concepts denses avec l'algorithme MagaliceA.

Tout d'abord, le type d'algorithme sera basé sur une approche gloutonne pour l'algorithme d'extraction des concepts formels et denses. L'approche gloutonne a comme objectif de faire un choix optimal localement dans le but qu'une solution optimale soit calculée au niveau global. Il est néanmoins admis qu'au niveau global ce ne sera pas toujours une solution optimale, cependant une solution valide (ou possible) sera quand même calculée.

Les principes de l'AFC doivent s'appliquer le plus possible aux concepts denses. Dès lors deux contraintes sont définies afin de protéger l'intégrité du treillis. Définissons d'abord la contrainte sur les concepts $\perp$ et $\top$.

Définition 8. 1 – Contrainte sur les concepts $\perp$ et $\top$

Soit $K = (O, A, I)$ un contexte formel ayant n transactions, $i \in [1, n]$, $\mathcal{L}_{[i]}$ un treillis dans sa i^e itération à partir de K, $c_{[i]} \in \mathcal{L}_{[i]}$ et c_{cand} un concept candidat. Si $c_{[i]}$ est $\perp$ ou $\top$ de $\mathcal{L}_{[i]}$, la condensation (ou la fusion) entre $c_{[i]}$ et c_{cand} ne doit pas se faire.

Les expériences sur les premières versions de l'algorithme MAD-G ont démontré que la contrainte sur le $\perp$ telle que définie précédemment est trop restrictive. D'une part, le concept $\perp$ bouge constamment contrairement au concept $\top$ dont l'extension est créée dès le début par l'algorithme MagaliceA. La figure 8.2 démontre que la position du concept $\perp$ ne se précise qu'à l'itération (iv). Il est donc possible que durant les itérations précédentes des occasions de condensation furent manquées. La définition suivante propose à cet effet une relaxation pour le concept $\perp$.

Proposition 8 2 – Relaxation de la contrainte sur le concept ⊥

Si l'extension du concept ⊥ est vide alors aucune condensation n'est permise sinon la condensation est permise. Le cas où un concept en construction joue temporairement le rôle de concept ⊥ est partiellement pris en compte.

À l'origine, nous avons considéré d'empêcher toute fusion sur les atomes et co-atomes[70]. Cette contrainte s'est avérée trop restrictive. Par contre, tel que mentionné à la sous-section 8.2, il existe un danger lors de la condensation de concepts dont l'extension ou l'intension n'a qu'un élément. La contrainte sur les singletons est énoncée à la définition suivante.

Définition 8. 2 – Contrainte sur le singleton

Soient $K = (O, A, I)$ un contexte formel ayant n transactions, $i \in [1, n]$, $\mathcal{L}_{[i]}$ un treillis dans sa i^e itération à partir de K, $c_{[i]} \in \mathcal{L}_{[i]}$ et c_{cand} un concept candidat. Si l'extension ou l'intension de $c_{[i]}$ ou c_{cand} est un singleton, la condensation entre $c_{[i]}$ et c_{cand} ne doit pas avoir lieu.

S'il fallait caractériser ces deux types de contraintes, on les considérerait sans doute comme des contraintes syntaxiques car elles ne relèvent pas du contexte formel mais uniquement du treillis (en construction), c'est à dire de la théorie *Th* vu au chapitre 3 de ce mémoire.

À ces deux contraintes, la contrainte de densité telle qu'énoncé dans (Besson, Robardet, & Boulicaut, 2006) est définie ci-après. La fonction Z a été définie au chapitre 7 de ce mémoire.

[70] Le terme co-atome provient de l'anglais co-atom. Dans la littérature, on utilise également les termes premier, co-premier ou prime.

Définition 8.3 – Concept dense, contrainte de densité

Soient (X, Y) un concept dense sur K et α, α^* (des entiers positifs) représentant le nombre de '0' sur X et Y respectivement. Le concept (X, Y) est dit dense ssi il satisfait la contrainte dense anti-monotone suivante.

$C_d(\alpha, \alpha^*, (X, Y)) \equiv (\forall x \in X, Z_o(x, Y) \leq \alpha)$ et $(\forall y \in Y, Z_a(y, X) \leq \alpha^*)$

Le dernier point est une heuristique. Il a été observé que la condensation est sensible à l'ordre des transactions de la représentation verticale du contexte formel. Dans l'ordre lexicographique habituel et en utilisant les contraintes précédemment énoncées, la condensation des concepts donne de faibles résultats, i.e. très peu de concepts sont fusionnés. En ordonnant la représentation verticale en ordre ascendant de taille des objets associés à un attribut des résultats semblables ont été également obtenus. Par contre, en ordonnant la représentation condensée en ordre descendant de taille des objets associés à un attribut, de meilleurs résultats ont été calculés quant à la condensation des concepts. La figure suivante illustre cet ordre désiré pour le contexte formel présenté au début de ce chapitre.

Objet	Attribut								
	P	T	V	B	C	F	H	S	A
0	✖	✖							
1	✖		✖			✖			
2	✖		✖			✖			
3	✖	✖	✖						
4	✖								
5				✖	✖			✖	
6	✖	✖	✖				✖		
7				✖					
8		✖							
9				✖	✖		✖	✖	✖

Figure 8.2 – Contexte formel réordonné

8.4 Algorithme de condensation par densité

8.4.1 Première approche

Une première approche dans la conception de l'algorithme de condensation, épousant la méthode gloutonne, fut la suivante. Lors de la création d'un nouveau concept, avant de calculer la relation de précédence (et les générateurs minimaux), on balaye en largeur le treillis du $\top$ au $\perp$ afin de trouver le premier concept satisfaisant les contraintes sur le $\perp/\top$, le singleton et la densité. L'algorithme en pseudo-code est illustré au tableau suivant.

Tableau 8.1 – Algorithme fusionPossible

```
▷ FONCTION fusionPossible – vérifie si un concept candidat peut être fusionné avec un concept existant.
▷ Paramètres
▷ L un treillis (E)
▷ c_cand un concept candidat à fusionner (E)
▷ α un nombre maximal de 0 sur l'extension permettant la fusion(défini par l'utilisateur) (E)
▷ α* un nombre maximal de 0 sur l'intension permettant la fusion (défini par l'utilisateur) (E)
▷ Retourner vrai si fusion possible sinon faux

(01)FONCTION fusionPossible(L, c_cand, α, α*)

(02)        ▷ Vérifier la contrainte du singleton
(03)        SI c_cand est un ALORS
(04)                RETOURNER faux
(05)        POUR CHAQUE c ∈ L de Infimum au supremum (traversée par niveau) FAIRE
                ▷ Vérifier les contraintes du singleton et de l'infimum/supremum sur le concept existant
(06)                SI c n'est pas un singleton ET c n'est pas un ⊤ ou ⊥ ALORS
(07)                        c_temp ← c and c_cand sont fusionnés
                        ▷ vérifier la contrainte de densité avec α, α* ALORS
(09)                                RETOURNER vrai
(10)        FIN POUR
(11)        RETOURNER faux
(12)FIN fusionPossible
```

Les lignes 03 et 04 de l'algorithme vérifient si le candidat est un singleton. Si c'est le cas il est inutile d'aller plus loin. En effet, cette contrainte énoncée à la définition 8.4, s'applique pour les concepts existants du treillis et le concept candidat. Si le concept candidat ne satisfait pas la condition de la ligne 03, i.e. le concept candidat n'est pas un singleton, alors on balaye le treillis afin de trouver un concept potentiel pour une fusion

Un problème se pose avec cet algorithme. Tout d'abord, MagaliceA a déjà beaucoup de boucles imbriquées outre un tri sur la collection de concepts. On doit maintenant ajouter à cela que l'algorithme fusionPossible peut potentiellement visiter tous les concepts du treillis. Si les seuils de densité sont faibles, ce balayage pourrait bien être coûteux dans des contextes de grande taille.

8.4.2 Deuxième approche

Il a été mentionné au chapitre 6 de ce mémoire que le calcul des générateurs minimaux dans l'algorithme INCA-GEN utilise la procédure COMPUTE_CLASS. Cette dernière calcule les classes d'équivalence et le mappage χ^+ permet d'extraire le concept minimal de ces classes. Ce concept est utile pour le calcul des générateurs minimaux et des liens de précédence lors de la création de nouveaux concepts. De plus, la création d'un nouveau concept formel dans MagaliceA se fait à partir du concept minimal. Dès lors, pourquoi ne pas vérifier si, pour des seuils de densité quelconques, la fusion du nouveau concept et du concept minimal produirait un concept dense. Le pseudo-code de cette nouvelle fonction est présenté au tableau suivant.

Tableau 8.2 – Algorithme estDense

```
▷ FONCTION estDense – vérifie si le candida peut être fusionné avec le concept minimal.
▷ Paramètres
▷ c_min le concept minimal associé au concept candidat (IN)
▷ c_cand le concept candidat (E)
▷ α un nombre maximal de 0 sur l'extension permettant la fusion(défini par l'utilisateur) (E)
▷ α* un nombre maximal de 0 sur l'intension permettant la fusion (défini par l'utilisateur) (E)
▷ Retourner vrai si fusion possible sinon faux

(01)FONCTION estDense(c_min, c_cand, α, α*)
        ▷ Vérifier la contrainte du singleton
(02)    SI c_cand has a singleton ALORS
(03)            RETOURNER faux
(04)    SI c_min n'est pas un ET n'est pas un ⊤ou l'extension de ⊥= ∅ ALORS
(05)            c_temp ← c_min et c_cand sont fusionnés
(06)            SI c_temp est dense selon α, α* ALORS
(07)                    RETOURNER vrai
(08)    RETOURNER faux
(09)FIN estDense
```

L'avantage de cet algorithme, par rapport à l'algorithme du tableau 8.1, est qu'il limite la portée de la condensation à un seul concept existant soit le concept minimal d'une classe d'équivalence. Ce dernier a d'ailleurs beaucoup plus de chances, lors du test sur la condensation (ligne 06 du tableau 8.2) avec le concept candidat, de produire un concept dense puisque le concept candidat a été construit à partir d'un concept minimal et de la transaction du contexte formel. Un autre avantage non-négligeable est que la complexité temporelle de cette fonction est de l'ordre de $\theta(1)$.

Le pseudo-code de haut niveau de cet algorithme, nommé MAD-G[71] basé sur MagaliceA, est présenté au tableau suivant.

[71] MAD-G pour MagaliceA extrayant les concepts Denses utilisant une méthode Gloutonne.
Également en anglais : MagaliceA extracting Dense concepts using a Greedy approach.

Tableau 8.3 – Algorithme MAD-G

```
▷ PROCÉDURE MAD-G – calcule le treillis de concepts
▷ Paramètres
▷ L un treillis (E/S)
▷ X l'ensemble des objets associés à l'attribut (E)
▷ a l'attribut du contexte formel (E)
▷ minsup, le support minimal défini par l'utilisateur (E)
▷ α, α•, le paramètres de densité définis par l'utilisateur (E)
▷ RETOURNER
▷ X un ensemble d'objets augmenté à être condensé ou NUL si la condensation est non nécessaire

(01) FONCTION MAD-G(L, X, a, minsup, α, α•) RETOURNER X

(02)      SI (|X| < minsup) ALORS RETOURNER NUL

(03)      SI (L = ∅) ALORS
(04)         Créer le top et le bottom de L
(05)         Calculer les liens
(06)         Calculer les générateurs
(07)         RETOURNER NUL
(08)      FIN SI

(09)      Calcule les Classes de L
(10)      Trier L en ordre non-décroissant de taille de l'intension
(11)      POUR chaque concept c in L FAIRE
(12)         POUR all c̄ in Classes(c) FAIRE
(13)            SI c.extension ⊆ X ALORS
(14)               c.intension ← c.intension ∪ {a} ▷ mise à jour du concept existant
(15)               Calculer les générateurs
(16)            SINON
(17)                  SI (|c̄.extension ∩ X| ≥ minsup) ALORS
(18)                     intersect ←c̄.extension ∩ X
(19)                     SI (intersect, c̄.intension ∪ {a}) ∉ Classes ALORS
(20)                           c_cand ← (intersect, c̄.intension ∪ {a})
(21)                           SI (estDense(c̄, c_cand, α, α•)) ALORS
(22)                              RETOURNER X ajouté de la différence entre c̄, et c_cand
(23)                           L ← L ∪ c_cand
(24)                           MODIFIER Classes
(25)                           Calculer liens
(26)                           Calculer les générateurs
(27)                     FIN SI
(28)                  FIN SI
(29)            FIN SINON
(30)         FIN POUR
(31)      FIN POUR
(32)      RETOURNER NUL
(33)FIN MAD-G
```

La différence entre MAD-G et MagaliceA se trouve aux lignes 20 à 22. Avant de traiter le concept candidat à l'insertion dans le treillis, il faut vérifier si une condensation entre le concept minimal et le concept candidate est possible. Si c'est le cas, l'algorithme augmente l'ensemble des objets X par la différence trouvée entre le concept minimal et le concept candidat. Dans le programme[72] appelant MAD-G, on exécutera à nouveau MAD-G avec les mêmes paramètres sauf pour L qui conserve ces changements déjà complétés et X qui fut augmenté. Un exemple d'appel est montré au tableau ci-dessous.

Tableau 8.4 – Exemple d'appel à MAD-G

```
▷ Variables
▷ L un treillis
▷ X l'ensemble des objets associés à l'attribut (E)
▷ newX l'ensemble des objets associés augmenté par MAD-G
▷ a l'attribut du contexte formel (E)
▷ minsup, le support minimal défini par l'utilisateur (E)
▷ α, α*, le paramètres de densité définis par l'utilisateur (E)

...
POUR chaque transaction dans K utilisant la représentation verticale FAIRE
        newX ← MAD-G(L, X, a,minsup, α, α*)
        TANT QUE(newX ≠ ∅) FAIRE
                X ← newX
                newX ← MAD-G(L, X, a,minsup, α, α*)
        FIN TANT QUE
FIN POUR
...
```

8.5 Trace partielle de l'exécution de MAD-G

Dans cette section une trace partielle sur le contexte formel présenté à la figure 8.3 avec les paramètres de condensation par densité $\alpha = \alpha^* = 3$ est effectuée. L'insertion des attributs P et T se fait comme à l'habitude. Par contre pour l'attribut V, le concept minimal (036, PT) et le concept candidat (36, PTV) satisfont la fonction estDense. L'objet 0 est

[72] Possiblement le programme principal.

augmenté de l'attribut V. On appelle MAD-G à nouveau avec cette transaction augmentée et le concept dense (01236, PV) est crée (figure 8.4 (ii)). Notons dans le TCF complet de la figure 8.2 (ix) le concept formel est (1236, PV) et sa version condensée (01236, PV) présentée ci-dessous.

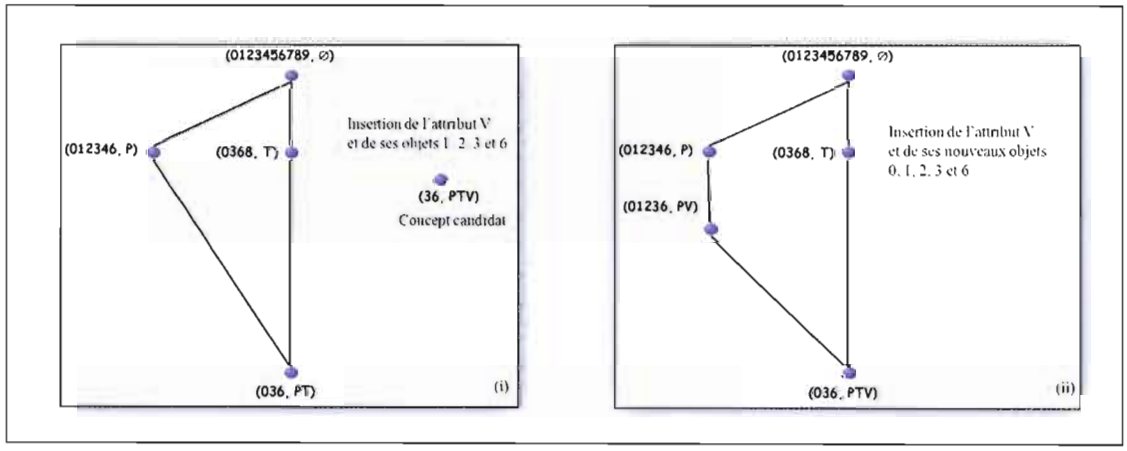


Figure 8.3 – Condensation du candidat (36, PTV)

Les attributs B et C sont traités sans possibilité de condensation. Puis vient l'attribut F et ses objets associés 1 et 2. Le concept minimal est (01236, PV) et le concept candidat est (12, FPV). La contrainte de densité est respectée, dès lors on augmente les objets associés à F et on appelle à nouveau MAD-G sur l'attribut F et ses objets associés 0, 1, 2, 3, 6. La figure suivante présente cette condensation.

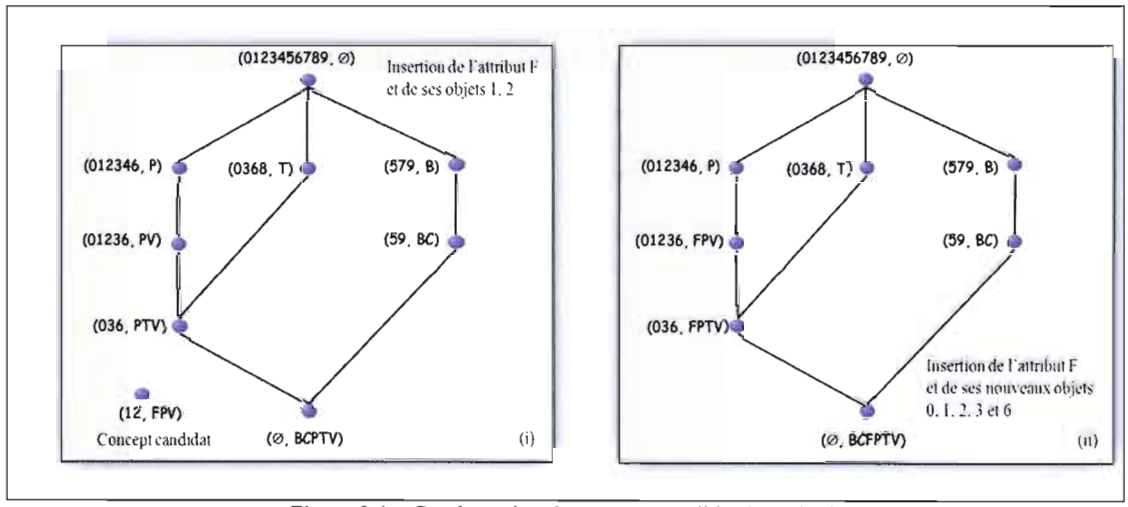


Figure 8.4 – Condensation du concept candidat (12, FPV)

Toutes les autres transactions n'offrent aucune possibilité de condensation. Le TCD final est présenté à la figure suivante à gauche, le TCF original à droite.

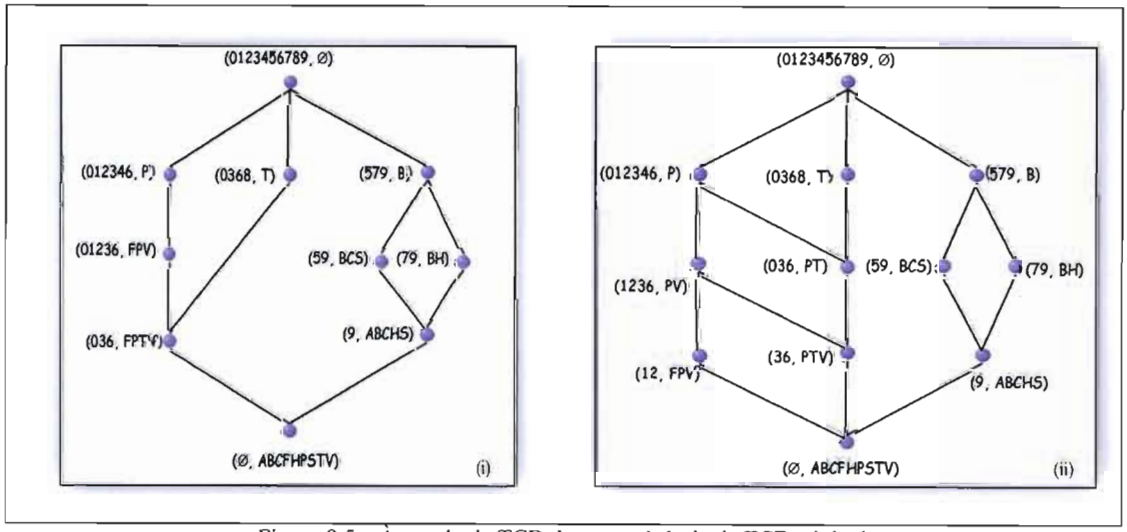


Figure 8.5 – À gauche le TCD denses et à droite le TCF original

La figure suivante présente le concept formel ordonné selon le TCD de la figure 8.6 (i). Cette représentation du contexte fait ressortir les rectangles maximaux avec exceptions trouvés par l'algorithme MAD-G.

Objet	Attribut								
	F	P	V	T	A	B	C	H	S
1	✖	✖	✖						
2	✖	✖	✖						
0	✖	✖	✖	✖					
3	✖	✖	✖	✖					
6	✖	✖	✖	✖					
4		✖							
8				✖					
5						✖	✖		✖
7						✖		✖	
9					✖	✖	✖	✖	✖

Figure 8.6 – Contexte formel ordonné sur le TCD

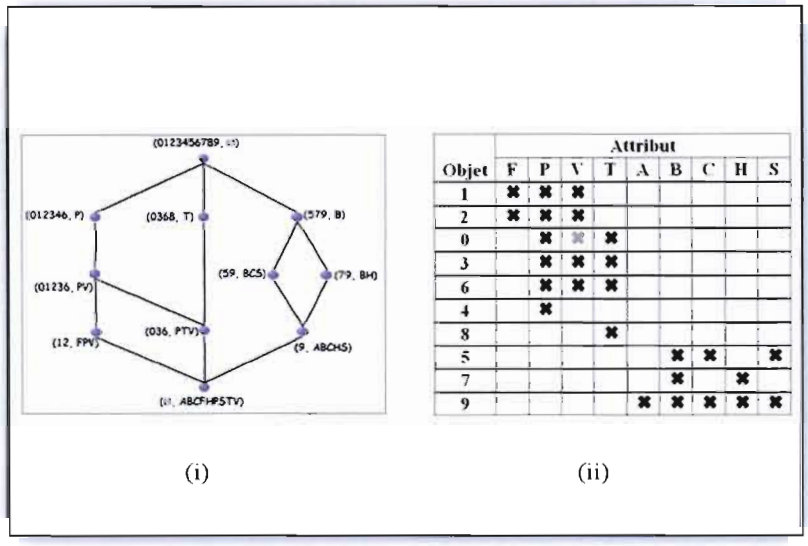
Les ✖ de couleur pale marquent là où les relations d'incidence *oIa* ont été supposées bruitées. On note que le rectangle maximal bruité ou concept dense (01236, FPV) possède tout au plus trois exceptions sur ses deux dimensions. Il en va de même pour le concept dense (036, FPTV). Ce dernier pourrait poser problème. En effet, ce concept a comme attribut F alors que dans son concept formel d'origine cet attribut n'apparaît jamais. Deux raisons expliquent ce phénomène.

- Puisque la relation d'ordre du treillis de concepts est maintenue selon la définition énoncée dans l'AFC[73], cette relation doit être respectée. Conséquemment, le concept (036, FPTV) est tributaire du concept (01236, FPV). Cette apparente aberration du concept dense (036, FPTV) est en fait une bonne nouvelle quant au maintien de l'intégrité des concepts denses et du treillis lui-même.

[73] Voir le chapitre 5 de ce mémoire pour la définition de la relation d'ordre

- La condensation des concepts est fonction des paramètres de densité choisis par l'utilisateur. Dans le cas qui nous occupe, α et α^* sont à 3 le concept (036, FPTV) réponds donc à la contrainte de densité, i.e. la contrainte de densité n'est pas violée.

La figure suivante présente le TCD avec une densité α et α^* à 1 et le contexte formel formé de rectangles maximaux avec exceptions.



Objet	Attribut								
	F	P	V	T	A	B	C	H	S
1	✖	✖	✖						
2	✖	✖	✖						
0		✖	✖	✖					
3		✖	✖	✖					
6		✖	✖	✖					
4		✖							
8				✖					
5						✖	✖		✖
7						✖		✖	
9					✖	✖	✖	✖	✖

(i) (ii)

Figure 8.7 – TCD (i) et rectangles maximaux (ii) pour $\alpha = \alpha^* = 1$

Les concepts denses de la figure ci-dessus présentent une condensation plus "réaliste" de la situation avec des paramètres de densité plus faibles. Cette expérience nous amène à parler d'une propriété voulue dans MAD-G, nommément l'effet de zoom. Plus on augmente les paramètres contrôlant la densité plus la granularité fine du concept d'origine s'étiole. Cependant, dans le cas du contexte formel DuCoin, la condensation des concepts se limite au treillis de la figure 8.6 (i), i.e. pour des $\alpha = \alpha^* \geq 3$, le treillis demeure stable, il n'y a plus possibilité de condensation par densité. Dans ce cas-ci, ce sont les contraintes sur les concepts $\perp$ et $\top$ sur les singletons qui limitent la condensation. Pour des $\alpha = \alpha^*$ à 1 ou 2, on obtient le treillis de la figure 8.8 (i).

Due à la nature de l'algorithme MAD-G, et par voie d'antécédence de l'algorithme MagaliceA, la contrainte sur les singletons préservent, dans le cas du contexte formel DuCoin, ses partitions "naturelles". Du moins, c'est notre intuition. Pour en être s'en assurer, il faudrait le prouver formellement.

8.6 Impact de la condensation sur les générateurs minimaux

Au chapitre 6 on a illustré l'ICF avec les générateurs minimaux (figure 6.10). La figure suivante présente l'iceberg de concepts denses et ses générateurs minimaux.

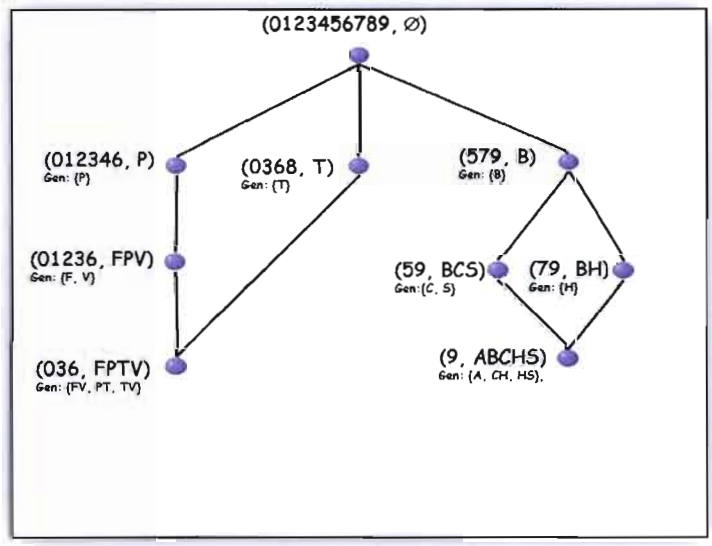


Figure 8.8 – Iceberg de concepts denses et générateurs minimaux

L'intuition première fut que l'algorithme INCA-GEN intégré dans l'algorithme MAD-G produirait des générateurs minimaux semblables aux δ-libres de (Boulicaut, Bykowski, & Rigotti, 2003). Il semble que ce ne soit pas le cas. Le cas de test mentionné dans la citation précédente a été exécuté et les mêmes δ-libres ont été trouvés. Malheureusement, les auteurs de (Boulicaut, Bykowski, & Rigotti, 2003) les ont généré avec

un δ de 1 alors que MAD-G les a généré avec les paramètres de densité α et α^* à 3. Il ne faut pas se surprendre de ce résultat pour autant. D'une part, on croit que l'algorithme MAD-G produira une collection de concepts denses valides mais pas nécessairement optimale. Il faut ajouter à cela d'autres limites de MAD-G qui seront traitées au chapitre suivant. Ce qui doit être retenu au sujet des générateurs minimaux qu'ils soient δ-libres ou non, est que le calcul d'INCA-GEN dans MAD-G est juste en fonction des concepts denses extraits du contexte formel. Bien que le calcul soit juste, est-il exact de présenter ces générateurs minimaux comme tels ? Cette question reste ouverte pour le moment.

8.7 Impact sur la caractérisation des concepts

Examinons tout d'abord les paramètres de densité $\alpha = \alpha^* = 3$ et la transaction ayant l'attribut H et ses objets associés 7 et 9. L'algorithme MAD-G, tout comme MagaliceA, trouvera le concept minimal (B, 579) qui sera le géniteur de (BH, 79). Dès lors (B, 579) est un géniteur de la classe $\mathbb{G}$ et (BH, 79) est donc un nouveau concept de la classe $\mathbb{N}$. Jusqu'à présent MAD-G réagit de la même façon que décrite au chapitre 6 (section 6.1). Il ne faut pas s'en surprendre puisque pour ce concept aucune condensation n'est possible due aux contraintes sur le singleton.

Examinons maintenant l'exemple illustré à la figure 8.4 avec les seuils de densité $\alpha = \alpha^* = 3$ et la transaction sur l'attribut V et ses objets associés 1, 2, 3 et 6. Le nouveau concept calculé est (36, PTV) et son géniteur (036, PT). En temps normal le nouveau concept serait créé, mais la fonction estDense indique que ces deux concepts peuvent être condensés. Dès lors, MAD-G calcule la nouvelle transaction soit l'attribut V et ses nouveaux objets associés 0, 1, 2, 3 et 6. Ceci a pour conséquence de terminer la présente exécution et de rappeler MAD-G avec la nouvelle transaction. L'ancien géniteur est modifié passant de (036, PT) à (036, PTV) et il y a création d'un nouveau concept soit (01236, PV) dont le géniteur est (P, 012346). Dans le cas de la figure 8.5, le concept à créer était (12, FPV) et son géniteur (01236, PV). Pour les mêmes raisons que la transaction précédemment discuté, MAD-G

termine son exécution parce que les deux concepts satisfont la contrainte de densité. MAD-G est rappelé avec la nouvelle transaction mais cette fois-ci il n'y a pas de création de nouveau concept mais des modifications sur deux concepts. De façon générale, la caractérisation des concepts sous extraction étudiée au chapitre 6 de ce mémoire est conservée. Reste maintenant à se demander si on doit caractériser les concepts nouveaux et géniteurs qui satisfont la contrainte de densité et ainsi déclenche la création de concepts denses. Cette question demeure ouverte pour le moment.

8.8 Complexité temporelle de MAD-G

Il a été démontré que la complexité temporelle de la fonction estDense est $\theta(1)$ et n'ajoute rien à la complexité temporelle théorique de MagaliceA. Est-ce que MAD-G conserve cette complexité temporelle ? La réponse est non.

On trouve dans le mémoire de maitrise de C. Pagé (Pagé, 2008) une analyse de la complexité de MagaliceA basé sur (Rouane, 2006). Cette complexité est $O(ml \,.\, (m + n))$, où l est le nombre de concepts dans l'iceberg après un l'ajout d'un attribut, m est le nombre d'attributs distincts du contexte formel et n le nombre d'objets du contexte formel. En se référant au pseudo-code illustré au tableau 8.4, on remarque que pour propager la contrainte de densité à son maximum, on doit explorer et traiter tous les cas où la condensation par densité est possible jusqu'à épuisement des possibilités. C'est le rôle de la boucle interne "TANT QUE" du pseudo-code MAD-G. Il est donc théoriquement possible que cette boucle tourne l fois pour chaque attribut à traiter. Puisqu'il y a une possibilité que tous les concepts soient visités à nouveau, notre intuition est que la complexité temporelle théorique du segment de pseudo-code du tableau 8.4 est $O(ml^2 \,.\, (m + n))$.

Il est clair que cette complexité est considérée théorique. Nos expérimentations ont démontrées néanmoins que plus les seuils de densité sont élevés, moins il y aura de concepts denses. Également, en fonction de la nature de l'algorithme MAD-G, direct descendant de

MagaliceA, la modification des concepts et la création des concepts se limitent à un sous-treillis ou quelques sous-treillis de TCF. De plus, MAD-G vérifie si la condensation par densité est possible uniquement lorsque des concepts candidats sont à créer. Aucune vérification de la densité n'est faite au niveau de la modification des concepts. Ceci aura un effet positif sur la performance de MAD-G. Mais, elle aura un effet adverse sur la qualité des concepts denses soit : la condensation par densité composée. Ce sujet sera couvert au prochain chapitre ainsi qu'à la conclusion de ce mémoire.

8.9 Discussion

Ce chapitre a jeté les bases de la condensation des concepts utilisant un algorithme incrémental. Les objectifs et propriétés désirées ont été définies dont certaines de ces dernières sont devenues des contraintes, nommément les contraintes sur les concepts $\top$ et $\bot$, les singletons et la densité. Un premier algorithme a été présenté qui ne satisfait pas les propriétés émises. Le deuxième algorithme est la fonction estDense. Il satisfait les contraintes et les propriétés désirées du treillis tel que définies dans l'AFC. Afin d'utiliser cette fonction, l'algorithme MagaliceA a été modifié et est devenu MAD-G. Ce dernier calcule les concepts formels et les concepts denses. Pour obtenir les premiers, il suffit d'initialiser les seuils de densité à 0. De plus, MAD-G permet le calcul des générateurs. Deux questions demeurent ouvertes à ce jour :

- Comment caractériser les générateurs minimaux issus des concepts denses ? Sont-ils des δ-libres ou autres types de générateurs ?
- Doit-on caractériser le concept nouveau et le concept géniteur que la fonction estDense déclare comme éligible à la condensation par densité ?

Ce chapitre ne parle pas des avantages et des limites des algorithmes estDense et MAD-G. Ces sujets sont différés au prochain chapitre qui discutera des expérimentations et des résultats techniques.

CHAPITRE IX
EXPÉRIMENTATIONS ET RÉSULTATS

Ce chapitre présente les résultats expérimentaux des exécutions de MAD-G sur divers contextes. En dépit du fait que DR-MINER extrait difficilement les DRBS en utilisant juste la contrainte de densité, il sera utilisé dans certaines expérimentations pour juger la qualité des concepts denses extrait par MAD-G.

Le chapitre débute par une discussion sur les différences entre notre implémentation de DR-MINER et l'implémentation originale. La section 9.2 décrit brièvement l'environnement de test. La section 9.3 présente les tests et résultats sur des données synthétiques. Dans cette section, les résultats de DR-MINER et MAD-G sont comparés tant pour le nombre de concepts extraits que pour les temps d'exécution. Dans la section 9.4 DR-MINER et MAD-G sont testés sur les bases de données Mushrooms, Spambase et Lenses (UC Irvine Machine Learning Repository, 2007). Sauf pour le dernier jeu de données, les autres jeux sont trop volumineux pour extraire tous les concepts denses (ou formels) avec DR-MINER. Des sous-ensembles de ces jeux de données furent extraits afin de tester la génération des concepts et la performance entre les deux algorithmes. La section 9.5 présente des tests de capacité sur MAD-G utilisant les bases Mushrooms et Spambase. La première de ces deux bases est réputée dense, alors que la deuxième peut être qualifiée de creuse. Ce chapitre se termine par une discussion sur les avantages, les limites de MAD-G ainsi que sur la qualification ou non de la collection extraite par MAD-G comme représentation condensée ε-adéquate valide.

9.1 De notre implémentation de l'algorithme DR-MINER

Cette section présente l'algorithme DR-MINER selon (Besson, 2005) et (Besson, Robardet, & Boulicaut, 2006) et des différences entre l'implémentation originale et notre implémentation.

9.1.1 De l'algorithme DR-MINER

On se rappellera, tel que présenté au chapitre 7 de ce mémoire, que le but de l'algorithme DR-MINER est d'extraire les DRBS. Une version de l'algorithme DR-MINER basée sur les deux documents précédemment cités[74] est présentée ci-après.

Tableau 9.1 – Algorithme DR-MINER

Un ensemble d'objets O, un ensemble d'attributs A, un contexte formel $K = (O, A, I)$, une conjonction de contraintes monotones et anti-monotones sur $2^O \times 2^A$, les paramètres contrôlant la densité α,α^*, les paramètres contrôlant la pertinence δ, δ^* et un candidat $T = (Y, P, N)$.

```
DRMINER(T= (Y, P, N).)
(01)    SI P ≠ (∅, ∅) ALORS
(02)            Choisir un élément x (objet ou attribut) de P
(03)            T1 ← propager_contraintes(Y ∪ x, P\x, N)
(04)            SI estConsistant(T1) ALORS
(05)                    DRMINER(T1)
(06)            T2 ← propager_contraintes(Y, P\x, N∪ x)
(07)            SI estConsistant(T2)ALORS
(08)                    DRMINER(T2)
(09)    SINON sauvegarder Y
FIN DRMINER
```

DR-MINER débute son exécution avec l'espace de recherche utilisant le candidat $((\varnothing, \varnothing), (O, A), ((\varnothing, \varnothing))$. La définition suivante concerne le candidat utilisé dans DR-MINER.

[74] Il y a de légères différences dans les pseudo-codes de DR-MINER présentés dans la thèse de doctorat et l'article scientifique de J. Besson. Ces différences n'affectent aucunement les définitions de bases des DRBS et de l'algorithme DR-MINER. Dans les faits, notre implémentation de DR-MINER fut réalisée à l'aide des deux pseudo-codes.

Définition 9.1 – Candidat (*Y, P, N*)

Un candidat est un triplet (*Y, P, N*) de trois bi-ensembles. $Y = (Y_O, Y_A)$ contient les éléments (objets et attributs) appartenant au candidat et à tous ses fils. $N = (N_O, N_A)$ contient les éléments (objets et attributs) qui n'appartiennent pas au candidat et à ses fils. $P = (P_O, P_A)$ contient les éléments non encore considérés pour une itération donnée[75].

L'algorithme débute en choisissant un élément de O ou A du bi-ensemble P. Il le place dans le bi-ensemble Y pour ensuite propager les contraintes sur la densité et la pertinence. Une fois ces contraintes satisfaites, le candidat T_1 doit satisfaire les contraintes de consistances et les autres contraintes (e.g. fréquence, maximalité). Si c'est le cas, DR-MINER est appelé récursivement sur T_1 sinon l'élément est déplacé dans le bi-ensemble N et le processus se répète pour le candidat T_2[76]. Une fois le bi-ensemble P vidé de ses éléments, le bi-ensemble Y est un DRBS valide en fonction des contraintes utilisées et est stocké dans la collection de DRBS. Le traitement reprendra pour le prochain élément de O et A. L'algorithme terminera lorsque tous les candidats seront traités.

L'espace de recherche produit par DR-MINER est un arbre d'énumération des candidats. La figure suivante présente un arbre d'énumération partiel.

[75] Y pour YES bi-set, N pour NO bi-set et P pour POTENTIAL bi-set.

[76] Le lecteur intéressé aux définitions exactes au sujet de la propagation des contraintes et de la vérification de la consistance peut consulter le document (Besson, 2005).

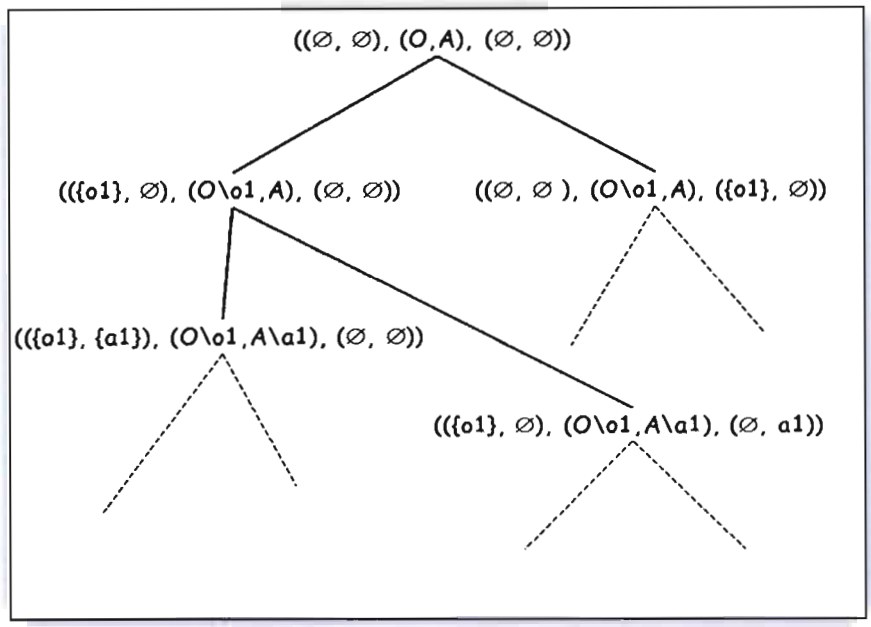


Figure 9.1 – Arbre d'énumération produit par DR-MINER

L'algorithme DR-MINER peut calculer tous les concepts formels, du moment que les valeurs des paramètres contrôlant la densité sont à 0 et que les valeurs des paramètres contrôlant la pertinence sont à 1. Si ces derniers demeurent à 1 et que les valeurs des seuils de densité sont plus grandes que zéro, DR-MINER calculera les concepts denses. Dans les cas où les valeurs des paramètres de densité sont à 0 et que les valeurs des seuils de pertinence sont plus grandes que 1, DR-MINER calculera uniquement les concepts pertinents. En dernier lieu, DR-MINER peut calculer les DRBS, soient les concepts denses et pertinents, selon les valeurs fournies aux paramètres de densité et de pertinence. Notons également que DR-MINER calcule l'iceberg de DRBS.

Revenons sur l'arbre d'énumération des candidats servant à calculer les DRBS. On peut facilement noter que la taille de l'arbre d'énumération est fonction de la taille des ensembles O et A. Dans des contextes formels très denses cela supposera que l'arbre d'énumération sera quasi complet. Autrement dit, DR-MINER calculera tous les sous-ensembles possibles sur les ensembles O et A. On sait également que le calcul de l'ensemble

de tous les sous-ensembles est de l'ordre de l'exponentiel. Il est clair que DR-MINER ne calcule pas l'ensemble de tous les sous-ensembles d'un ensemble mais, dans ce cas-ci, il calcule l'ensemble de tous les sous-ensembles de $O \times A$. Dès lors, on sait pertinemment que l'extraction des DRBS sera difficile (Boulicault & Besson, 2008).

Il est vrai qu'en utilisant la contrainte de pertinence couplée aux contraintes de fréquence minimal et de la contrainte de maximalité des DRBS, on parvient presque toujours à extraire les DRBS. Cependant, il faut se rappeler que l'algorithme MAD-G calcule uniquement les concepts formels et denses. Pourquoi donc avoir choisi DR-MINER comme algorithme pour la comparaison de nos mesures ? Tout simplement parce que DR-MINER calcule parfaitement les concepts formels. De plus, avec un bon choix des seuils de densité et de fréquence minimal, les DRBS sont de grande qualité (Besson, 2005). En dernier lieu, à notre connaissance DR-MINER est le seul algorithme respectant les principes de l'AFC pour l'extraction des DRBS. Ce dernier point a pesé lourd dans la décision d'utiliser DR-MINER. Il dépasse même celle de la performance dans le cadre de nos recherches. Des mesures de performance entre les deux algorithmes sont présentées dans ce chapitre. Il ne faudra pas se surprendre que, dans la plupart des cas, MAD-G est plus performant. On doit cependant garder à l'esprit que la qualité des concepts denses extraient par MAD-G est une mesure tout aussi importante. Les DRBS extraits par DR-MINER serviront de standards comparatifs avec les concepts denses extraient sous MAD-G.

9.1.2 De notre réalisation de l'algorithme DR-MINER

Dans cette sous-section, on note certaines différences entre notre réalisation de l'algorithme DR-MINER et la réalisation originale.

À la conférence ICFCA 2008 à Montréal, M. J.F. Boulicaut nous a confirmé, lors d'une conversation privée, que DR-MINER a été écrit en langage C\C++. L'une des difficultés de ce langage est que le développeur est responsable de l'allocation mémoire pour les variables et les structures de données. Bien que ce ne soit pas une tâche facile pour tous

les développeurs, il est possible, pour le programmeur chevronné, d'aller chercher de la performance en temps et en mémoire. Notre implémentation de DR-MINER est en Java. Ce langage permet aux développeurs de ne pas se soucier de la gestion de l'allocation mémoire pour les structures de données. Ainsi le programmeur peut se concentrer sur l'algorithme. Lorsque des structures de données sont libérées, le ramasse-miettes de Java prend en charge de libérer la mémoire. Malheureusement, personne n'a la main mise sur le ramasse-miettes, sur le quand il accomplira sa tâche ni sur le comment il l'accomplira. Dès lors, il ne faudra pas se surprendre si certains de nos résultats semblent moins performants que les mesures prises par la réalisation originale de DR-MINER (Besson, Robardet, & Boulicaut, 2006).

Nous avons utilisé l'heuristique proposée dans (Besson, Robardet, & Boulicaut, 2006) permettant d'élaguer le plus possible des éléments lors de la propagation des contraintes. Dès lors, les représentations horizontale et verticale furent triées de manière ascendante sur la taille de l'ensemble des attributs dans le cas de la représentation horizontale et sur la taille de l'ensemble des objets pour la représentation verticale. Dans notre implémentation de DR-MINER nous les passons en paramètres afin de conserver les éléments des deux représentations lors du retour d'un appel récursif. Ceci a pour effet de copier pour chaque appel récursif ces deux structures, dès lors un coût supplémentaire sur la mémoire.

Afin de rendre l'exécution de la vérification de la consistance des candidats plus rapide, des représentations horizontale et verticale en ordre lexicographique furent créées. Heureusement, ces deux représentations sont statiques en mémoire. Il est à noter que les quatre représentations ont été prétraitées avant l'exécution de DR-MINER.

Nous avons préféré pousser le plus de contraintes possibles dans la procédure responsable de la propagation des contraintes. L'idée est de limiter le nombre d'appels récursifs pour ainsi économiser la mémoire. C'est un compromis entre le temps d'exécution et la disponibilité de la mémoire.

En dernier lieu, l'algorithme DR-MINER produit des doublons. On peut faire un post-traitement pour les éliminer. Nous avons préféré faire l'élimination des doublons à l'intérieur de la procédure responsable du stockage de DRBS.

9.2 Environnement de test

L'environnement de test a les caractéristiques suivantes.

- PC avec micro-processeur Duo-CORE 2.1 GHz
- Mémoire vive à 3.3 GB
- Système d'exploitation Windows XP
- Langage de programmation Java 1.6
- Environnement de programmation NetBeans 6.0

Les programmes de MAD-G ont été développés à partir du code des programmes de MagaliceA version autonome.

9.3 Mesures sur des contextes formels synthétiques

Cette expérimentation a pour but d'évaluer si DR-MINER et MAD-G retrouvent dans des contextes formels bruités les concepts originaux projetés. Les jeux de données construits sont trois concepts disjoints de 10 objets et 5 attributs, donc un contexte formel de 30 objets par 15 attributs. DR-MINER et MAD-G ont été exécuté sur ce contexte n'ayant aucune exception. Les résultats de ce test sont donnés ci-dessous.

Algorithme	#concepts	Temps (sec.)
DR-MINER	5	60
MAD-G	5	0

Le but est donc de trouver si possible, à l'aide des 2 algorithmes, les 5 concepts formels d'origine dans des contextes bruités. Pour ce faire 5 jeux d'essais ont été générés pour chaque niveau de bruit.

Nous tenons à rappeler au lecteur que l'objectif de ces expériences est de vérifier la capacité de condensation de MAD-G versus DR-MINER. Les temps d'exécution de DR-MINER seront bien entendu plus élevés que ceux de MAD-G. Ils sont inclus dans les résultats qu'à titre informationnel seulement.

Deux définitions informelles sont proposées en ce qui concernent la qualité des concepts denses (ou DRBS selon le cas).

Définition 9.2 - Saturation

Si on augmente la valeur des seuils de densité et que la valeur des concepts denses ne change pas, on dit qu'il y a saturation.

Définition 9.3 - Dégénérescence

Si des éléments sur l'intension et/ou sur l'extension d'un concept dense appartiennent à 2 concepts disjoints, on dit qu'il y a dégénérescence du concept dense.

La dernière définition signifie que dans le cas des données synthétiques, le bruit ajouté aux concepts disjoints se retrouve à l'intérieur de chacune des "partitions" des concepts. Dès lors, si un algorithme crée un concept dense avec des valeurs de 2 concepts disjoints, ce concept est dégénéré puisqu'il dépasse les bornes de la "partition" qui lui est assigné. La figure suivante illustre des concepts disjoints non dégénérés.

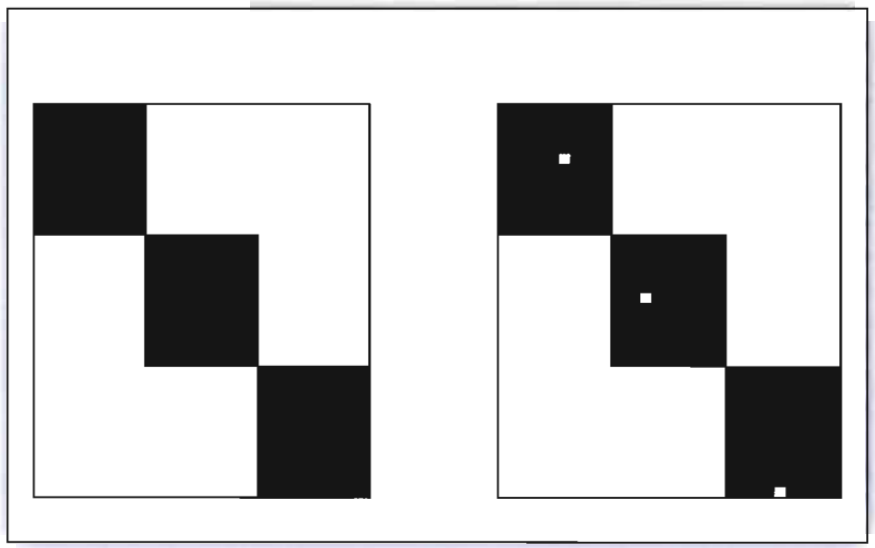


Figure 9.2 – Contexte sans exception (à gauche), avec exception (à droite)

9.3.1 Test contexte formel 30 × 15 avec 2% de bruit

DR-MINER	α	α*	Moy. du # de concepts	E.T.[77]	Temps moyen (sec.)	E.T. (sec.)
	0	0	8.0	0.00	46.0	0.00
	1	1	8.0	0.00	23.0	0.00
	2	2	5.0	0.00	69.0	0.00

MAD-G	α	α*	Moy. du # de concepts	E.T.	Temps moyen (sec.)	E.T. (sec.)
	0	0	8.0	0.00	0.0	0.00
	1	1	5.0	0.00	0.0	0.00

Les deux algorithmes sont en mesure de trouver exactement les cinq concepts recherchés. Un fait intéressant est présenté dans les résultats entre MAD-G et DR-MINER. Ce dernier retrouve les cinq concepts d'origine avec des seuils de densité de 2, alors que MAD-G retrouve les mêmes concepts avec des seuils de densité de 1.

[77] E.T. : Écart type

9.3.2 Test contexte formel 30 × 15 avec 5% de bruit

DR-MINER	α	α*	Moy. du # de concepts	E.T.	Temps moyen (sec.)	E.T. (sec.)
	0	0	12.0	1.72	25.0	0.00
	1	1	6.0	0.86	18.0	1.32
	2	2	5.0	0.86	37.0	7.66
	3	3	5.0	0.00	295.0	48.51

MAD-G	α	α*	Moy. du # de concepts	E.T.	Temps moyen (sec.)	E.T. (sec.)
	0	0	12.0	1.72	0.0	0.00
	1	1	6.0	0.00	0.0	0.00
	2	2	5.0	0.00	0.0	0.00

Pour des $\alpha = \alpha^* = 2$, DR-MINER parvient à retrouver exactement les 5 concepts d'origine mais pas pour tous les contextes formels générés. Par contre, lorsque les valeurs de ces paramètres sont à 3, DR-MINER calcule exactement les concepts voulus pour tous ces contextes formels. MAD-G trouve les 5 concepts d'origine pour $\alpha = \alpha^* = 2$.

9.3.3 Test contexte formel 30 × 15 avec 10% de bruit

DR-MINER	α	α*	Moy. du # de concepts	E.T.	Temps moyen (sec.)	E.T. (sec.)
	0	0	30.2	6.02	3.23	2.37
	1	1	19.8	4.30	6.30	1.32
	2	2	11.0	2.58	15.55	4.63
	3	3	6.0	0.43	83.16	19.87
	4	4	5.0	0.00	360.36	39.73

MAD-G	α	α*	Moy. du # de concepts	E.T.	Temps moyen (sec.)	E.T. (sec.)
	0	0	30.2	6.02	0.0	0.00
	1	1	19.8	4.30	0.0	0.00
	2	2	7.4	0.93	0.0	0.00
	3	3	6.8	0.86	0.0	0.00

Avec des seuils de densité à 4, DR-MINER parvient à extraire les cinq concepts recherchés. Dans le cas de MAD-G, avec des seuils de densité à 3, extrait en moyenne 6.8 concepts plutôt que 5. Pour des seuils de densité plus grands que 3 MAD-G calcule 6.8 concepts en moyenne. Il y a donc saturation pour MAD-G à $\alpha = \alpha^* = 3$. Il est à noter que les concepts denses extraits par MAD-G ne sont pas dégénérés.

9.3.4 Test contexte formel 30 × 15 avec 20% de bruit

DR-MINER	α	α^*	Moy. du # de concepts	E.T.	Temps moyen (sec.)	E.T. (sec.)
	0	0	44.8	10.75	0.0	0.00
	1	1	22.4	1.29	4.0	0.66
	2	2	12.8	0.86	7.2	0.99
	3	3	5.6	0.86	40.9	3.31

MAD-G	α	α^*	Moy. du # de concepts	E.T.	Temps moyen (sec.)	E.T. (sec.)
	0	0	44.8	10.75	0.0	0.00
	1	1	32.0	4.30	0.0	0.00
	2	2	16.2	2.58	0.0	0.00
	3	3	11.6	1.29	0.0	0.00
	4	4	10.6	2.15	0.0	0.00

L'écart se creuse entre le nombre de concepts denses extrait par DR-MINER et MAD-G. DR-MINER parvient à créer en moyenne une collection de 5.6 concepts denses (ou DRBS) avec des seuils $\alpha = \alpha^* = 3$. Ces concepts denses sont presque de bonne qualité, c'est à dire qu'il y a un faible chevauchement sur les concepts disjoints d'origine. Si les seuils α et α^* dépassent 3, les concepts sont dégénérés.

MAD-G parvient à extraire avec des seuils $\alpha = \alpha^* = 4$ une collection de 10.6 concepts denses en moyenne. Aucun des concepts denses chevauchent les partitions des concepts disjoints. On peut dire que ce sont des concepts denses de qualité puisqu'ils respectent les partitions du contexte formel. Il y a saturation sur les valeurs des seuils α et α^*

dépassant 4. L'une des raisons pour laquelle MAD-G ne peut pousser la condensation par densité est le respect de la contrainte des singletons.

9.3.5 Test contexte formel 30 × 15 avec 30% de bruit

DR-MINER	α	α*	Moy. du # de concepts	E.T.	Temps moyen (sec.)	E.T. (sec.)
	0	0	59.2	5.59	0.0	0.00
	1	1	21.0	4.30	0.0	0.00
	2	2	10.8	1.29	4.0	0.00
	3	3	7.8	2.15	28.5	10.24

MAD-G	α	α*	Moy. du # de concepts	E.T.	Temps moyen (sec.)	E.T. (sec.)
	0	0	59.2	5.59	0.0	0.00
	1	1	41.4	9.03	0.0	0.00
	2	2	29.2	4.30	0.0	0.00
	3	3	20.6	3.44	0.0	0.00
	4	4	18.0	1.29	0.0	0.00

À partir d'un contexte formel bruité à 30%, on remarque la supériorité de la condensation de DR-MINER sur MAD-G. On note également que pour de faibles valeurs de seuils de densité, l'extraction des concepts denses par DR-MINER est très performantes dans des contextes formels considérés creux. Cependant, plus les seuils de densité sont élevés plus l'extraction produite par DR-MINER perd en performance. On peut tenter d'expliquer ce fait. DR-MINER considère des exceptions comme des cas valides lors de la construction d'un concept, on se trouve donc à produire des concepts denses à partir de contextes formels "virtuellement" plus denses que creux. Pour des seuils $\alpha = \alpha^* = 2$, DR-MINER extrait des concepts denses respectant les partitions des concepts disjoints d'origine. Avec $\alpha = \alpha^* = 3$, DR-MINER produit quelques concepts denses légèrement dégénérés. Au delà de ces seuils, les concepts denses se dégénèrent rapidement perdant ainsi leur valeur informative.

À partir de ce point, la contrainte des singletons imposée dans l'algorithme MAD-G limite sérieusement la condensation par densité des concepts. Il n'en demeure pas moins, outre la performance, que les concepts denses extraits ne sont pas dégénérés.

9.3.6 Test contexte formel 30 × 15 avec 50% de bruit

DR-MINER	α	α*	Moy. du # de concepts	E.T.	Temps moyen (sec.)	E.T. (sec.)
	0	0	46.8	7.74	0.0	0.00
	1	1	11.0	3.01	0.0	0.00
	2	2	10.4	4.73	0.0	0.00
	3	3	8.4	4.30	0.0	0.00

MAD-G	α	α*	Moy. du # de concepts	E.T.	Temps moyen (sec.)	E.T. (sec.)
	0	0	46.8	7.74	0.0	0.00
	1	1	43.4	8.17	0.0	0.00
	2	2	38.6	6.45	0.0	0.00
	3	3	37.0	5.16	0.0	0.00
	4	4	36.2	4.30	0.0	0.00

Pour des contextes formels avec un bruit de 50%, DR-MINER atteint sa pleine performance. Cependant, pour les seuils de densité $\alpha = \alpha^* = 3$, les concepts extraits sont dégénérés en tout ou en partie. Donc, pour obtenir une collection de concepts denses de bonne qualité, il serait préférable d'utiliser la collection extraite avec $\alpha = \alpha^* = 2$. Même si cette collection contient en moyenne 10.4 concepts plutôt que 8.4 pour des $\alpha = \alpha^* = 3$, elle demeure plus petite que la collection de concepts denses que MAD-G a extraite avec les seuils de densité à 4.

9.3.7 Observations

On remarque que MAD-G parvient à extraire des concepts denses très rapidement. Pour des contextes formels "partitionnés" faiblement bruités, MAD-G parvient à extraire des concepts denses très près des concepts disjoints recherchés. Plus le bruit augmente dans les partitions moins MAD-G parvient à extraire des concepts denses se rapprochant des concepts disjoints recherchés. Comme précédemment noté, l'une des raisons de ce rapprochement est le respect de la contrainte sur les singletons[78]. Autrement dit, MAD-G semble respecter ces

[78] Il y a sûrement d'autres raisons. Ceci mériterait une investigation!

partitions lors de l'extraction des concepts denses. Ceci pourrait être un avantage et ce en dépit du fait que DR-MINER extrait des concepts denses en moins grand nombre que MAD-G. Admettons que MAD-G extrait des concepts denses d'un contexte bruité mais partitionné. Une fois les concepts denses extraits, un post-traitement pourrait être appliqué sur les concepts denses afin de fusionner ceux qui répondent à la contrainte de densité tout en abandonnant, dans ce post-traitement, la contrainte des singletons.

9.4 Mesure sur Mushrooms, Spambase et Lenses

Les jeux de données Mushrooms, Spambase et Lenses de (UC Irvine Machine Learning Repository, 2007) sont utilisés pour mesurer la condensation par densité et la performance entre MAD-G et DR-MINER. Étant donné que l'extraction des DRBS avec DR-MINER sans l'utilisation des seuils de pertinence est difficile, un sous-ensemble de transactions des deux premières bases a été extrait.

9.4.1 Concepts denses avec Spambase

Spambase est un jeu de données de 4600 objets et 57 attributs, ces derniers sont des valeurs réelles. Il y a beaucoup de valeurs à zéro dans les transactions ce qui en fera, lors de sa binarisation, un contexte formel creux. Il fut décidé que seuls les attributs, ayant une valeur plus grande ou égale à 1.0, seront conservés. Afin de rendre possible l'extraction par DR-MINER un sous-ensemble de 75 transactions fut extrait de Spambase.

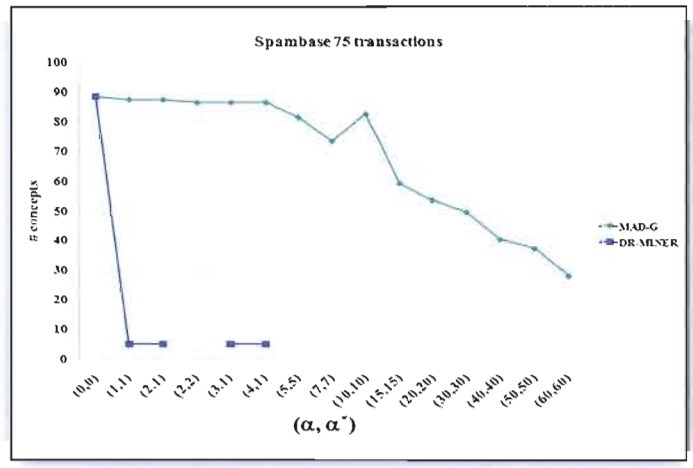


Figure 9.3 – Concepts denses sur Spambase avec DR-MINER et MAD-G

La figure ci-dessus illustre bien la grande différence entre la condensation par densité produite par DR-MINER par rapport à MAD-G. Il y a saturation pour MAD-G lorsque les seuils $\alpha = \alpha^* = 60$ et le nombre de concepts denses est de 28. Alors que DR-MINER, avec les seuils $\alpha = \alpha^* = 1$, extrait 5 concepts denses. La figure suivante présente les temps d'exécution. Il est clair que MAD-G est plus rapide que DR-MINER.

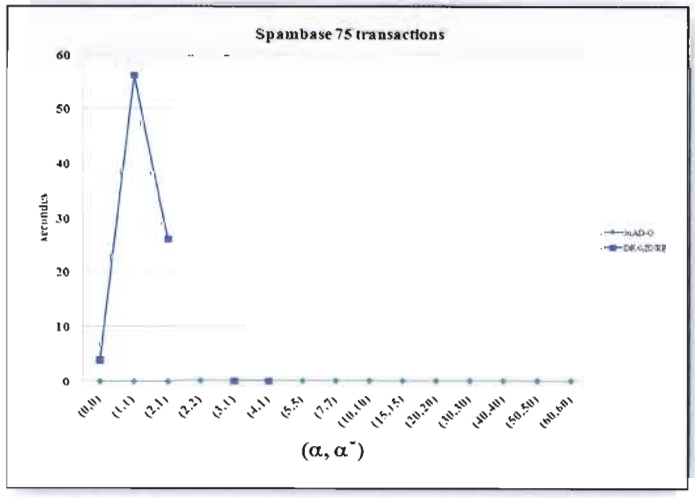


Figure 9.4 – Performance entre DR-MINER et MAD-G sur Spambase

9.4.2 Concepts denses avec Mushrooms

La figure 9.5 présente l'extraction de concepts denses pour un contexte formel de Mushrooms ayant 200 transactions. Encore une fois le nombre de concepts denses extraits par DR-MINER est beaucoup plus petit que celui de MAD-G. DR-MINER pour des seuils $\alpha = 4$ et $\alpha^* = 1$ extrait 3 concepts denses. MAD-G, pour des seuils de $\alpha = \alpha^* = 200$, extrait 9 concepts. Par contre, pour ce qui est de la performance, MAD-G est beaucoup plus performant que DR-MINER comme l'indique la figure 9.6. Dans les faits, le temps d'extraction de DR-MINER pour des seuils $\alpha = \alpha^* = 0$ n'est pas inclus dans le graphe parce que ce temps dépasse les 30 minutes.

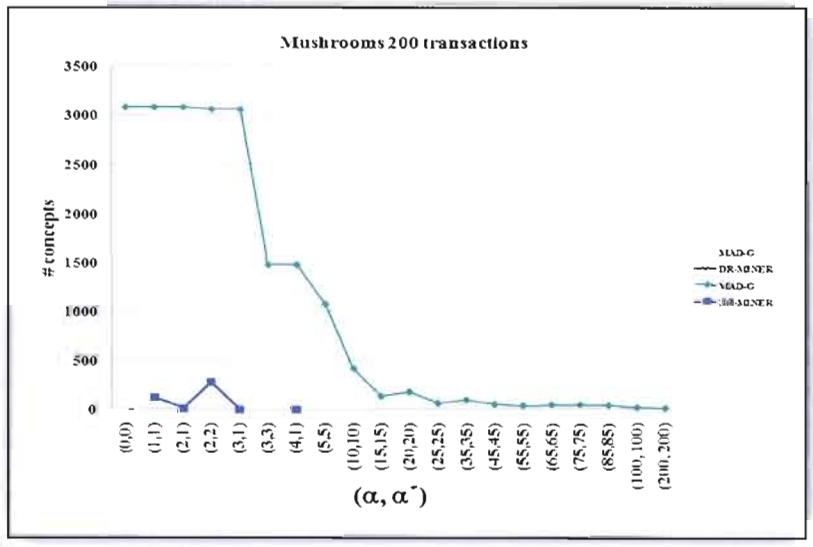


Figure 9.5 – Concepts denses sur Mushrooms avec DR-MINER et MAD-G

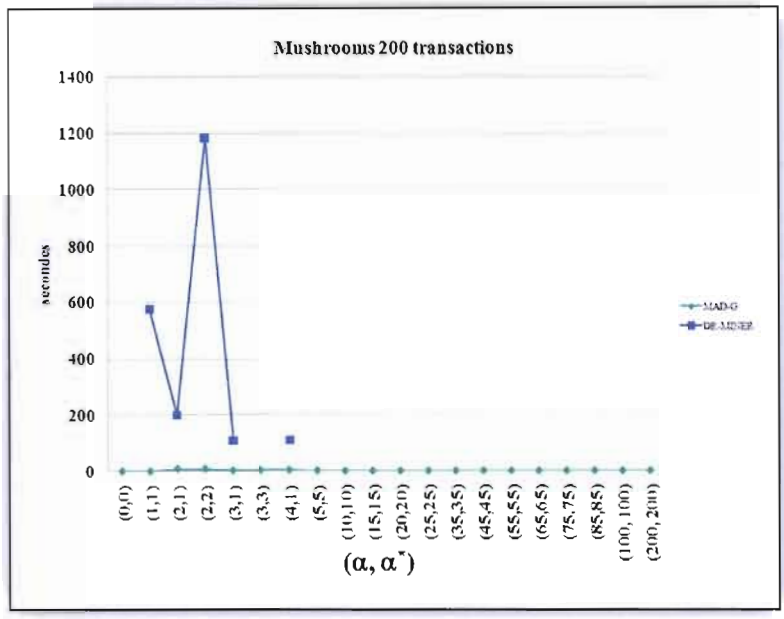


Figure 9.6 – Performance entre DR-MINER et MAD-G sur Mushrooms

9.4.3 Iceberg de concepts denses avec Mushrooms

Le prochain test consiste à incorporer un seuil minimal de fréquence. De Mushrooms 1000 transactions furent extraites. Un seuil de fréquence à 7% déterminera l'extraction des concepts denses.

Encore une fois DR-MINER produit moins de concepts denses (sauf lorsque les seuils de densité sont à 0) que MAD-G. Un fait intéressant est à noter. DR-MINER, pour des seuils de densité à 2 ou plus, produit qu'un seul concept (voir figure 9.7) soit le concept $\top$ de l'ICD. Est-ce que ça reflète bien la réalité des données ? Ceci pourrait mériter une investigation[79]. Pour ce qui est de MAD-G, on remarque que pour des seuils de densité variant de 1 à 20, il semble avoir un plafonnement dans la condensation. Ceci pourrait nous amener à conclure que le point de saturation a été atteint, mais ce n'est pas le cas comme

[79] Nous n'avons pas eu le temps d'une telle investigation étant donné que les algorithmes traitent une base de 1000 transactions.

l'illustre la figure 9.7. Dès que les paramètres de densité ont des valeurs plus grandes que 20, le nombre de concepts denses diminue grandement.

Pour ce qui est la performance, MAD-G se démarque encore. Pour toute les extractions, indépendamment des seuils de densité choisis, le temps d'extraction des concepts denses est constant, c'est à dire 0 (figure 9.8).

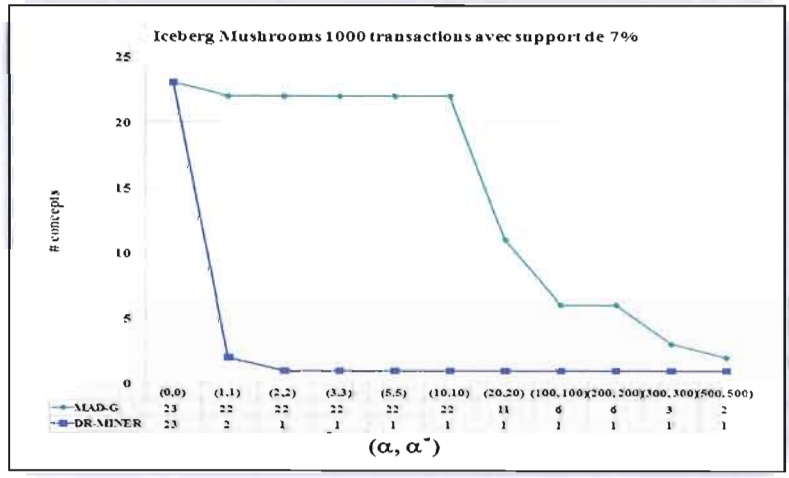


Figure 9.7 – Extraction de l'ICD de Mushrooms par DR-MINER et MAD-G

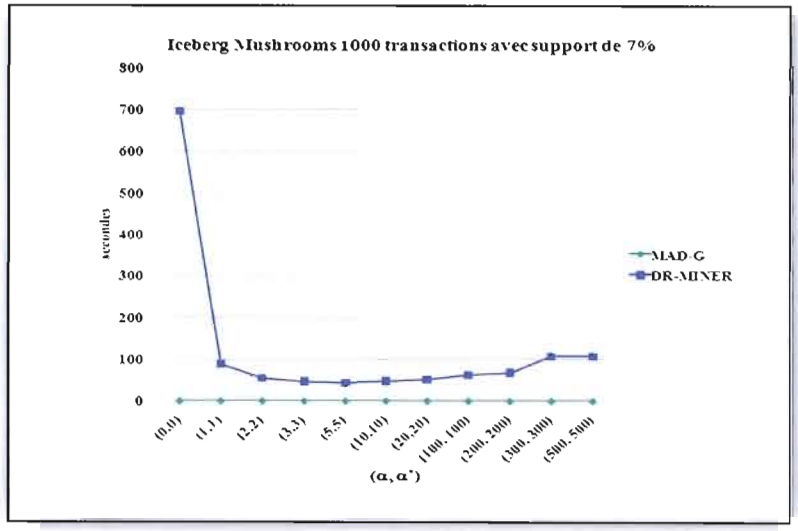


Figure 9.8 – Performance entre DR-MINER et MAD-G sur l'ICD de Mushrooms

9.4.4 Concepts denses avec Lenses

(Besson, 2005) a utilisé la base de données Lenses pour ses tests de condensation. C'est une petite base de 24 objets par 11 attributs[80]. C'est une base intéressante car elle permet à DR-MINER d'extraire les DRBS (ou concepts denses) très rapidement. La figure 9.9 suivante présente la condensation par densité pour les deux algorithmes.

Lorsque les seuils de densité $\alpha = \alpha^* = 1$, DR-MINER calcule 17 concepts denses alors que MAD-G calcule 97 concepts denses. L'écart se rétrécit lorsque les seuils de densité atteignent 6. Il n'en demeure pas moins que MAD-G calcule en moyenne 10 concepts denses de plus que DR-MINER à partir du seuil de densité 6. Lorsqu'un seuil de densité de 24 est utilisé, DR-MINER calcule un seul concept dense alors que MAD-G calcule 13 concepts denses. Dans les faits, l'extraction par MAD-G est saturé lorsque pour des seuils de densité plus grands ou égales à 10. L'explication est toujours la même pour MAD-G. La contrainte des singletons empêche de réduire la taille de la collection de concepts denses. DR-MINER n'a évidemment pas ce problème. La figure 9.10 présente la performance de l'exécution des deux algorithmes.

[80] Dans (Besson, 2005), elle est de 24 × 12. Ceci est probablement dû à l'un des champs possédant une valeur astigmate ou non-astigmate et qui fut converti en deux champs pour le contexte formel. Nous avons préféré convertir ce champ en un seul attribut pour le contexte formel qui aura deux valeurs soient : vrai (ou présent) si le patient est astigmate, faux (ou absent) sinon.

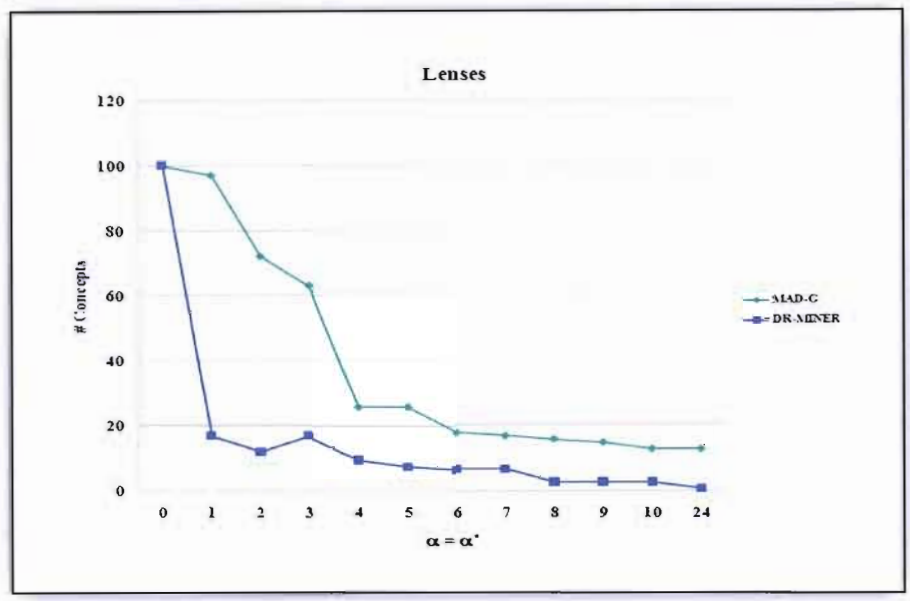


Figure 9.9 – Concepts denses de Lenses avec DR-MINER et MAD-G

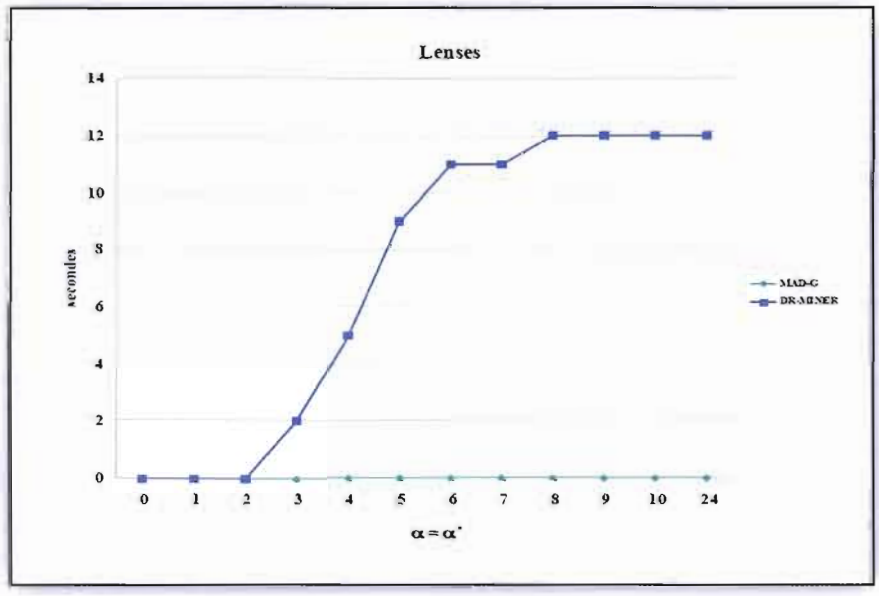


Figure 9.10 – Performance entre DR-MINER et MAD-G sur Lenses

9.4.5 Consistance des concepts denses

Les bases de données Mushrooms et Spambase sont trop volumineuses pour examiner si les concepts denses extraits par MAD-G sont consistants avec les seuils de densité. On a cependant une chance inespérée de vérifier la consistance des concepts denses avec la base de données Lenses.

Les concepts denses issus de l'extraction produite par MAD-G avec les seuils de densité $\alpha = \alpha^{*} = 10$ ont été comparés avec des résultats attendus calculés manuellement sur le jeu de données Lenses. Des 13 concepts denses extraits, 5 concepts denses ne respectent pas les seuils de densité choisis. On peut dès lors douter de la consistance de certaines collections extraites jusqu'ici par MAD-G. On reviendra sur ce sujet à la section 9.6 de ce chapitre.

9.5 Tests de capacité sur MAD-G

Des tests de capacité sur les bases Spambase et Mushrooms ont été effectués sur MAD-G incluant dans certains cas l'extraction des générateurs. Le tableau suivant présente l'extraction des concepts denses sur le contexte formel de Mushrooms pour des valeurs de seuils de densité allant de 0 à 8000. On remarquera encore une fois que le temps d'exécution diminue rapidement avec des valeurs élevées de seuils de densité.

Tableau 9.2 – Concepts denses sur Mushrooms (8128 transactions) par MAD-G

Mushrooms – 8128 transactions × 127 attributs		
$\alpha = \alpha^*$	**# Concepts**	**Temps d'exécution (secondes)**
0	238710	3372
50	70475	3973
100	29434	1020
250	10190	285
500	5138	100
750	1349	25
1000	1187	18
2000	113	2
4000	16	0
6000	24	0
8000	9	0

La figure suivante illustre l'extraction des concepts denses par MAD-G sur Spambase avec des seuils de densité de 0 à 4000.

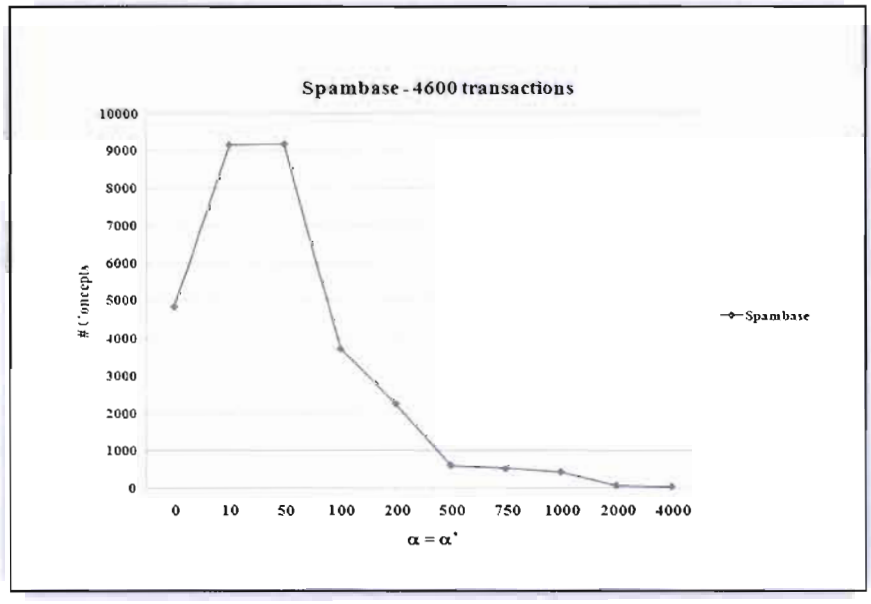


Figure 9.11 – Concepts denses sur Spambase (4600 transactions) avec MAD-G

L'extraction des concepts denses sur Spambase présente une particularité intéressante. On voit que pour les valeurs de seuils de densité 10 et 50 l'extraction produit un plus grand nombre de concepts denses que de concepts formels. Cette particularité n'est pas unique à MAD-G seulement, il a été remarqué avec DR-MINER également. La figure suivante présente les temps d'exécution de MAD-G sur l'extraction des concepts denses avec et sans les générateurs minimaux. On remarquera qu'il n'y a pas de valeurs pour les seuils de densité 10, 50, 100 et 200 pour l'extraction des concepts denses et des générateurs minimaux. Pour ces valeurs le temps d'exécution dépassait une heure[81]. L'intuition première fut que le temps d'exécution de l'extraction des concepts denses et de leurs générateurs minimaux serait à peine plus élevé que l'extraction sans générateurs minimaux. Dans le cas de Spambase, cette intuition s'est avérée fausse.

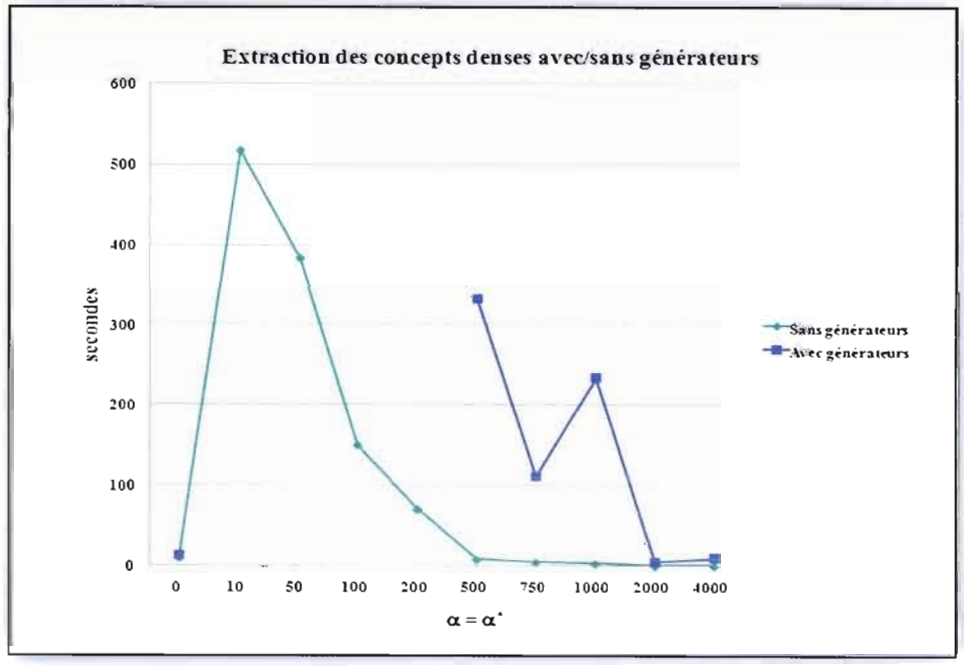


Figure 9.12 – Performance de MAD-G sur Spambase avec/sans générateurs minimaux

[81] Pour le seuil de densité 50, MAD-G n'avait toujours pas fini l'extraction des concepts denses et des générateurs minimaux après 4 heures d'exécution.

Serait-il possible que Spambase soit une exception à la règle ? Pour valider si Spambase est une exception il fut décidé de tester l'extraction des concepts denses et des générateurs sur Mushrooms. On sait qu'à partir de Mushrooms on extrait plus de 200 000 concepts formels, donc extraire les générateurs pourrait être une lourde tâche. Par conséquent, seules les transactions dont les champignons sont vénéneux ont été retenues pour ce test[82]. La figure 9.13 présente l'extraction des concepts denses par MAD-G sur la base Mushrooms ayant 3916 transactions. La figure 9.14 présente les temps d'exécution. On remarque que l'extraction des concepts denses et des générateurs minimaux pour les seuils de densité 2, 5 et 10 est difficile[83].

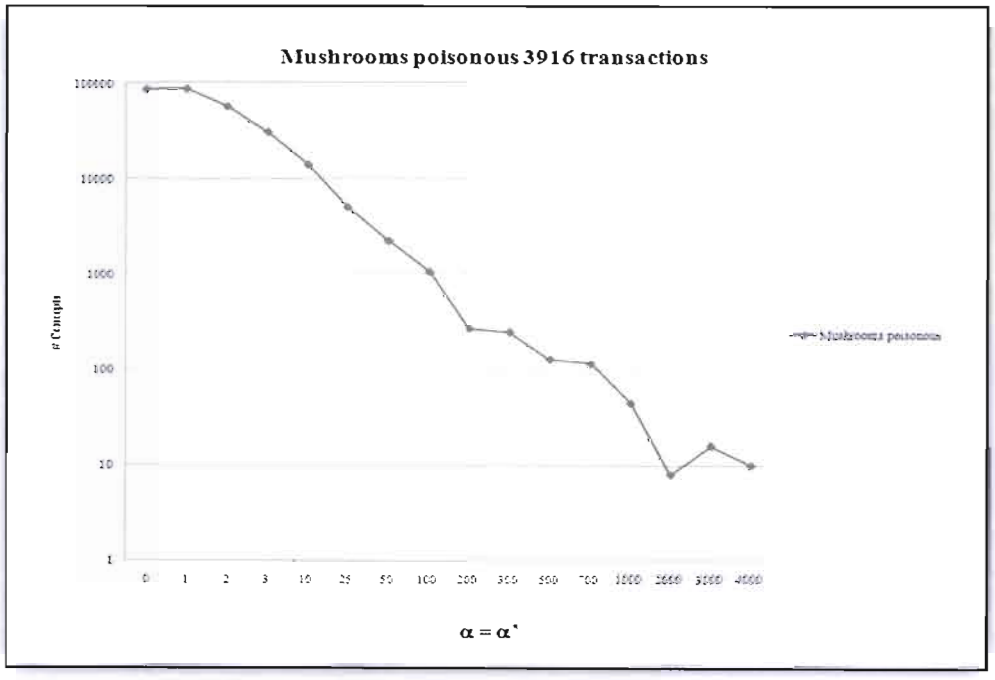


Figure 9.13 – Concepts denses par MAD-G sur 3916 transactions de Mushrooms

[82] C'est à dire 3916 transactions. Dans (Besson, Robardet, & Boulicaut, 2006), ils ont décidé de faire de même pour valider les principes de pertinence et de densité des DRBS.

[83] Après plus d'une heure l'extraction n'était toujours pas terminée.

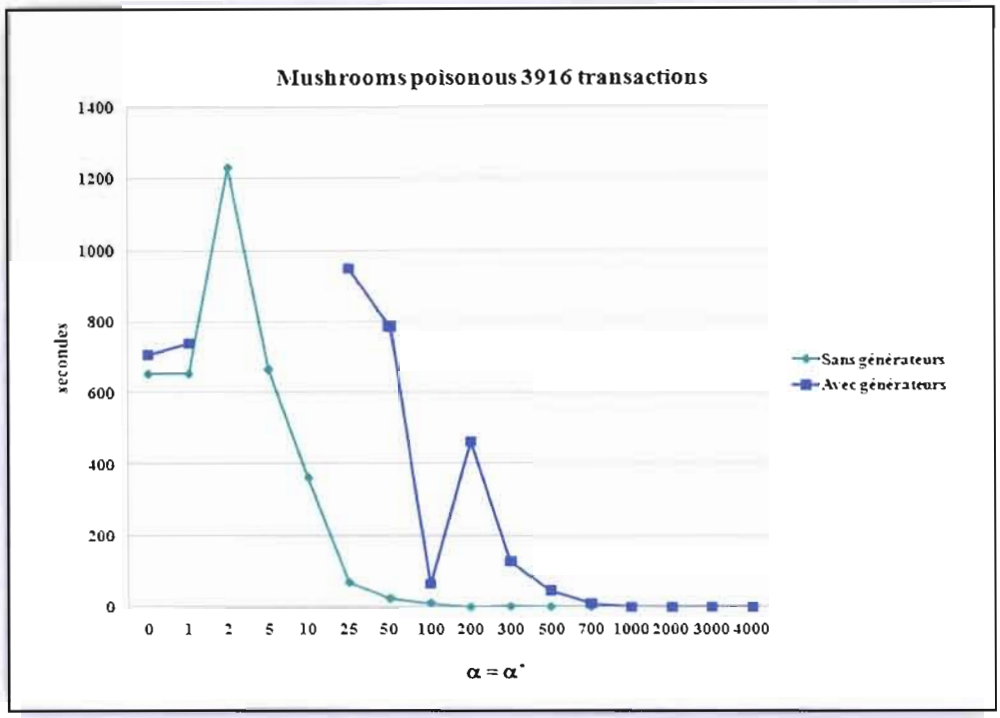


Figure 9.14 – Performance sur MAD-G sur Mushrooms (3916 transactions)

Que conclure à cet effet ? L'extraction des générateurs minimaux a bel et bien un impact sur lors de l'extraction des concepts formels, mais c'est un impact minime dans la plupart des cas comme on peut le voir dans les figures précédentes. Que se passe-t-il donc lors de l'extraction des concepts denses ? Notre seule hypothèse est que la boucle interne du programme principal (voir tableau 8.4 du chapitre 8) tourne plus souvent pour certaines valeurs de seuil de densité que pour d'autres car le nombre de fusions est plus élevé. Les temps d'exécution sur les icebergs de concepts denses munis de générateurs minimaux sont beaucoup plus acceptables. Naturellement le seuil de fréquence minimal diminue grandement le nombre de concepts denses (ou formels) permettant une économie de temps appréciable tant pour les concepts denses sans générateurs minimaux que pour les concepts denses et générateurs minimaux.

9.6 Discussion

9.6.1 Avantages de MAD-G

Bien que MAD-G extrait plus de concepts denses que DR-MINER, il n'en demeure pas moins qu'il les extrait et que la collection de concepts denses selon les seuils de densité choisi est, dans la plupart des cas, plus petite que la collection de concepts formels. Par exemple, au tableau 9.2, pour un seuil de densité à 50 on extrait 70475 concepts denses. Il y a une réduction de 70% sur la taille de la collection des concepts formels. L'avantage ultime de MAD-G par rapport à DR-MINER est la vitesse d'exécution. Il fut possible de calculer les concepts denses dans des temps respectables là où il était pratiquement impossible de le faire avec DR-MINER. Il faut rappeler cependant que DR-MINER extrait des DRBS. L'une des qualités de l'utilisation des seuils de pertinence est de permettre d'extraire des DRBS dans des temps acceptables dans la plupart des cas. Si les seuils de pertinence ne sont pas satisfaits il y a élagage dans l'arbre d'énumération.

Notons cependant ce que MAD-G permet et que DR-MINER ne fait pas à notre connaissance. D'une part, MAD-G permet de calculer l'ordre de précédence entre les concepts denses (et formels). D'autre part, MAD-G permet de calculer les générateurs minimaux. Il n'y a pas de quoi s'étonner puisque MAD-G est un descendant direct de MagaliceA. Dans le premier cas, si une application à besoin de tracer le treillis de concepts denses, les informations nécessaires à cet effet sont incorporés à même les concepts. Dans le deuxième cas, si on veut générer des règles "presque exactes" d'association, les itemsets fermés "denses" dans les concepts denses (l'intension du concept) et les générateurs minimaux "denses" sont disponibles.

Conceptuellement, il y a peu de changement à effectuer dans l'algorithme MagaliceA pour devenir MAD-G. Ceci a une certaine importance. MagaliceA permet de calculer les treillis de concepts formels ainsi que les icebergs de concepts formels, sans oublier les générateurs minimaux. L'ajout de l'option d'extraction des concepts denses épousant la

philosophie de MagaliceA permet à même un seul algorithme la possibilité d'offrir une collection d'un nouveau motif[84]. Le choix pourrait se faire grâce à des options proposées à l'utilisateur.

La nature des concepts denses doit être également couverte. Que vaut un concept dense si l'information est dégénérée par la condensation par densité. Il faut se rappeler cependant que l'utilisateur a une part de responsabilité dans le choix des seuils de densité[85]. Supposons que les choix des seuils sont valables. On a remarqué, à la section 9.3, que MAD-G respecte les "frontières" des concepts disjoints dans les contextes formels. Même s'il ne parvient pas pour un certain pourcentage d'exception à recréer les concepts d'origine, les concepts denses extraits demeurent dans les frontières des concepts disjoints. Autrement dit si, comme dans les exemples de la section 9.3, il y a 3 concepts disjoints, on trouvera donc 3 sous-treillis dans le TCD. En admettant que le but premier était de reconstituer les concepts d'origine, rien n'empêche, une fois l'extraction par MAD-G terminée, de faire un traitement a posteriori pour créer un concept dense par sous-treillis.

En dernier lieu, MAD-G retient le principe d'insertion des transactions et de construction des concepts par incrément.

9.6.2 Limites de MAD-G

En regard à l'avant dernier point cité dans la sous-section précédente, s'il existe quelques objets ayant une relation d'incidence avec des attributs à l'extérieur de la frontière des concepts disjoints dans le contexte formel, l'avantage précédemment cité ne tient plus pour MAD-G.

---

[84] De même famille que les concepts formels, bien sur.

[85] Tout comme des seuils de fréquence minimale ou de confiance.

DR-MINER permet l'utilisation de contraintes sur la maximalité des DRBS. En un mot, la contrainte de maximalité signifie que seuls les DRBS ayant les dimensions maximales sur les deux dimensions des DRBS seront conservés. Dans le contexte de MAD-G, on peut également le faire en éliminant les concepts denses afin de satisfaire ce type de contraintes. Mais ce sera fait a posteriori, c'est à dire une fois que la collection sera complète. Il faut souligner cependant que MAD-G calcule les liens de précédence. Donc, si on élimine les concepts denses ne satisfaisant pas les contraintes de maximalité a posteriori, les liens de précédence ne tiennent plus. En d'autres termes, si on insiste pour garder les liens de précédence tout en ne gardant que les concepts denses répondant aux contraintes de maximalité, il faut recalculer les liens de précédence à nouveau pour pouvoir tracer le treillis ou l'iceberg.

Due à la nature de l'algorithme MAD-G, on est obligé de faire intervenir des contraintes qui ne sont pas nécessaires à DR-MINER, notamment la contrainte sur les singletons et la contrainte sur les concepts $\top$ et $\bot$. DR-MINER est un algorithme de type traitement en lot. Autrement dit, DR-MINER calcule les DRBS un à la fois. Une fois un DRBS extrait, DR-MINER calculera le prochain DRBS. Ce n'est pas le cas de MAD-G (et de MagaliceA), c'est un traitement incrémental. Il n'y a donc aucune possibilité lors d'une ou plusieurs itérations de savoir si un ou des concepts denses (ou formels) sont complets. Les contraintes sur les singletons et les concepts $\top$ et $\bot$ permettent de protéger l'intégrité du treillis. Toutefois, il y a là un effet adverse soit la limitation de la condensation par densité.

Il est également vrai que MAD-G calcule les générateurs minimaux. Mais ce calcul s'avère plus difficile que prévu pour certains seuils de densité. D'autre part, sur le même sujet, il faudrait caractériser ces générateurs minimaux. L'intuition fut qu'ils sont de même nature que les δ-libres mentionné dans (Boulicaut, Bykowski, & Rigotti, 2003) mais il faudrait en faire la preuve formelle.

En dernier lieu, la sous-section 9.4.5 parle de la consistance des concepts denses extraits par MAD-G. Il a été noté que certains de ces concepts denses ne respectent pas la contrainte de densité imposée. Ce dernier point est approfondi dans la sous-section suivante.

9.6.3 La collection de concepts denses produite par MAD-G, une représentation condensée ε-adéquate ?

Soit $\varepsilon = (\alpha,\alpha^*)$. Peut-on affirmer que la collection de concepts denses extraite par MAD-G est une représentation condensée ε-adéquate ? La réponse est non. Il fut démontré à la sous-section 9.4.5 que la consistance des concepts denses n'est pas respectée. En d'autres mots, certains concepts denses ne respectent pas la contrainte de densité. Avant de donner une explication sur ce qui arrive lors de l'extraction par MAD-G l'exemple suivant illustre le problème.

Exemple 9.1

Soient l'ensemble d'objets O = {0, 1, 2, 3, 4, 5, 6}, l'ensemble d'attributs A = {A, B, C, D, E, F, G} et le contexte formel K = {O, A, I} tel qu'illustré ci-dessous.

Contexte *K*							
Objets	**Attributs**						
	A	**B**	**C**	**D**	**E**	**F**	**G**
0	✖	✖	✖	✖	✖	✖	✖
1	✖	✖	✖	✖	✖	✖	✖
2	✖	✖		✖	✖		
3	✖		✖	✖	✖		
4	✖	✖	✖	✖			
5	✖	✖	✖	✖			
6	✖	✖	✖	✖			

Si $\alpha = \alpha^* = 0$, MAD-G calculera la collection de concepts formels suivante : {(0123456, AD), (012456, ABD), (013456, ACD), (0123, ADE), (01456, ABCD), (012, ABDE), (013, ACDE), (01, ABCDEFG)}.

Supposons maintenant que $\alpha = \alpha^* = 1$. MAD-G calculera la collection suivante : {(0123456, ACD), (012456, ABCD), (0123, ACDE), (012, ABCDEFG)}. La taille de la collection est passée de 8 concepts formels à 4 concepts denses soit 50% de réduction. Après examen, on se rend compte que les 3 premiers concepts denses respectent la contrainte de densité pour les seuils choisis. Cependant, pour le concept dense (012, ABCDEFG), la contrainte de densité n'est pas respectée. Pourquoi ? Avant de s'attaquer à la raison le tableau suivant présente l'ordre des transactions traitées par MAD-G.

#	Attribut	Objets
1	A	0, 1, 2, 3, 4, 5, 6
2	D	0, 1, 2, 3, 4, 5, 6
3	B	0, 1, 2, 4, 5, 6
4	C	0, 1, 3, 4, 5, 6
5	E	0, 1, 2, 3
6	F	0, 1
7	G	0, 1

Après la cinquième itération (attribut : E et objets : 0, 1, 2, 3), on a le treillis illustré à la figure 9.15 (a). Le concept (012, ABCDE) a déjà une exception.

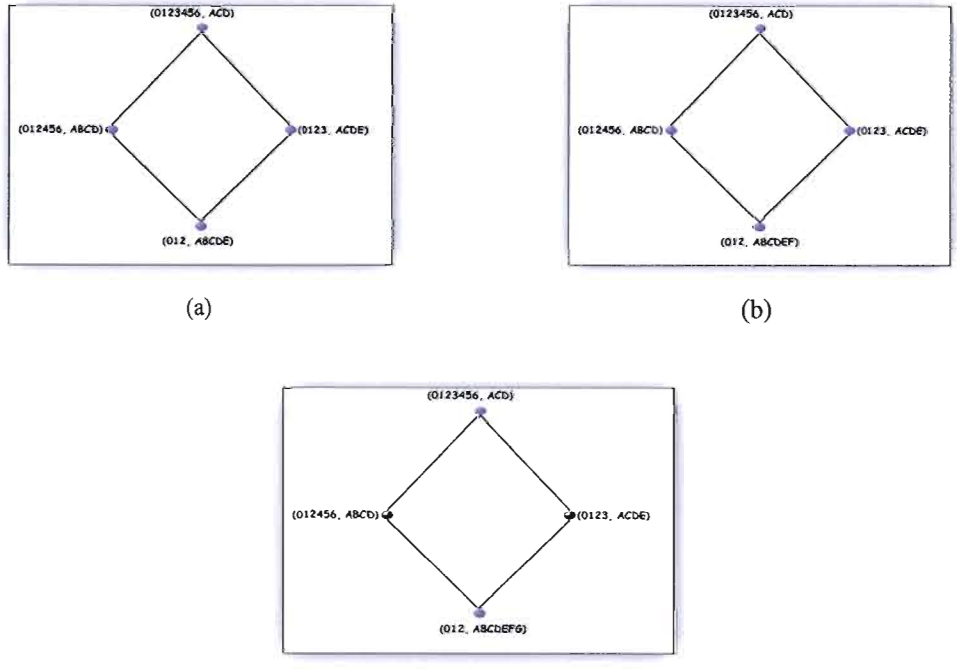


Figure 9.15 – trace du problème de condensation par densité composée

Dans la prochaine itération on traite la transaction 6 (attribut : F et objets 0, 1) et le treillis sera mise à jour tel que montré à la figure 9.15 (b). Le concept qui a été modifié est le $\perp$ du treillis. Il est passé de (012, ABCDE) à (012, ABCDEF). Il est à noter que (012, ABCDE) avait déjà une exception sur l'objet 2. Curieusement, bien qu'une exception soit présente dans ce concept, MAD-G ajoute l'attribut F au $\perp$ et permet une seconde exception sur l'objet 2 malgré le fait que les seuils de densité $\alpha = \alpha^* = 1$. À la dernière transaction, le $\perp$ est modifié pour devenir (012, ABCDEFG) (figure 9.15 (c)). Une troisième exception sur l'objet 2 est acceptée. Force nous est d'admettre que MAD-G ne produit pas une représentation condensée ε-adéquate.

❑

Que s'est-il passé pour que MAD-G ne puisse se rendre compte qu'il y a trois exceptions sur l'objet 2 et qu'elles sont acceptées malgré la valeur des seuils de densité ? Nous appellerons ce problème, la condensation par densité composée. L'expression est empruntée d'un terme dans le domaine bancaire, l'intérêt composé.

Le problème découle du fait qu'après chaque itération les concepts denses n'ont pas de fonctions mémoire (ou de structures) gardant la trace des transactions et du nombre de 0 sur les lignes et les colonnes d'un concept dense. Ainsi, lorsque MAD-G traite la transaction 6, l'algorithme trouve qu'il y a une exception sur les extensions du concept minimal et du concept candidat et qu'une exception sur les intensions du concept minimal et du concept candidat, même si dans les faits il y a une exception déjà présente mais non perçue dans le concept minimal. Par contre, il n'y a aucune indication que l'objet 2 du concept minimal (012, ABCDE) a déjà une exception. Le concept dense (012, ABCDE) est en fait traité comme un concept formel, i.e. sans exception. Lorsque la transaction 6 est traitée, MAD-G n'a aucune idée qu'un des objets (par voie de conséquence un attribut également) a une exception. Ne sachant pas cette information et traitant le concept (012, ABCDE) comme formel, il accepte donc la fusion avec le concept candidat puisqu'il n'y a "apparemment" qu'une seule exception. Il en va de même pour la transaction 7.

Alors comment DR-MINER fait pour reconnaître que des objets et/ou attributs ont déjà des exceptions ? Si on revient au tableau 9.1, DR-MINER exécute la fonction de propagation vérifiant de façon locale si les exceptions pour la densité sont acceptables ou non. Par contre, la vérification sur la pertinence est faite de manière globale[86]. Lors de la vérification de la consistance (*estConsistant* dans l'algorithme DR-MINER), on vérifie encore si les exceptions sont bien gérées. Cette dernière vérification est faite contre le contexte formel. Il est donc clair que DR-MINER lit plusieurs fois le contexte formel. MagaliceA et MAD-G ne lit qu'une seule fois le contexte formel. C'est l'un des avantages de ces deux algorithmes pour l'extraction des concepts formels. Malheureusement, pour ce qui est des concepts denses, il semble que ce soit une faiblesse. Tout au moins pour le cas qui nous occupe. Mais, il est clair que ce cas n'est pas une rareté, surtout dans des cas où les contextes formels sont denses. Ce chapitre se termine sur cette question, toutefois ce sujet sera visité à nouveau dans la section perspective du prochain chapitre qui conclura ce mémoire.

[86] Voir la définition 7.10 de la pertinence au chapitre 7. Ces vérifications sont faites en fonction de l'ensemble des objets et de l'ensemble des attributs. Même si les seuils minimaux de pertinence sont à 1 (produisant des concepts formels), DR-MINER vérifie toujours la contrainte de pertinence. Donc DR-MINER a une vue globale de la situation d'un DRBS en construction dans l'arbre d'énumération.

CHAPITRE X
CONCLUSION

10.1 Contributions principales

10.1.1 Généralités

Dans ce mémoire, il a été prouvé que l'algorithme MAD-G basé sur MagaliceA peut effectivement extraire des concepts denses et formels ainsi que les générateurs minimaux et les liens de précédence préservant l'intégrité du treillis selon les principes de l'AFC. La performance de MAD-G est sans conteste si on la compare à la performance de DR-MINER. Ce dernier n'extrait pas les générateurs minimaux, ni ne calcule les liens de précédence ; ce que MAD-G fait. De plus, MAD-G a extrait des collections de concepts denses là où DR-MINER ne pouvait pas les extraire ou les extrayait difficilement[87]. Il est à noter cependant que les principes de l'AFC sont présents autant pour MAD-G que pour DR-MINER.

Il a été observé néanmoins que DR-MINER produit une condensation par densité (et par pertinence[88]) plus concise que MAD-G. Cela tient du fait que MAD-G utilise deux contraintes, la contrainte sur les singletons et la contrainte sur les concepts $\top$ et $\bot$, afin de préserver l'intégrité du treillis des concepts en construction.

---

[87] Voir les chapitres 7 et 9 de ce mémoire qui donnent les raisons de cette limitation.

[88] Par défaut, les seuils de pertinence minimale dans DR-MINER sont fixés à 1. Dès lors, la contrainte de la pertinence est toujours effective.

10.1.2 De la qualité des concepts denses extraits par MAD-G

10.1.2.1 Cas des contextes formels avec partitionnement "naturel"

Dans le cas des contextes formels formés de concepts disjoints avec des exceptions à l'intérieur desdits concepts, on décèle que pour certaines valeurs de seuils de densité, la collection de concepts denses extraite par DR-MINER peut produire des concepts dégénérés, c'est à dire que des relations entre objets et attributs sont créés à l'extérieur des partitions "naturelles" des disjonctions. La contrainte sur les singletons dans MAD-G semble protéger des chevauchements indus entre les partitions des concepts disjoints avec ou sans erreur à l'intérieur de leurs frontières respectives. Conséquemment, même si MAD-G extrait plus de concepts denses que DR-MINER, les concepts denses de MAD-G reflètent plus la réalité du contexte formel avec partitions.

10.1.2.2 Cas normal des contextes formels

Dans le cas où le contexte formel n'a pas ou peu de concepts disjoints, i.e. ne possédant peu ou pas des partitions "naturelles", DR-MINER a plus de chances d'extraire des concepts denses de qualité que MAD-G. Quelle en est donc la raison ?

La faiblesse principale de MAD-G est qu'il traite la transaction et le concept à modifier ou à créer sans égard à la situation globale des attributs et des objets dans le contexte formel. MAD-G calcule les concepts denses (ou formels) en ne lisant le contexte formel qu'une seule fois. Ainsi à chaque exécution de l'algorithme pour une transaction quelconque, les concepts à modifier et les concepts géniteurs sont traités comme des concepts formels pour l'itération en cours. Donc, les concepts en construction "ne se souviennent plus" s'ils sont en réalité des concepts formels en construction ou des concepts denses en construction. Puisque qu'on ne s'assure pas de la consistance du concept dense en construction par une relecture des objets et des attributs dans le contexte formel, on n'a que le concept en état de construction comme unique point de référence pour la consistance et ce

concept est considéré de facto comme formel pour l'itération en cours. Ceci est insuffisant et a pour effet adverse de produire ce que nous avons appelé la condensation par densité composée. Étant donné ce problème, il a été noté au chapitre 9 de ce mémoire qu'on ne peut affirmer que la collection de concepts denses calculée par MAD-G peut être appelée une représentation condensée ε-adéquate de la collection de concepts formels car, après un certain temps ou un certain nombre d'itérations, les concepts denses ne respectent plus les seuils de densité déterminés par l'utilisateur.

En comparaison, DR-MINER vérifie toujours que le concept dense en construction est consistant par rapport au contexte formel avant l'appel récursif. Naturellement cela signifie que DR-MINER pourrait lire et relire les mêmes rangées (objets) et/ou les mêmes colonnes (attributs) des concepts denses dans le contexte formel pour la validation de la consistance. Donc la lecture des transactions du contexte formel ne se fait pas en une seule passe mais, dans la plupart des cas, c'est au prix d'une meilleure qualité pour les concepts denses et du respect des seuils de densité choisis par l'usager.

10.1.3 Des générateurs minimaux

On n'a pu déterminer ou caractériser les générateurs minimaux extraits des concepts denses par MAD-G. L'intuition était qu'ils seraient semblables aux motifs δ-libres. Étant donné, que des concepts denses extraits par MAD-G ne respectent pas la contrainte de densité, il est inutile à ce stade de tenter de les caractériser.

10.1.4 De la qualité des concepts denses, question ouverte

Il est de notre avis qu'il sera difficile d'extraire une collection parfaite de concepts denses. L'exemple suivant illustre cet état de chose.

Exemple 10.1

Soient l'ensemble d'objets $O = \{1, 2, 3, 4, 5, 6\}$, l'ensemble d'attributs $A = \{A, B, C, D, E\}$ et le contexte formel $K = (O, A, I)$ tel que présenté au tableau ci-dessous. Supposons $\alpha = \alpha^* = 1$. Supposons également qu'on ne tient pas compte des contraintes sur la pertinence, les singletons et sur les concepts $\perp$ et $\top$. Notons que le contexte est ordonné de manière à être traité par un algorithme quelconque (disons Fusion-Dense) en utilisant la représentation verticale et les objets associés aux attributs sont en ordre de cardinalité des objets.

Objets	Attributs				
	A	B	C	D	E
1	✖	✖	✖	✖	✖
2	✖	✖	✖	✖	✖
3	✖	✖	✖	✖	✖
4	✖	✖	✖	✖	✖
5	✖	✖	✖	✖	
6	✖	✖			

Fusion-Dense calculera le concept dense en construction (123456, AB) une fois la transaction B terminée. Le concept (123456, AB) est techniquement un concept formel en construction pour le moment. Lorsque la transaction (C, 12345) est traitée, un concept candidat (12345, ABC) est créé et vérifié avec le concept minimal si la contrainte de densité est respectée pour $\alpha = \alpha^* = 1$. Elle l'est, alors l'algorithme Fusion-Dense calculera le concept dense en construction suivant : (123456, ABC). Passons maintenant à la transaction (D, 12345). En principe Fusion-Dense refusera de fusionner puisqu'elle violera la contrainte de densité sur le nombre de '0' accepté sur la ligne de l'objet 6 du contexte formel.

❑

Qu'est-ce qui dicte qu'une transaction peut être condensée alors qu'une autre ne le peut pas dans des circonstances égales? Dans l'exemple précédent, les transactions C et D avaient des chances égales d'être fusionnées au concept en construction (123456, AB). Dans ce cas-ci, par rapport aux seuils de densité choisis, une transaction sera fusionnée au concept et l'autre sera rejeté de cette fusion. Notons que MAD-G n'a pas ce problème puisqu'il est

frappé d'un autre problème celui de la condensation par densité composée. Supposons que ce problème de MAD-G est solutionné, on se retrouvera pris avec le problème exemplifié dans exemple 10.1. Il est à noter que DR-MINER semble faire exactement ce que l'exemple précédent démontre, soit accepter une transaction et rejeter l'autre[89]. On peut naturellement répondre à ce problème en modifiant les seuils de densité afin d'accepter les deux transactions.

10.1.5 Des contraintes, question ouverte

Jusqu'à présent MagaliceA n'utilisait qu'une seule contrainte (anti-monotone) nommément la contrainte de la fréquence minimale. Avec MAD-G, descendant direct de MagaliceA, de nouvelles contraintes se sont ajoutées, soient :

- La contrainte sur la densité des concepts denses,
- La contrainte sur les singletons et,
- La contrainte sur les concepts $\top$ et $\bot$.

Les deux dernières contraintes sont utilisées afin de préserver l'intégrité du treillis de concepts. On pourrait dire de ces deux contraintes, en langage technique de tous les jours, qu'elles sont des correctifs[90] temporaires.

Dans le cas de la contrainte sur la densité, quelle est sa relation par rapport à la contrainte de la fréquence minimale ? Dans (Jeudy, 2002), l'auteur parle des effets adverses que certaines contraintes peuvent avoir sur d'autres contraintes. L'ordre d'exécution de ces contraintes peut affecter les résultats. Par rapport à MAD-G (et DR-MINER), la question est la suivante. Quand doit-on appliquer la contrainte de fréquence minimale, avant la fusion par densité ou une fois que le concept est fusionné ?

---

[89] Elle sera rejetée pour le concept dense (ABC, 123456), mais le concept (ABCD, 12345) sera créé en vertu que ce dernier est un concept formel. La collection des concepts formels est après tout un sous-ensemble de la collection des concepts denses.

[90] Connu sous le vocable habituel de patch.

Se référant à l'exemple ci-dessus, supposons que le support minimal σ est 5 et que $\alpha = \alpha^* = 3$. Si on applique la contrainte de fréquence minimale avant la contrainte de densité, la transaction (E, 1234) sera rejetée. Par contre, si on applique la contrainte de densité avant celle de la fréquence minimale, la transaction (E, 1234) sera fusionnée et sera acceptée. La réponse facile serait de dire qu'on peut toujours donner le choix de l'ordonnancement de certaines contraintes à l'utilisateur. Cela impliquera une programmation et une gestion des contraintes plus complexes.

10.1.6 De l'incrémentation, question ouverte

Dans ce mémoire, les exemples et expérimentations se sont limités au traitement d'un contexte formel pour un certain état. Dans la "vraie vie", les bases de données sont continuellement augmentées de nouvelles transactions. Dans les systèmes commerciaux OLAP, cet état de chose est pris en compte. Les nouvelles transactions sont prises en compte par le processus ETC (extraction, transformation, chargement)[91] afin d'alimenter l'entrepôt de données.

L'algorithme incrémental par cardinalité calculant les concepts formels en utilisant la représentation verticale ou horizontale du contexte formel peut théoriquement alimenter la collection de concepts formels et la modification des liens de précédence dans son treillis. C'est en fait la force de l'algorithme incrémental. Les algorithmes de type traitement en lot, doivent reprendre le calcul sur le contexte formel augmenté de ses nouvelles transactions.

Il est clair que l'algorithme actuel MAD-G ne peut prendre en compte les nouvelles transactions ajoutées dans le contexte formel et ce, après l'extraction des concepts denses. La raison principale est due à la condensation par densité composée. Il est alors évident que si pour un contexte formel donné, les concepts denses calculés ne respectent pas la contrainte de densité, l'ajout de nouvelles transactions dans le contexte formel et le traitement de ces

[91] De l'anglais ETL, extraction, transformation and loading.

transactions par MAD-G pour modifier la collection de concepts denses ne fera qu'amplifier le problème de condensation par densité composée.

10.2 Perspectives

Les quelques propositions de cette section ne seront pas sans rappeler certains principes des BDI mentionnées dans l'introduction de ce mémoire.

10.2.1 MAD-G et la consistance, une solution possible

Il serait intéressant d'implémenter le principe de consistance dans l'algorithme MAD-G. Il y aura certainement un impact sur la performance de l'algorithme mais au prix d'une meilleure condensation par densité et de résultat plus "exacts[92]". Cette dernière sera meilleure du fait de l'élimination (ou presque) de la condensation par densité composée. D'autre part, l'implémentation de la consistance dans MAD-G pourrait vouloir dire l'élimination des contraintes sur les singletons et sur les concepts $\perp$ et $\top$. La collection de concepts denses extraite par MAD-G serait plus semblable à la collection extraite par DR-MINER, donc de plus petite taille. Ce qui est souhaitable. Elle pourrait répondre non seulement à la production de concepts denses de qualité mais également au traitement de nouvelles transactions dans un contexte formel augmenté de celles-ci.

Nous croyons également que l'utilisation du principe de la consistance par MAD-G lors de l'extraction des concepts denses permettra d'extraire des générateurs minimaux de type δ-libre de qualité. À partir de ce moment, il sera possible de vérifier leurs qualités intrinsèques et de les comparer à l'extraction des δ-libres produite par MIN-EX.

[92] C'est à dire des concepts denses plus exacts ou reflétant plus la réalité.

Une possibilité (ou intuition) d'implémentation de la consistance serait la suivante. Il faudrait charger en mémoire les représentations verticale et horizontale du contexte formel en utilisant des tables de hachage[93] afin de promouvoir la consistance des concepts denses. Ainsi, la recherche pour la consistance sera un peu plus rapide. Si le nombre de transactions est trop élevé pour un chargement en mémoire centrale, on peut également utiliser un SGBDR/O pour le contexte formel ainsi que des vues logiques et des index créés spécifiquement pour le contexte formel[94].

En dépit de ces changements qui affecteront la performance de MAD-G, nous tablons sur la performance actuelle de MagaliceA et de MAD-G, qui est dans la plupart des cas linéaire, pour supporter le changement.

10.2.2 Un rôle possible des concepts denses dans les BDI/SGBDI

Dans (Mielikäinen, 2004), l'auteur écrit que la question fondamentale en ce qui regarde le forage de données et la BDI est la suivante. Nous avons des données, que devons-nous rechercher dans celles-ci ? Une question plus concrète, selon le même auteur, serait celle-ci. Nous avons cette base de données[95], quels sont les (type de) requêtes pertinentes en relation avec cette base de données ? Ajoutons à cela notre propre question. Comment déterminons-nous des requêtes pertinentes et des paramètres (ou seuils) associés à celles-ci sur une base de données quelconque ? Cette dernière question est pertinente surtout si on a

---

[93] MagaliceA utilise déjà cette approche pour la représentation verticale du contexte formel dont les attributs sont triés dans l'ordre lexicographique.

[94] Ici on entrevoit le couplage base de données et collection de motifs tel que proposé dans le modèle DBI/SGBDI.

[95] On présume une base de données inductive ou relationnelle dans un SGBDI.

peu ou pas de connaissances sur les données à traiter[96]. T. Mielikäinen affirme que ces deux questions (et la nôtre) met en évidence la notion suivante sur les BDI. Selon lui, une BDI est constituée des composantes et fonctionnalités des bases de données usuelles et d'une collection de classements R d'outils de manipulation de données. Un classement est une liste d'opérations de manipulation des données dans Q[97] et chaque opération de manipulation de données n'apparait qu'une seule fois. L'opération la plus importante des classements est l'extraction de la tête de la liste. Cette simple opération serait suffisante pour plusieurs tâches de forage de données.

Il y a là, à notre avis, un élément intéressant applicable à la collection des concepts denses. Il a été noté que plus les seuils de densité sont élevés moins de concepts denses MAD-G extrait et plus l'algorithme est performant. Ainsi, pour des seuils de densité maximaux, le treillis de concepts denses serait un classement dans le cadre de la BDI selon (Mielikäinen, 2004). Considérant que le SGBDI se veut un outil qui promeut l'exploration et l'interactivité, nous faisons la proposition suivante. L'extraction de la collection des concepts denses, avec des seuils maximaux de densité, et de son treillis devrait être la première opération de forage avant tout autre opération de forage cherchant des particularités aux données. L'exemple suivant illustre le principe en vue.

---

[96] Par peu ou pas de connaissances, nous voulons dire que l'utilisateur ne connaît pas la configuration des données. Par exemple, plusieurs expériences sont faites par des puces à ADN et produisant une grande quantité d'information. Le biologiste peut avoir une idée de ce que les expériences devraient démontrer, mais il n'a aucune idée si les expériences valideront ses hypothèses ou non ou si les expériences produiront des erreurs dans les résultats. Si c'est le cas, les suppositions du biologiste ne seront pas validées… sauf si l'outil de prospection de données permet de tenir compte des exceptions.

[97] Nous rappelons que Q est une classe de requêtes sur une classe de structures (Mannila & Toivonen, 1996). Le chapitre 3 de ce mémoire examine les représentations condensées, les classes de structures et de requêtes.

Exemple 10.2

La figure suivante illustre la première tâche de l'analyste de données qui est d'extraire (à l'aide de MAD-G par exemple) le treillis de concepts denses avec des seuils maximaux de densité. On peut penser que l'extraction pourrait se faire automatiquement durant l'exécution des tâches de nuit. Le résultat serait matérialisé. L'analyste, le matin venu, pourra vérifier la collection des motifs extraits et/ou le treillis produit.

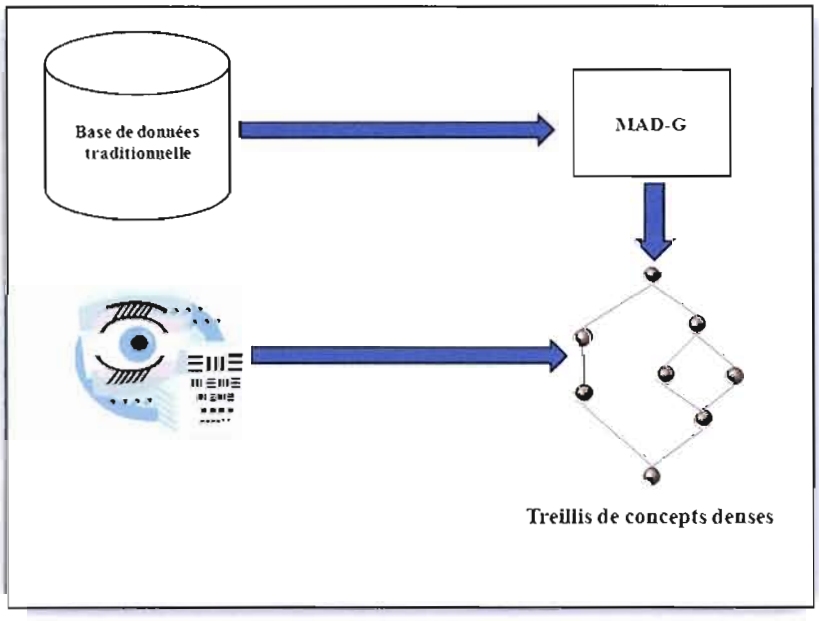


Figure 10.1 – Extraction du treillis de concepts denses avec seuils maximaux

La figure suivante illustre l'utilisateur choisissant un concept dense d'intérêt parmi la collection de concepts denses extraite durant les tâches de nuit. Il demande l'extraction de la collection des concepts et de son treillis à partir du concept dense d'intérêt. L'utilisateur peut choisir d'extraire les concepts formels ou bien les concepts denses selon les seuils de densité désirés. Le concept dense précédemment choisi contenant les objets et les attributs guide l'extraction d'un sous-contexte formel à partir du contexte formel d'origine et MAD-G par exemple, extraira le treillis de concepts (formels ou denses) avec les seuils choisis. L'utilisateur peut alors examiner cette nouvelle collection issue du

concept d'intérêt. Il peut choisir de matérialiser cette collection et/ou il peut choisir d'autres concepts d'intérêt et reprendre le processus itératif et exploratoire.

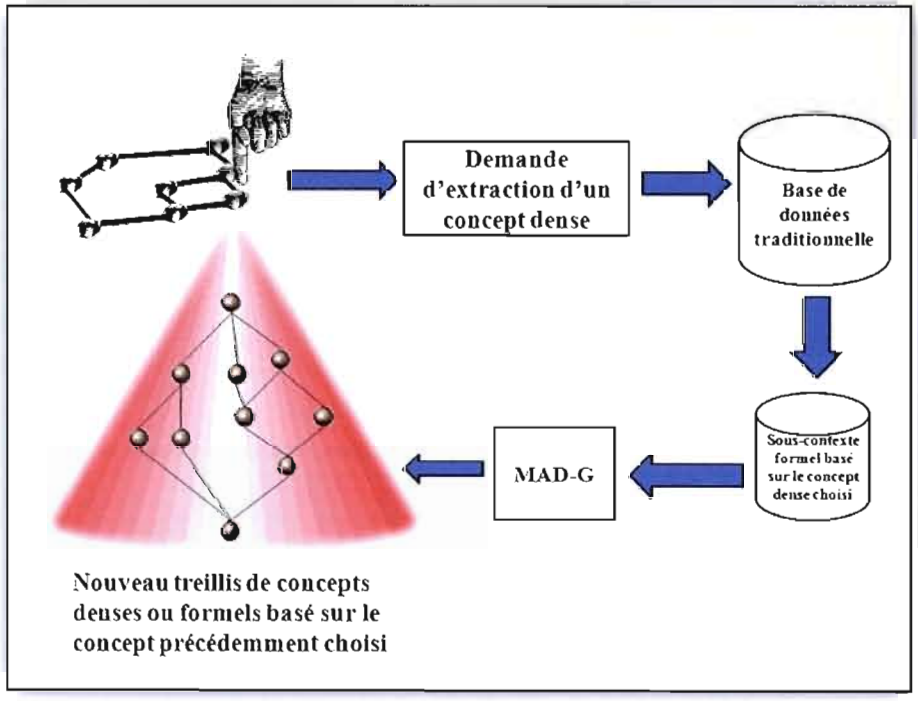


Figure 10.2 – Interactivité et exploration

❑

Bien entendu, il faut, pour que l'exemple ci-dessus soit réalisable, que MAD-G calcule correctement les concepts denses, tout au moins selon les idées émises à la sous-section 10.2.1.

10.2.3 Divers

10.2.3.1 De la qualité des concepts denses, une solution possible

Il a été noté dans (Besson, 2005) que l'utilisation de la densité relative (un pourcentage admis de '0' dans un concept) pourrait produire des concepts denses dégénérés. L'utilisation des valeurs absolues de seuils de densité permet de circonscrire le problème mais dans certains cas, certaines transactions pourtant également valides seront rejetées par la contrainte de densité à valeurs absolues (voir sous-section 10.1.4). Une solution possible pourrait bien être l'utilisation de valeurs absolues et relatives de seuils de densité. Nous croyons que cette utilisation augmentera la qualité des concepts dense. Ce sera, probablement, au prix d'une augmentation considérable de la complexité de l'algorithme, de sa conception et de sa réalisation.

10.2.3.2 De l'utilisation des contraintes, une solution possible

L'extraction des concepts denses par DR-MINER est renommée difficile. Cependant, dans (Boulicault & Besson, 2008), les auteurs croient que l'ajout de nouvelles contraintes pourrait augmenter la performance de DR-MINER. Parmi les contraintes déjà explorées, on trouve la contrainte sur la pertinence, la contrainte de la maximalité des dimensions des concepts, les contraintes syntaxiques telles que les choix d'attributs et d'objets particuliers à extraire.

C'est également notre avis en ce qui a trait à MAD-G. On sait que l'implémentation de la gestion de la consistance réduira la performance de MAD-G. D'un autre coté, l'utilisation de certaines contraintes pourrait palier l'effet négatif de la gestion de la consistance des concepts en construction. Il n'en demeure pas moins que le problème émis dans la sous-section 10.1.5 de l'ordonnancement des contraintes demeure entier.

10.2.3.3 Les percées technologiques du matériel et du logiciel à notre rescousse

Dans le domaine du forage et de l'analyse des données, les solutions actuelles tant pour l'architecture matérielle que pour les logiciels sont onéreuses[98]. Il est assuré que cet état de chose sera la même pour le développement des SGBDI.

Il n'en demeure pas moins que des solutions technologiques actuelles sont très intéressantes. Par exemple, les réseaux de stockage SAN[99], la matérialisation des vues et le partitionnement des tables dans les SGBDR/O modernes, le traitement en parallèle et le GRID sont des exemples flagrants d'outils qui augmentent de façon significative la performance de certains logiciels et des grands projets informatiques. Là où il n'y a pas si longtemps, certains projets informatiques étaient relégués au domaine du rêve ou de la fantaisie, on entrevoit maintenant la possibilité de les concevoir !

Il est clair pour nous que l'utilisation de ces technologies est plus que désirable voire obligatoire, si on veut voir un jour des SGBDI commerciaux. Dans une moindre mesure et dans le cadre de l'extraction des concepts formels et denses, il est plus que probable que la solution à ce genre d'extraction passe par l'utilisation de SGBDR/O performants offrant la matérialisation des vues et le partitionnement des tables. Elle passe probablement par l'utilisation du parallélisme des processus et même du GRID.

---

[98] Même si les composantes comme la mémoire principale et secondaire et les micro-processeurs sont relativement peu dispendieux, les coûts de l'ensemble de ces composantes, des logiciels de développement, d'intégration, de gestion, de maintenance des produits OLAP et de la main-d'œuvre sont astronomiques. Selon notre expérience, le budget de certains de ces projets avoisinait dans les 5 à 15 millions de dollars et les projets s'échelonnaient sur des mois voire des années.

[99] SAN : Storage Area Network

BIBLIOGRAPHIE

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. *Proc. 1994 Int. Conf. Very Large Databases (VVLDB' 94)*, (pp. 487-499). Santiago, Chile.

Ben Yahia, S., & Nguifo, E. M. (2004). Approches d'extraction de règles d'association basées sur la correspondance de Galois. Dans *Motifs dans les bases de données* (pp. 23 - 55). RSTI - ISI.

Besson, J. (2005). *Découverte des motifs pertinents pour l'analyse du transcriptome: Application à l'insulino-résistance.* Thèse de Doctorat.

Besson, J., Robardet, C., & Boulicaut, J. (2006). Mining a new fault-tolerant pattern type as an alternative to formal concept discovery. *LNCS , 3933*, 144-157.

Boulicault, J., & Besson, J. (2008). Actionability and Formal Concepts: A Data Mining Perspective. *ICFCA 2008. LNAI 4933*, pp. 14-31. Montréal: Springer.

Boulicaut, J., Bykowski, A., & Rigotti, C. (2003). Free-Sets: A Condensed Representation of Boolean Data for the Approximation of Frequency Queries. *DMKD 7 , (1)*, 5-22.

Bucila, C., Gehre, J., Kifer, D., & White, W. (2003). Dualminer: A dual-pruning algorithm for itemsets with constraints. *DMKD 7 (4)* , 241-272.

Carpineto, C., & Romano, R. (2004). *Concept Data Analysis - Theory and Applications.* Wiley.

Dchaspe, L., & Toivonen, H. (1999). Discovery of frequent Datalog patterns. *DMKD , 3*, 7-36(30).

De Raedt, L. (2003). A perspective on inductive databases. *SIGKDD Explorations* , 4(2) 69-77.

Gionis, A., Mannila, H., & Seppänen, J. (2004). Geometric and combinatorial Tiles in 0-1 Data. *PKDD 2004. 3202/2004*, pp. 173-184. Berlin/Heidelberg: Springer.

Godin, R. (2006). *Systèmes de gestion de bases de données par l'exemple* (éd. 2). Longueuil, Québec, Canda: Loze-Dion.

Godin, R., Missaoui, R., & Alaoui, H. (1995). Incremental Concept Formation Algorithms Based on Galois (Concept) Lattices. *Computational Intelligence , 11 (2)*, 246-267.

Imielinski, T., & Mannila, H. (1996). A database perspective on knowledge discovery. *Communications of ACM, 39 (11)* , 58-64.

Jay, N., Kohler, F., & Napoli, A. (2008). Analysis of Social Communities with Iceberg and Stability-Based Concept Lattices. *ICFCA 2008. LNAI 4933*, pp. 258-272. Berlin/Heidelberg: Springer.

Jeudy, B. (2002). *Optimisation de requêtes inductives: Application à l'extraction sous contrainte de règles d'association.* Thèse de doctorat.

Kaufman, K. A., & Michalski, R. S. (2003). The development of the inductive database system VINLEN: A review of current research. *International Intelligent Information Processing and Web Mining Conference* (pp. 393-398). Springer.

Kuznetsov, S., Obiedkov, S., & Roth, C. (2007). Reducing the Representation Complexity of Lattice-based Taxonomies. *LNCS* (4604/2007), 241-254.

Lipschutz, S. (1976). *Theory and Problems of Discrete Mathematics.* (McGraw-Hill, Éd.)

Lipschutz, S., & Lipson, M. (1992). *2000 Solved Problems in Discrete Mathematics.* (McGraw-Hill, Éd.)

Mannila, H., & Toivonen, H. (1997). Levelwise search and borders of theories in knowledge discovery. *DMKD, 1(3)* , 241-258.

Mannila, H., & Toivonen, H. (1996). Multiple uses of frequent sets and condensed representations. *Proc. of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*, (pp. 189-194). Portland, Oregon.

Mannila, H., Toivonen, H., & Verkamo, A. I. (1995). Discovering Frequent Episodes in Sequences. *Proc. of the First International Conference on Knowledge discovery and Data Mining (KDD'95)*, (pp. 210-215). Montreal, Canada.

Mielikäinen, T. (2004). Inductive Databases as Ranking. *LNCS, Data Warehousing and Knowledge Discovery* , 149-158.

Nehmé, K., Valtchev, P., Rouane, M., & Godin, R. (2005). On computing the minimal generator family for concept lattices and iceberg. *Proceedings of the International Conference on Formal Concept Analysis, 3403*, pp. 192-207.

Pagé, C. (2008). *Bases de règles multi-niveaux - Mémoire de maitrise.* UQAM.

Pei, J., Tung, A., & Han, J. (2001). Fault-tolerant frequent pattern mining: Problems and challenges. *DMKD'01* (p. n.a.). ACM.

Pensa, R., & Boulicaut, J. (2005). Towards Fault-Tolerant Formal Concept Analysis. *LNAI. 3673*, pp. 212-223. Springer-Verlag.

Pensa, R., Robardet, C., & Boulicaut, J. (2005). A Bi-clustering Framework for Categorical Data. *PKDD 2005* (pp. 643-650). Berlin/Heidelberg: Springer.

Piatetsky-Shapiro, G., & Frawley, W. J. (1991). *Knowledge Discovery in Databases.* Menlo Park, CA: AAAI Press/MIT Prees.

Roth, C., Obiedkov, S., & Kourie, D. (2008). Toward Concise Representation for Taxonomies of Epistemic Communities. *LNCS , 4923/2008*, 240-255.

Rouane, M. H. (2006). *Étude de l'analyse formelle dans les données relationnelles Application à la restructuration des modèles structuraux UML.* Thèse de Doctorat, Université de Montréal.

Rouane, M. H., Nehme, K., Valtchev, P., & Godin, R. (2004). On-line maintenance of iceberg concept lattices. *ICCS 2004*, (p. 14 p.).

Seppänen, J., & Mannila, H. (2004). Dense itemsets. *ACM SIGKDD'04* (pp. 683-688). ACM.

Stumme, G., Taouil, R., Bastide, Y., Pasquier, L., & Lakhal, L. (2002). Computing Iceberg Concept Lattice with Titanic. *Journal on Knowledge and Data Engineering , 42 (2)*, 189-222.

UC Irvine Machine Learning Repository. (2007). *UCI Machine Learning repository.* Récupéré sur http://archive.ics.uci.edu/ml/

Valtchev, P., Grosser, D., Roume, C., & Hacenum, R. (2003). Galicia: an open platform for lattices. *Proceedings of ICCS'03, Suppl. volume*, (pp. 241-254). Dresden.

Valtchev, P., Missaoui, R., Godin, R., & Meridji, M. (2002). Generating Frequent Itemsets Incrementally: Two novel approaches Based on Concept Analysis. *Journal of Experimental & Theoritical Artificial Intelligence* , 14(2-3):115-142.

Ventos, V., & Soldano, H. (2005). Alpha Galois Lattices: An Overview. *LNCS- Formal Concept Analysis , 3403/2005*, pp. 299-314.

Yang, C., Fayyad, U., & Bradley, P. (2001, August). Efficient discovery of error-tolerant frequent itemsets in high dimensions. *SIGKDD* , 194-203.