UNIVERSITE FRANÇOIS RABELAIS
Blois - Tours - Chinon
LABORATOIRE D'INFORMATIQUE

Ecole doctorale : Santé, Sciences et Technologies

THESE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITE DE TOURS

Discipline : Informatique

présentée et soutenue publiquement par :

CHEIKH TALIBOUYA DIOP

le 08 décembre 2003

TITRE

ETUDE ET MISE EN ŒUVRE DES ASPECTS ITERATIFS DE L'EXTRACTION DE REGLES D'ASSOCIATION DANS UNE BASE DE DONNEES

Directeur de thèse :

DOMINIQUE LAURENT

JURY :

Boulicaut Jean-François	Maître de conférences, HDR	Rapporteur	LIRIS CNRS, INSA, Lyon
Giacometti Arnaud	Maître de conférences	Co-directeur	Université de Tours
Laurent Dominique	Professeur	Directeur	Université Cergy-Pontoise
Niane Mary Teuw	Professeur	Co-directeur	U.G.B, Saint-Louis (Sénégal)
Spyratos Nicolas	Professeur	Examinateur	Université Paris-11
Vrain Christel	Professeur	Rapporteur	Université d'Orléans

Cette thèse en co-tutelle a été financée par la coopération française

A ma famille

A mon épouse

Remerciements

Je ne pense pas que quelques mots de remerciements puissent exprimer le sentiment de profonde gratitude et de reconnaissance que j'éprouve à l'endroit de mes encadreurs, Dominique Laurent et Arnaud Giacometti, pour m'avoir accueilli à l'Antenne chaleureuse de Blois, et encadré avec diligence, disponibilité totale et une clairvoyance remarquable pour ces travaux. Qu'ils trouvent ici l'expression de mes remerciements les plus sincères.

Je voudrais exprimer également mes sincères remerciements à Monsieur Mary Teuw Niane, Professeur à l'université Gaston Berger de Saint-Louis (Sénégal), qui n'a pas ménagé d'efforts pour le financement de cette thèse, et grâce à qui la convention de co-tutelle entre l'université de Tours et l'université Gaston Gerger a été établie.

Je remercie Madame Christel Vrain, Professeur à l'université d'Orléans, pour l'honneur qu'elle me fait d'avoir accepté d'être rapporteur de ma thèse et membre du jury.

Je remercie Monsieur Jean François Boulicaut, Maître de Conférences, HDR à l'INSA de Lyon, pour l'honneur qu'il me fait d'avoir accepté d'être rapporteur de ma thèse et membre du jury.

Je remercie Monsieur Nicolas Spyratos, Professeur à l'université Paris-Sud, Orsay, pour l'honneur qu'il me fait d'avoir accepté d'être membre du jury.

Je remercie toute l'équipe de l'Antenne Universitaire de Blois pour l'ambiance véritablement chaleureuse dans laquelle j'ai vécue durant tous mes séjours à Blois, ce qui a représenté pour moi un soutien moral important.

Je remercie également tous les membres de l'UFR SAT de l'université Gaston Berger de Saint-Louis, et en particulier ceux de la section informatique, pour leur soutien constant et leur compréhension.

Je remercie ma famille, mon épouse et tous mes amis pour leur amour et leur soutien permanent.

Je remercie Arnaud et Cécile qui m'ont toujours soutenu et aidé durant tous mes séjours à Blois, et qui représentent ma petite famille à Blois.

Cette thèse a été financée par une bourse en alternance du Ministère Français de la Coopération. Je tiens à remercier le service de Coopération et d'Action Culturelle de l'Ambassade de France à Dakar pour m'avoir offert cette opportunité.

Table des matières

Chapitre I

Introduction

Avec l'augmentation de la capacité de stockage, nous avons assisté durant ces dernières années à une croissance importante des moyens de génération et de collection des données. Il s'est ainsi créé un besoin d'acquisition de nouvelles techniques et méthodes intelligentes de gestion qui permettent d'extraire des données des informations utiles (également appelées connaissances). C'est ainsi que l'on a commencé à parler de découverte de connaissances à partir de données (KDD) ou encore de Data Mining ou de Fouille de données. Les techniques de Data Mining permettent de découvrir des informations importantes *cacheés* dans les données. Elles permettent par exemple de trouver les caractéristiques des clients les plus fidèles d'une banque, ou encore les tendances qui se dégagent dans les ventes de supermarché. La connaissance de telles informations peut permettre à la banque ou au supermarché d'élaborer des stratégies commerciales ou de marketing en directions de ces clients.
Cependant la découverte de telles informations, que l'on appelle aussi des motifs, pose un certain nombre de problèmes.
Le premier est le temps de calcul de ces motifs. En effet, les requêtes d'extraction de ces motifs sont des requêtes complexes et prennent ainsi du temps à s'exécuter. Ce problème est d'autant plus crucial que l'utilisateur ne sachant pas a priori ce qu'il cherche, doit poser un certain nombre de requêtes pour trouver ce qui l'intéresse, d'où une itération du processus. C'est ainsi que la découverte de connaissances dans les bases de données, appelée aussi Data Mining est définie comme l'extraction d'informations intéressantes, non triviales, implicites, préalablement inconnues et potentiellement utiles à partir de grandes bases de données [36]. Lorsqu'on se trouve dans un environnement multi-utilisateurs, où chaque utilisateur pose indépendamment ses requêtes, le problème se pose de manière encore plus cruciale. Le deuxième problème de la découverte d'informations intéressantes est la taille importante des réponses. Le nombre de motifs trouvés est généralement très important. L'expérience a montré que l'extraction n'est d'ailleurs pas toujours possible, du fait de ce nombre important de motifs et de leur taille. On peut se retrouver dans des situations où il y a un très grand nombre de motifs de très grande taille, ce qui fait qu'ils ne peuvent tous être chargés en mémoire pour être traités. C'est ainsi qu'est apparue l'idée de représentations condensées qui consiste à trouver un sous ensemble des motifs à partir duquel on pourra retrouver tous les autres sans accéder aux données. C'est ce sous-ensemble qui est appelé représentation condensée.

La principale contribution de ce mémoire est de proposer une approche itérative pour l'extraction des motifs intéressants. Celle-ci consiste à utiliser les résultats des extractions antérieures, plus précisément la représentation condensée de ces résultats, pour optimiser le calcul des extractions futures. Il s'agit après la découverte des motifs, de trouver leur représentation condensée, de la stocker et ensuite de réutiliser cette représentation condensée pour l'optimisation de l'extraction. Ainsi, deux problèmes se posent : celui d'un stockage efficace d'un ensemble de requêtes, et celui d'une réutilisation intelligente de cet ensemble. Nous proposons dans ce sens des méthodes efficaces de stockage et de réutilisation d'un ensemble de motifs dans une base de données.
D'autre part, étant donné que nous disposons de plusieurs ensembles de motifs, nous introduisons, dans l'optique d'une meilleure gestion de ces ensembles, un langage de manipulation d'ensembles de motifs, basé sur les opérations de l'algèbre relationnelle.
Ainsi, l'utilisateur peut exprimer une extraction à partir d'opérations (sélection, jointure,...) sur des requêtes d'extraction dont les résultats sont déjà stockés, et nous utilisons ces résultats pour optimiser la réponse à la nouvelle extraction.

L'autre contribution est de proposer une représentation condensée non pas d'une réponse à une requête mais plutôt d'un ensemble de réponses à des requêtes.
En effet, étant donné que les ensembles de motifs trouvés ne sont pas indépendants, un problème crucial qui se pose est la redondance dans le stockage. Il apparaît alors nécessaire de ne pas

stocker les représentations condensées indépendamment les unes des autres, mais de trouver une représentation condensée d'un ensemble de réponses à des requêtes.

Le mémoire est divisé en trois chapitres (le premier étant l'introduction). Le deuxième chapitre composé de trois parties, fait un état de l'art sur la découverte de motifs intéressants, les représentations condensées, et l'extraction itérative de motifs.
Dans la découverte des motifs intéressants, nous revenons sur le sujet qui se trouve au cœur de la découverte de connaissances dans les bases de données et qui a fait l'objet de plusieurs études dans la communauté des chercheurs. Nous partons du formalisme général de découverte de motifs intéressants défini par Mannila et al. [63], et nous montrons que ce formalisme peut s'adapter à différents contextes. Par la même occasion, nous parlerons des différentes approches qui ont été proposées pour l'optimisation de la découverte de motifs.
Dans la partie sur les représentations condensées, nous décrivons les différentes formes de représentations et les différentes approches qui ont été proposées.
L'extraction itérative de motifs est peut-être la partie la plus liée à notre mémoire. Comme nous l'avons déjà dit, le processus d'extraction de connaissances est un processus interactif et itératif complexe, pour lequel plusieurs théories doivent être calculées [13]. Pour cela, il est nécessaire de disposer d'un langage de requêtes qui permet à l'utilisateur de sélectionner des sous-ensembles de données, mais aussi de sélectionner des motifs. C'est ainsi qu'est apparu le concept de *Base de données inductive*, introduit pour la première fois par Mannila et al. [48] et ensuite repris dans [60, 61]. Nous présentons dans cette partie notamment des exemples de langages de requêtes qui rentrent dans ce cadre et nous montrons le lien qui existe entre l'extraction itérative de motifs et les bases de données inductives.

Le troisième chapitre est consacré à la contribution de la thèse. Nous présentons d'abord l'approche itérative d'extraction des motifs intéressants dans un formalisme logique, puis dans un formalisme algébrique, et ensuite nous décrivons plus brièvement notre contribution concernant la représentation condensée d'un ensemble de réponses à des requêtes.
Nous parlerons dans un premier temps de l'extraction itérative de requêtes conjonctives. Cette approche, qui utilise les représentations des anciennes extractions pour optimiser les futures, se distingue par l'introduction des vues dans l'expression des motifs. Dans cette approche, l'optimisation du calcul de la réponse à la nouvelle requête va se faire par des tests d'inclusions de requêtes, tests qui seront facilités par la nature conjonctive des requêtes et des vues.
L'extraction itérative de motifs avec un formalisme algébrique est la partie sur laquelle nous avons le plus travaillé dans cette thèse. Nous introduisons dans un premier temps un langage de manipulation d'ensembles de requêtes, et d'autre part, nous montrons comment optimiser le calcul d'un ensemble de requêtes en utilisant les ensembles de requêtes déjà calculés. De plus, dans la partie implémentation, nous détaillons les techniques d'optimisations utilisées.
La dernière partie de ce chapitre, consacrée à notre contribution concernant la représentation condensée d'un ensemble de réponses à des requêtes est brièvement décrite, le formalisme de notre approche étant présenté dans un article joint en annexe à ce mémoire.
Enfin, dans la conclusion, nous résumons les principales contributions proposées dans le mémoire et nous présentons un certain nombre de directions de recherches pouvant être envisagées à partir de nos travaux.

Chapitre II

Etat de l'art

1 Introduction

La découverte de motifs intéressants est aujourd'hui un sujet qui intéresse de nombreux chercheurs. Cependant, un certain nombre de problèmes se pose à elle. D'une part, la taille des données sur lesquelles l'extraction est effectuée est très grande, d'où des temps de calculs importants. C'est ainsi qu'un certain nombre d'algorithmes d'optimisation des algorithmes de base ont été proposés. Ensuite, le nombre de motifs découverts est généralement très important, ce qui a conduit à l'extraction de représentations condensées.
D'autre part, la découverte de motifs intéressants est un processus dans lequel l'utilisateur ne sait pas à priori ce qu'il cherche, et doit poser un certain nombre de requêtes pour trouver ce qui l'intéresse vraiment, d'où une itération du processus.
Cette partie du mémoire fait l'état de l'art sur les différents problèmes que nous venons d'évoquer. La première partie présente la découverte de motifs intéressants. La deuxième porte sur les représentations condensées de la réponse à une requête d'extraction, et le troisième sur l'extraction itérative de motifs.

2 Découverte de motifs intéressants

2.1 Introduction

La découverte de connaissances dans les bases de données, appelée aussi Data Mining est définie comme l'extraction d'informations intéressantes, non triviales, implicites, préalablement inconnues et potentiellement utiles à partir de grandes bases de données [36]. Elle occupe aujourd'hui une place importante dans le monde de l'entreprise et de la recherche pour le développement de systèmes d'aide à la décision. Dans ce cadre, la découverte de motifs intéressants (voir ci-dessous) a fait l'objet de nombreuses études. C'est ainsi qu'un formalisme général de découverte de motifs intéressants a été défini par Mannila et al [63]. Nous allons dans cette partie montrer que ce formalisme peut s'exprimer dans différents contextes, et qu'il peut être repris dans un contexte logique.
D'autre part, nous essayerons de mettre en relief l'apport de la logique dans la représentation et dans l'expressivité des motifs.
Cette partie est ainsi organisé comme suit : la section 2.2 fait une présentation du formalisme général. Les sections 2.3, 2.4 et 2.5 montrent respectivement comment ce formalisme s'exprime dans le cas de données stockées dans une table de transactions, dans une table relationnelle, dans une table multidimensionelle. La section 2.6 montre comment le formalisme s'exprime dans le cas de données stockées dans plusieurs tables. La section 2.6, qui utilise un formalisme logique, permettra de montrer la puissance d'expression de la logique du premier ordre.

2.2 Formalisme général de découverte de motifs fréquents

Dans le cadre de l'Extraction de connaissances à partir de Données, Mannila et Toivonen [63] ont proposé un formalisme général de découverte de motifs fréquents ou intéressants qui se définit comme suit : Etant donné un ensemble de données $\mathcal{S}$, un langage $\mathcal{L}$ d'expression des propriétés sur $\mathcal{S}$ appelés *motifs*, et un critère de qualité q défini pour tout motif $\varphi \in \mathcal{L}$ par : $q(\varphi, S) = true$ si et seulement si φ est intéressant, la recherche de motifs intéressants peut être vue comme le processus de construction d'une théorie $\mathcal{T}h(\mathcal{L}, \mathcal{S}, q)$, où

$$\mathcal{T}h(\mathcal{L}, \mathcal{S}, q) = \{\varphi \in \mathcal{L} \mid q(\varphi, S) = true\}.$$

Cette théorie est construite par l'intermédiaire d'un algorithme générique qui utilise une relation de spécialisation/généralisation entre les différents motifs.

Définition 2.1 - Ordre partiel.
Soit E un ensemble, et $\Re$ une relation binaire sur E. $\Re$ est un ordre partiel sur E si et seulement si
- *$\Re$ est réflexive, i.e., $\forall x \in E : x\Re x$*
- *$\Re$ est transitive, i.e. $\forall x, y, z \in E : x\Re y$ et $y\Re z \Rightarrow x\Re z$.*
- *$\Re$ est antisymétrique, i.e. $\forall x, y \in E : x\Re y$ et $y\Re x \Rightarrow x = y$.*

Définition 2.2 *Une relation de spécialisation $\preceq$ est un ordre partiel sur les motifs de $\mathcal{L}$.*
Soit φ et θ deux motifs de $\mathcal{L}$. On dit que φ est plus spécifique que θ si $\varphi \preceq \theta$; on dit aussi que θ est plus général que φ.

Définition 2.3 - Treillis, Semi-treillis.
Un ensemble de motifs $\mathcal{L}$, muni de la relation de spécialisation est un inf-semi-treillis, si chaque couple de motifs (φ_1, φ_2) admet un plus petit majorant commun unique, noté $\varphi_1 \vee \varphi_2$ qui est le motif le plus général qui soit plus spécifique que φ_1 et φ_2, i.e. $\varphi_1 \vee \varphi_2 = min_{\preceq}\{\varphi \in \mathcal{L} \mid \varphi_1 \preceq \varphi$ et $\varphi_2 \preceq \varphi\}$.
Un ensemble de motifs $\mathcal{L}$, muni de la relation de spécialisation est un treillis, si chaque couple de motifs (φ_1, φ_2) admet un plus petit majorant commun unique, $\varphi_1 \vee \varphi_2$ et un plus grand minorant commun, noté $\varphi_1 \wedge \varphi_2$ qui est le motif le plus spécifique qui soit plus général que φ_1 et φ_2, i.e. $\varphi_1 \wedge \varphi_2 = max_{\preceq}\{\varphi \in \mathcal{L} \mid \varphi \preceq \varphi_1$ et $\varphi \preceq \varphi_2\}$.

En général, $(\mathcal{L}, \preceq)$ est un treillis.

Définition 2.4 - Opérateur de spécialisation.
Un opérateur de spécialisation ρ pour un espace de recherche $\mathcal{L}$ est une application de $\mathcal{L}$ dans $2^{\mathcal{L}}$ telle que :
- *Pour tout $(\varphi, \varphi') \in \mathcal{L}^2$: $\varphi' \in \rho(\varphi)$, si $(\varphi' \prec \varphi)$ et il n'existe pas de $\varphi^* \in \mathcal{L}$ tel que $\varphi' \prec \varphi^* \prec \varphi$.*

Pour tout $X \subseteq \mathbb{L}$, on note :

$$\gamma(X) = \bigcup_{\varphi \in X} \rho(\varphi), \text{ et } \gamma^*(X) = \bigcup_{i=0}^{+\infty} \gamma^i(X).$$

$\gamma^*(X)$ regroupe l'ensemble des motifs obtenus par applications successives de ρ sur les différents motifs de X et leurs spécialisations. On l'appelle la clôture transitive de ρ.

Définition 2.5 - Complétude.
Un opérateur de spécialisation ρ est complet relativement à $L \subseteq \mathcal{L}$ si $\gamma^*(X) = L$, où $X = \{\varphi \in L \mid (\nexists \varphi' \in L)(\varphi' \succ \varphi)\}$.

Une relation de spécialisation sur $\mathcal{L}$ induit souvent une structure de treillis et un opérateur de spécialisation ρ sur $\mathcal{L}$. L'algorithme générique utilise ainsi cette relation de spécialisation pour le parcours du treillis .

Définition 2.6 - Monotonie du critère de qualité.
Le critère de qualité q, est défini pour tout motif $\varphi \in \mathcal{L}$ par : $q(\varphi, S) = true$ si et seulement si φ est intéressant.
q est monotone si pour tout $(\varphi, \gamma) \in \mathcal{L}^2$, si $\varphi \preceq \gamma$ et $q(\gamma, S) = true$, alors $q(\varphi, S) = true$.

Algorithme générique :
- **Entrée :** un ensemble de données S, un langage $\mathcal{L}$ muni d'une relation de spécialisation $\preceq$ et un prédicat de sélection q monotone.
- **Sortie :** l'ensemble $Th(\mathcal{L}, S, q)$.

1. $C_1 = \{\varphi \in \mathcal{L} \mid$ il n'existe aucun φ' tel $\varphi \prec \varphi'\}$;
2. i = 1 ;
3. **Tant que** $(C_i) \neq \emptyset$) **Faire**
4. // **Phase d'évaluation**
5. $F_i = \{\ \varphi \in C_i \mid q(\varphi, S)$ *est vrai*$\}$;
6. // **Phase de génération**
7. $C_{i+1} = \{\ \varphi' \in \rho(\varphi) \mid \varphi \in F_i\}$;
8. $C_{i+1} = \{\ \varphi' \in C_{i+1} \mid (\forall \varphi \in \mathcal{L})(\ \varphi' \in \rho(\varphi) \Rightarrow \varphi \in F_i)\}$;
9. i = i + 1 ;
10. **Fin Tant que** ;
11. **Retourner** $\cup_{j<i} F_j$;

FIG. II.1 – Algorithme générique de Mannila et Toivonen

Définition 2.7 - Anti-monotonie du critère de qualité.
Le critère de qualité q est anti-monotone si pour tout $(\varphi, \gamma) \in \mathcal{L}^2$, si $\varphi \preceq \gamma$ et $q(\varphi, S) = true$, alors $q(\gamma, S) = true$.

L'algorithme générique de construction de $\mathcal{T}h(\mathcal{L}, \mathcal{S}, q)$ fonctionne ainsi avec un critère de qualité anti-monotone, dont la propriété peut être reprise comme suit : si un motif φ est intéressant, alors tous les motifs plus généraux que lui le sont aussi. Ceci est l'idée de base de la proposition suivante : [56, 63, 67]

Proposition 2.1 *Si $(\mathcal{L}, \preceq)$ est un treillis et q est anti-monotone par rapport à $\preceq$, alors $\mathcal{T}h(\mathcal{L}, \mathcal{S}, q)$ est un inf-semi-treillis.*

L'algorithme parcourt niveau par niveau le treillis $(\mathcal{L}, \preceq)$ et construit les motifs intéressants en partant des motifs les plus généraux (Figure II.1), en alternant :
- une phase d'évaluation des motifs candidats de niveau i. Durant cette phase, seuls les candidats qui vérifient le critère de qualité sont retenus (cf. ligne 5).
- une phase de génération des motifs candidats de niveau $i+1$ à partir des motifs intéressants de niveau i. Cette phase se décompose en deux : dans un premier temps, les motifs intéressants de niveau i sont spécialisés (cf. ligne 7). Ensuite ne sont retenus que les motifs dont tous les sous-motifs sont intéressants (cf. ligne 8). Ces deux phases sont respectivement appelées *phases de spécialisation* et *d'élagage*.

Selon le formalisme utilisé pour la représentation des données, on peut avoir différents types de motifs. C'est ce qui justifie la classification des motifs suivant le type de données que nous proposons ci-dessous :
- Extraction de motifs intéressants à partir de données stockées dans une table.
- Extraction de motifs intéressants à partir de données stockées dans plusieurs tables.

Nous allons donc étudier les motifs intéressants selon cette classification, et dans chaque cas, nous préciserons les points suivants :

- Les données.
- La forme des motifs.
- Les critères de sélection.
- Les algorithmes.

Nous allons présenter l'extraction de motifs intéressants à partir de données stockées dans une table sous plusieurs formes : le cas transactionnel, le cas d'une table relationnelle et le cas multidimensionnel. Ensuite nous verrons l'extraction des motifs intéressants à partir de données stockées dans plusieurs tables.

2.3 Cas transactionnel

L'extraction de motifs intéressants a fait l'objet de plusieurs travaux qui prennent principalement leur source des travaux de Agrawal et al. [2]. Cependant, nous utiliserons dans cette section un formalisme introduit par [37] pour la représentation des données et des motifs. Nous introduirons dans un premier temps ce formalisme qui est souvent appelé formalisme $\mathbb{ORP}$. Signalons que nous ferons souvent référence à ce formalisme dans ce mémoire. Ensuite nous verrons comment l'utiliser dans différents cas de données stockées dans une table et enfin les algorithmes utilisés.

2.3.1 Formalisme $\mathbb{ORP}$

Le formalisme $\mathbb{ORP}$ sur lequel nous allons travailler se définit comme suit :
Soit $\mathbb{O}$ un ensemble fini d'objets et $\mathbb{P}$ un ensemble fini de propriétés. Soit $\mathbb{R} \subseteq \mathbb{O} \times \mathbb{P}$ une relation binaire, où $(o, p) \in \mathbb{R}$ veut dire que l'objet o a la propriété p. Le triplet $\mathbb{D} = (\mathbb{O}, \mathbb{P}, \mathbb{R})$ est appelé ensemble de données.
Un sous-ensemble P de $\mathbb{P}$ est appelé motif. C'est une conjonction de propriétés.

Définition 2.8 *Un motif P est inclus dans un objet $o \in \mathbb{O}$, si $(o, p) \in \mathbb{R}$ pour tout $p \in P$. On note f la fonction qui fait correspondre à un motif l'ensemble des objets qui le contiennent, et g sa fonction duale :*

$$(\forall X \subseteq \mathbb{P})\ f(X) = \{\ o \in \mathbb{O} \mid \forall x \in X, (o, x) \in \mathbb{R}\}$$
$$(\forall O \subseteq \mathbb{O})\ g(O) = \{\ x \in X \mid \forall\ o \in O, (o, x) \in \mathbb{R}\}$$

L'ensemble de données $\mathbb{D}$ peut être représenté sous la forme d'une base de données transactionnelle (voir ci-dessous) ou relationnelle.

2.3.2 Les données

Dans ce cadre :

- les objets de l'ensemble $\mathbb{O}$ sont des identifiants de transactions.
- l'ensemble des propriétés $\mathbb{P}$ est un ensemble d'items (ou articles) noté $\mathbb{I}$.

Enfin, on appelle *base de données transactionnelles* un ensemble de transactions Bd, une transaction étant un couple (o, X) où $o \in \mathbb{O}$ et $X \subseteq \mathbb{I}$. o est un identifiant de transaction(TID) et X est l'ensemble des items associés à $\mathbb{O}$. Une base de données transactionnelle consiste généralement en un fichier où chaque enregistrement représente une transaction [45].
L'ensemble de données est ainsi le triplet $\mathbb{D} =< \mathbb{O}, \mathbb{I}, \mathbb{R} >$, où $\mathbb{O} = \{o_1, o_2, \ldots, o_n\}$, $\mathbb{I} = \{i_1, i_2, \ldots, i_p\}$ et $\mathbb{R}$ est défini par : $(o, i) \in \mathbb{R}$ si $(\exists (o, X) \in Bd)(i \in X)$.

Exemple 2.1 *La base de données transactionnelle représentée à la figure II.2 est une représentation de l'ensemble de données $\mathbb{D}$ défini par :*

- $\mathbb{I} = \{\ A, B, C, D, E, F\ \}$.

TID	Liste d'items
T1	A D F
T2	A B D E
T3	B C
T4	F
T5	D E

FIG. II.2 – Base de données transactionnelles

TID	item
T1	A
T1	D
T1	F
...	...

TID	A	B	C	D	E	F
T1	1	0	0	1	0	1
T2	1	1	0	1	1	0
T3	0	1	1	0	0	0
T4	0	0	0	0	0	1
T5	0	0	0	1	1	0

FIG. II.3 – Représentation d'une base de données transactionnelles sous forme relationnelle (à gauche) ou sous forme binaire (à droite).

- $\mathbb{O} = \{\ T1, T2, T3, T4, T5\ \}$.
- $\mathbb{R} = \{\ (T_1, A), (T_1, D), (T_1, F), ...\ \}$

Cette base de données sera utilisée par la suite comme exemple de référence.

Les transactions peuvent être stockées dans une table, avec un enregistrement par transaction, comme dans la figure II.2.
Etant donné que les bases de données ne supportent pas de types de données ensemblistes, une base de données transactionnelles est habituellement stockée dans un fichier sous un format similaire à celui de la figure II.2, ou dans une relation (figure II.3) soit sous forme relationnelle soit sous la forme binaire .

2.3.3 La forme des motifs

Il y a plusieurs types de motifs. Nous parlerons de ceux classsiques : les itemsets et les règles d'association.

Définition 2.9 *Un itemset est un sous-ensemble de* $\mathbb{I}$. *Un itemset de taille* k *est un* $k-itemset$.

Définition 2.10 *Une règle d'association est une implication de la forme* $X \Longrightarrow Y$, *où* $X \subseteq \mathbb{I}$, $Y \subseteq \mathbb{I}$, *et* $X \cap Y = \emptyset$.

Pour étudier la qualité des motifs, on utilise généralement deux mesures d'intérêt : le support et la confiance.

Définition 2.11 - Support d'un itemset.
Soit $\mathbb{D} =< \mathbb{O}, \mathbb{I}, \mathbb{R} >$ *un ensemble de données. Le support d'un itemset* $X \subseteq \mathbb{I}$ *est défini par la formule suivante :*

$$sup(X, \mathbb{D}) = \mid f(X) \mid / \mid \mathbb{O} \mid.$$

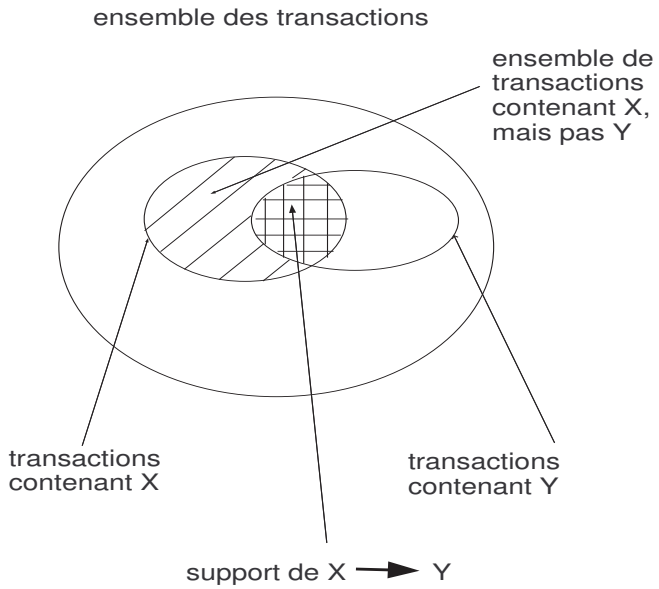


FIG. II.4 – Diagramme des probabilités $P(X, Y)$ et $P(X, \neg Y)$

Si on suppose que l'ensemble de données est fixe, on peut noter $sup(X)$ le support d'un motif X. On dit que la règle $X \Longrightarrow Y$ a pour confiance c si $c\%$ des transactions dans $\mathbb{D}$ qui contiennent X contiennent aussi Y. Elle a pour support s si $s\%$ des transactions dans $\mathbb{D}$ contiennent $X \cup Y$. En termes probabilistes, le support et la confiance s'expriment ainsi comme suit :

Définition 2.12 - Support d'une règle.
Etant donné un ensemble de données $\mathbb{D} =< \mathbb{O}, \mathbb{I}, \mathbb{R} >$*, et une règle d'association* $X \Longrightarrow Y$*, le support de* $X \Longrightarrow Y$ *noté* $sup(X \Longrightarrow Y)$ *est défini par :*

$$sup(X \Longrightarrow Y) = P(X \cup Y) = \mid f(X \cup Y) \mid / \mid \mathbb{O} \mid.$$

Définition 2.13 - Confiance d'une règle.
Etant donné un ensemble de données $\mathbb{D} =< \mathbb{O}, \mathbb{I}, \mathbb{R} >$*, et une règle d'association* $X \Longrightarrow Y$*, la confiance de* $X \Longrightarrow Y$*, notée* $conf(X \Longrightarrow Y)$ *est défini par :*

$$conf(X \Longrightarrow Y) = P(Y/X) = sup(X \cup Y)/sup(X).$$

Intuitivement, le support exprime la fréquence avec laquelle X et Y apparaissent ensemble, et la confiance exprime la fréquence d'apparition de Y sachant X.
Il existe cependant plusieurs autres mesures. Nous en citerons quelques unes, le lecteur intéressé pourra les trouver dans [53, 58, 54, 12, 17].
Nous nous baserons sur le schéma de la figure II.4 qui nous aidera à mieux comprendre ces mesures. Dans ce schéma $\neg Y$ signifie le complémentaire de Y, et $P(\neg Y) = 1 - P(Y)$.
Le support de $X \Rightarrow Y$ est défini ici par $P(X \wedge Y)$. C'est la probabilité que X et Y soient tous les deux vrais, i.e. qu'une transaction contienne à la fois X et Y. La couverture d'une règle $X \Rightarrow Y$ est la proportion des transactions dans l'ensemble de données contenant X.

Définition 2.14 - Couverture.
Etant donné un ensemble de données $\mathbb{D} =< \mathbb{O}, \mathbb{I}, \mathbb{R} >$*, et une règle d'association* $X \Longrightarrow Y$*, la*

couverture de $X \Longrightarrow Y$*, notée* $Couv(X \Longrightarrow Y)$ *est défini par :*

$$Couv(X \Longrightarrow Y) = P(X).$$

La mesure d'intérêt, proposée par Brin et al. dans [12, 17] spécifie le degré de dépendance entre X et Y en prenant en compte la probabilité d'occurrence de Y.

Définition 2.15 - Intérêt.
Etant donné un ensemble de données $\mathbb{D} =< \mathbb{O}, \mathbb{I}, \mathbb{R} >$*, et une règle d'association* $X \Longrightarrow Y$*, l'intérêt de* $X \Longrightarrow Y$*, noté* $Int(X \Longrightarrow Y)$ *est défini par :*

$$Int(X \Longrightarrow Y) = P(X \wedge Y) \ / \ (P(X) * P(Y)).$$

Cette mesure a été utilisée par Brin et al [17] à la place de la confiance afin de générer des règles d'associations à partir de l'ensemble des itemsets fréquents. Les règles d'associations générées sont celles dont l'intérêt est supérieur à 1.
Introduite par Brin et al. [17], la mesure de conviction permet de mesurer la déviation entre le corps (X) et la négation de la tête. La conviction est une mesure qui essaie de prendre en compte les avantages de la confiance et de l'intérêt. Elle mesure le degré d'implication entre le corps et la tête.

Définition 2.16 - Conviction.
Etant donné un ensemble de données $\mathbb{D} =< \mathbb{O}, \mathbb{I}, \mathbb{R} >$*, et une règle d'association* $X \Longrightarrow Y$*, la conviction de* $X \Longrightarrow Y$*, notée* $Conv(X \Longrightarrow Y$ *est définie comme suit :*

$$Conv(X \Longrightarrow Y) = P(X) * P(\neg Y) \ / \ P(X, \neg Y).$$

2.3.4 Les critères de sélection

Pour les critères de sélection, nous allons considérer les deux mesures les plus utilisées : le support et la confiance.

Définition 2.17 *Etant donné un seuil minimum de support minsup, un itemset* X *est dit fréquent si* $sup(X) \geq minsup$*.*

Définition 2.18 *Etant donné un seuil minimum de confiance* $minconf$*, une règle d'association* $X \Longrightarrow Y$ *est dite intéressante si* $sup(X \cup Y) \geq minsup$ *et* $conf(X \Longrightarrow Y) \geq minconf$*.*

Le problème posé est le suivant : étant donné deux seuils fixés a priori par l'utilisateur, plus précisément un seuil minimal de support et un seuil minimal de confiance, il s'agit de trouver toutes les règles d'association intéressantes, c'est à dire qui satisfont aux critères de qualité cités ci-dessus.

2.3.5 Les algorithmes

La découverte de règles d'association passe par la découverte des itemsets fréquents. L'extraction des règles se fait en deux étapes :

- la recherche des itemsets fréquents, i.e. dont le support est supérieur au seuil de support fixé.
- La recherche à partir de ces itemsets des règles intéressantes, i.e. dont la confiance est supérieure au seuil de confiance fixé par l'utilisateur.

Algorithme Apriori :
– **Entrée :** un ensemble de données $\mathbb{D}$, un seuil de support $minsup$;
– **Sortie :** l'ensemble $Freq_{minsup}(\mathbb{D})$.

1. $C_1 = \{ \{x\} \mid x \in \mathbb{I} \}$;
2. i = 1 ;
3. **Tant que** $(C_i \neq \emptyset)$ **Faire**
4. // **Phase d'évaluation**
5. $F_i = \{ X \in C_i \mid sup(X) \geq minsup \}$;
6. // **Phase de génération**
7. $C_{i+1} = \{ X1 \cup X2 \mid X1 \in F_i,\ X2 \in F_i,\ \mid X1 \cap X2 \mid = i-1 \}$;
8. $C_{i+1} = \{ X \in C_{i+1} \mid (\forall x \in X)(\ X \backslash \{x\} \in F_i) \}$;
9. i = i + 1 ;
10. **Fin Tant que** ;
11. **Retourner** $\cup_{j<i} F_j$;

FIG. II.5 – Algorithme Apriori

La performance de l'extraction de règles d'association est surtout déterminée par la première phase puisque c'est elle la plus coûteuse. La deuxième phase ne nécessite pas d'accés à la base de données, car tous les supports sont déjà calculés. L'algorithme de base pour l'extraction des itemsets fréquents est l'algorithme Apriori. Il a été introduit par [2] en 1994 et demeure le fondement d'un certain nombre d'algorithmes qui ont ensuite été proposés.

Algorithme Apriori

L'algorithme Apriori est une instance de l'algorithme générique de Mannila et Toivonen. Il calcule tous les itemsets fréquents et à partir d'eux, trouve toutes les règles d'associations intéressantes. La première étape de la découverte des règles d'associations consiste ainsi à construire la théorie $\mathcal{T}h(2^{\mathbb{I}}, \mathbb{D}, sup(X/\mathbb{D}) \geq minsup)$. Soit $Freq_\alpha(\mathbb{D}) = \{ X \subseteq \mathbb{I} \mid sup(X, \mathbb{D}) \geq \alpha \}$ l'ensemble des itemsets dont le support est supérieur ou égal à α. Alors on a :

$$\mathcal{T}h(2^{\mathbb{I}}, \mathbb{D}, sup(X/\mathbb{D}) \geq minsup) = Freq_{minsup}(\mathbb{D}).$$

La relation de spécialisation est ici la relation d'inclusion ensembliste. L'algorithme est basé sur l'anti-monotonie du critère de sélection $sup(X) \geq minsup$ qui se décrit comme suit :

Proposition 2.2 *Tout sous-ensemble d'un itemset fréquent est fréquent.*
$\forall X \in Freq_{minsup}(\mathbb{D})$, *si* $Y \subseteq X$, *alors* $Y \in Freq_{minsup}(\mathbb{D})$.

Cette proposition implique que tout sur-ensemble d'un itemset non fréquent est non fréquent.
L'algorithme Apriori est décrit à la figure II.5.
Il faut signaler que dans ce cas l'opérateur de spécialisation ρ est défini pour tout $X \subseteq \mathbb{I}$ par :

$$\rho(X) = \{ X \cup \{x\} \mid x \in \mathbb{I} \backslash X \}.$$

Cependant il est plus efficace de générer les candidats de niveau $i+1$ par combinaison de deux fréquents de niveau i dans un premier temps (voir ligne 7 de la figure II.5). La phase

de spécialisation porte ici le nom de *phase de jointure*. La phase d'élagage consiste à vérifier que tous les sous-ensembles du candidat généré sont fréquents (ligne 8 de la figure II.5).

Exemple 2.2 *Supposons que* $F_3 = \{ \{ABC\}, \{ABD\}, \{ACD\}, \{ACE\}, \{BCD\} \}$. *Après la phase de jointure* $C_4 = \{ \{ABCD\}, \{ACDE\} \}$, *et après la phase d'élagage* $C_4 = \{ \{ABCD\} \}$.

Autant dire que la méthode d'évaluation est fondamentale puisqu'elle est déterminante dans la performance de l'algorithme. En effet, l'évaluation, contrairement à la génération, nécessite un accès à la base de données pour le calcul des supports, ce qui demande du temps. L'évaluation des candidats d'un niveau donné se fait en une seule passe (parcours) de la base de données.

Proposition 2.3 - Nombre de passes *Si d est la taille maximale des motifs, l'algorithme générique parcourt l'ensemble des données au plus* $d + 1$ *fois.*

Dans le souci d'optimiser l'élagage et l'évaluation, les itemsets candidats sont en général stockés dans un arbre de hachage. Un nœud de cet arbre contient soit une liste d'itemsets (s'il s'agit d'une feuille de l'arbre), soit une table de hachage (s'il s'agit d'un nœud interne). Chaque case d'une table de hachage au niveau k de l'arbre pointe sur un nœud de niveau $k + 1$, la racine de l'arbre se trouvant au niveau 1. Lors de l'ajout d'un itemset dans l'arbre, un parcours en profondeur est effectué, en partant de la racine de l'arbre jusqu'à ce qu'une feuille soit atteinte. Quand un nœud interne de j-ème niveau de l'arbre est atteint, le nœud suivant à parcourir est déterminé en appliquant une fonction de hachage sur le j-ème item de l'itemset. Au départ, on est en présence d'une racine sans feuilles. Lorsque le nombre d'itemsets stockés dans une feuille dépasse un certain seuil N (plus de N candidats), ce nœud est scindé en P, où P est le nombre de valeurs possibles de la fonction de hachage. On répartit alors les candidats dans les P nouvelles feuilles.
Durant la phase d'élagage, il faut vérifier que tous les sous-ensembles des motifs candidats sont fréquents. La recherche de ces sous-ensembles va se faire en utilisant le hachage qui permettra de les retrouver rapidement. De même, lors de la phase d'évaluation, le hachage permet de trouver rapidement les candidats qui sont potentiellement inclus dans une transaction donnée.

Exemple 2.3 *Soit N=2, P=2, h(A) = 1 ; h(B) = 2 ; h(C) = 1 ; h(D) = 2 ;*
On suppose qu'on ajoute successivement les itemsets AB, AC, AD puis BC (voir figure II.6). Dès que $N > 2$, *alors la première feuille est éclatée en deux, mais le premier item des itemsets AB, AC et AD est le même, ce qui fait qu'ils se retrouvent dans la même feuille. Celle-ci est découpée en deux et le hachage sur le 2-ème item les répartit dans les feuilles correspondantes.*
Donnons maintenant un exemple d'utilisation lors de la phase d'élagage. Etant donné le candidat ABC, il faut vérifier si tous ses sous-ensembles sont fréquents. La recherche d'un itemset se fera simplement par le hachage sur le 1-er puis sur le 2-ème item des 3 sous-ensembles de 2 items (AB, AC et BC).

La découverte de règles d'associations, comme nous l'avons déjà dit, consiste en deux étapes. La première étape, que nous venons de voir, consiste à trouver l'ensemble des itemsets fréquents. La deuxième étape consiste à trouver l'ensemble des règles d'associations intéressantes à partir des itemsets fréquents. Il s'agit ainsi de construire la théorie $\mathcal{T}h(R, \mathbb{D}, q)$, où $R = \{X \Longrightarrow Y \mid X, Y \in Freq_{minsup}(\mathbb{D})$ *et* $X \cap Y = \emptyset\}$ est l'ensemble des règles d'associations $X \Rightarrow Y$ telles que :

- $X \cup Y$ est fréquent et
- $q(X \Longrightarrow Y, \mathbb{D})$ est vrai ssi $Conf(X \Longrightarrow Y, \mathbb{D}) \geq minconf$.

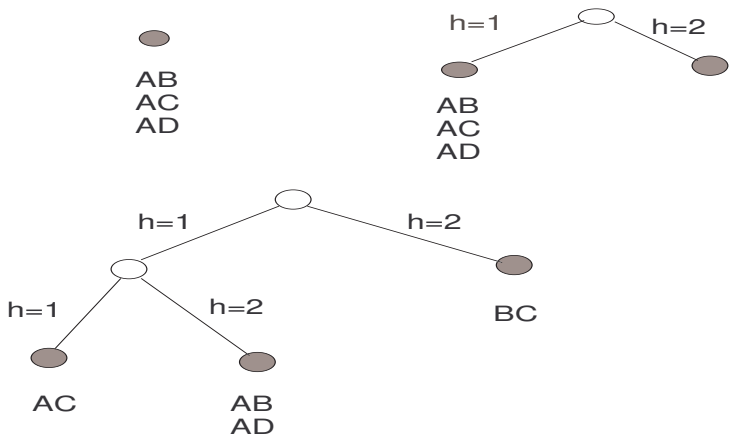


FIG. II.6 – Exemple de construction d'arbre de hachage

Optimisations de Apriori

Un certain nombre d'optimisations de l'algorithme Apriori ont été proposées. Deux idées fondamentales se dégagent. La première consiste à réduire le nombre de passes dans la base, et la seconde à trouver une partie de l'ensemble des fréquents et d'en déduire les autres.

- **Par la réduction du nombre de passes dans la base :**
  - **Technique de hachage**
    Pour réduire la taille des ensembles de k-itemsets candidats C_k ($k > 1$), Park et al. [74] ont proposé avec l'algorithme DHP (Direct Hashing and Pruning) une technique, basée sur les tables de hachage, qui permet de générer beaucoup moins de 2-itemsets candidats, ce qui va globalement accélérer la phase de calcul. Lors du parcours des données pour la génération de l'ensemble F_1 des 1-itemsets fréquents à partir de l'ensemble C_1 des 1-itemsets candidats, tous les 2-itemsets sont générés pour chaque transaction. Ensuite, une fonction de hachage est utilisée pour les répartir dans différents buffers (buckets) d'une table de hachage (voir figure II.7). Chaque buffer a un compteur qui est incrémenté à chaque insertion d'un itemset. La valeur de ce compteur est ainsi un majorant du support de tous les 2-itemsets qui se trouvent dans ce bucket. Par conséquent, un 2-itemset dont le compteur du buffer correspondant dans la table de hachage est inférieur au seuil de support ne peut être fréquent, et sera donc détruit de l'ensemble des candidats.
    La figure II.7 présente un exemple de table de hachage pour les 2-itemsets candidats. Cette table a été générée en utilisant notre exemple de référence (Figure II.2) lors du calcul de F_1 à partir de C_1. Si le seuil de support minimal est égal à 2 par exemple, alors les itemsets des buffers 1 et 5 ne peuvent être fréquents, et donc ne peuvent être rajoutés à C_2. On obtient ainsi $C_2 = \{ \{AD\}, \{AF\}, \{BD\}, \{DE\}, \{DF\}, \{BE\} \}$ au lieu de $C_2 = \{ \{AB\}, \{AD\}, \{AE\}, \{AF\}, \{BD\}, \{BE\}, \{BF\}, \{DE\}, \{DF\}, \{EF\} \}$ obtenu avec l'algorithme Apriori.
  - **Partitionnement des données** [82]
    L'idée de cette méthode est de partitionner les données, c'est à dire de les répartir en ensembles disjoints, de telle sorte que l'on n'effectue que deux parcours de la base de données pour trouver les itemsets fréquents (figure II.8). Pour cela, la méthode se divise en deux phases. Dans la première phase, l'algorithme subdivise les transactions

Adresse des buffers	0	1	2	3	4	5	6
Compteur des buffers	2	1	2	2	2	1	0
Contenu des buffers	AD AD	AE	AF BC	BD DE	DF BE	AB	

FIG. II.7 – Table de hachage pour les 2-itemsets candidats

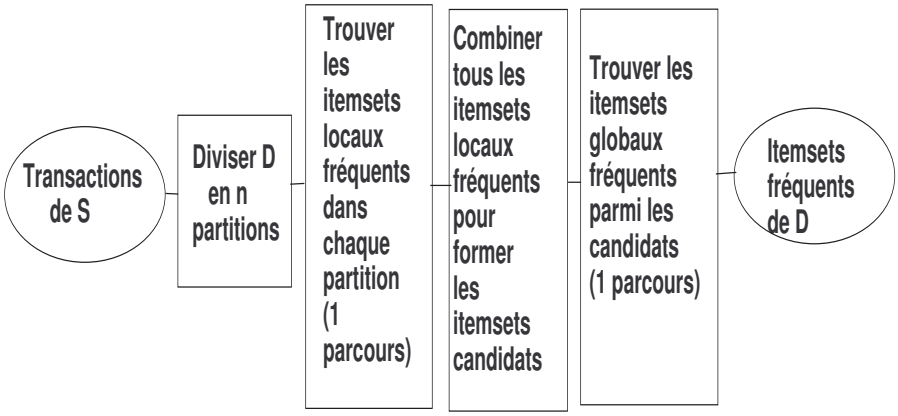


FIG. II.8 – Technique de partitionnement

de l'ensemble de données $\mathbb{D}$ en n parties disjointes. La taille d'une partie et le nombre de parties sont choisis de sorte que chaque partie puisse tenir en mémoire. Si le seuil de support minimum pour les transactions dans $\mathbb{D}$ est *minsup*, alors le seuil de support minimum pour une partie est $minsup \times (le\ nombre\ de\ transactions\ dans\ la\ partition)$. Tous les itemsets fréquents d'une partie, appelés *itemsets fréquents locaux* sont d'abord trouvés. L'algorithme utilise une structure de données spéciale qui enregistre pour chaque itemset les identifiants (TID) des transactions contenant les items de l'itemset. Cela permet de trouver tous les k-itemsets fréquents locaux, pour $k = 1, 2, \ldots$, et cela en un seul parcours. Un itemset localement fréquent n'est pas forcément fréquent par rapport à l'ensemble de données $\mathbb{D}$. Cependant, *tout itemset potentiellement fréquent dans* $\mathbb{D}$ *est forcément fréquent dans l'une au moins des parties.* Par conséquent, tous les itemsets fréquents locaux seront candidats par rapport à $\mathbb{D}$. L'ensemble des itemsets fréquents de toutes les parties forme l'ensemble des *itemsets candidats globaux* par rapport à $\mathbb{D}$.
Dans la deuxième phase, un second parcours de $\mathbb{D}$ est effectué pour calculer les supports des itemsets qui sont candidats globaux.

– **Echantilllonnage** [87]
Cette technique est aussi appelée technique de fenêtrage. L'idée est d'effectuer la recherche des itemsets fréquents sur un sous-ensemble D de $\mathbb{D}$ choisi de façon aléatoire. Un choix est ainsi fait pour l'efficacité au détriment de la précision. La taille de D est choisie de sorte que la recherche des itemsets puisse se faire en mémoire, et ainsi un seul parcours des transactions de D est effectué. Puisque la recherche s'effectue sur D et non

OID	Produit	Magasin	Adresse
1	Lait	Auchan	Blois
2	Lait	Carrefour	Paris
3	Beurre	Super U	Blois
4	Lait	Auchan	Nantes
5	Sucre	Carrefour	Nantes
6	Fromage	Auchan	Paris
7	Lait	Carrefour	Nantes
8	Beurre	Auchan	Blois
9	Fromage	Carrefour	Paris
10	Lait	Carrefour	Blois

FIG. II.9 – Table classique

sur $\mathbb{D}$, il est possible que certains itemsets soient omis. Pour réduire cette possibilité, la recherche des itemsets fréquents dans D est effectuée avec un seuil de support $minsup_2$ plus petit que le seuil de départ $minsup_1$. Pour calculer les fréquents sur $\mathbb{D}$ complet, on ajoute alors aux fréquents de $Freq_{minsup_2}(D)$ la frontière négative de celle-ci.

Définition 2.19 Frontière négative
La frontière négative d'un ensemble S dans $\mathcal{L}$ est l'ensemble des motifs les plus généraux qui ne sont pas dans S. Elle est notée $Bd^-(S)$.

$$Bd^-(S) = \{\ \varphi \in \mathcal{L} \backslash S \mid (\forall\ \psi \in \mathcal{L}),\ (\varphi \prec \psi \Rightarrow \varphi \in S)\}.$$

On fait ainsi une passe sur $\mathbb{D}$ pour calculer les supports de tous les éléments de $Freq_{minsup_2}(D)$ et des éléments de sa frontière négative. Si aucun élément de cette frontière négative n'est fréquent par rapport au seuil $minsup_1$, alors on a trouvé tous les itemsets fréquents et leurs supports exacts. Si malheureusement un élément de cette frontière négative est fréquent, on déclare un échec. Dans ce cas, tout le processus doit être répété (à moins que l'on accepte les résultats obtenus et que l'on fasse abstraction des quelques éléments manquants).

– **Par la réduction du nombre de fréquents générés**
L'idée générale consiste à trouver une partie de l'ensemble des fréquents et d'en déduire les autres. Cet ensemble des fréquents est appelé représentation condensée. Etant donné que nous avons une partie consacrée aux représentations condensées, nous parlerons de façon plus précise de celles-ci dans la partie sur les représentations condensées.

2.4 Cas d'une table relationnelle

Nous appelerons table classique une table vue sous la forme d'une relation. Par la suite, il s'agit d'une table comportant comme premier attribut, un attribut identifiant les numéros de ligne (identifiants de tuples) de la table (figure II.9).

2.4.1 Les données

L'ensemble $\mathbb{O}$ des objets est l'ensemble des numéros de ligne de la table (attribut OID), i.e., $\mathbb{O} = \{\ o_1, o_2, \ldots, o_n\ \} = \{\ 1, 2, \ldots, n\ \}$.

Etant donné que dans ce cas, on est en présence d'attributs qui prennent leurs valeurs dans des domaines, l'ensemble $\mathbb{P}$ est l'ensemble défini comme suit :

$$\mathbb{P} = \{\ A = a \mid A \in \mathbb{A}, a \in dom(A)\ \}$$

où $\mathbb{A}$ représente l'ensemble des attributs, et $dom(A)$ représente le domaine d'un attribut A. Une propriété est ainsi notée par $A = a$. Dans [49], de telles propriétés sont appelées descripteurs. Dans de cadre, la relation $\mathbb{R}$ est définie par : $(\forall o \in \mathbb{O})(\forall (A = a) \in \mathbb{P})$, $o\mathbb{R}(A = a)$ s'il existe une ligne d'identifiant o prenant la valeur a pour l'attribut A. Notons que s'il existe une dépendance fonctionnelle entre l'attribut OID et un attribut A ($OID \rightarrow A$), alors un objet o ne peut pas avoir deux propriétés avec le même attribut A, i.e. si $o_1\mathbb{R}(A = a)$ et $o_1\mathbb{R}(B = b)$, alors $A \neq B$. L'ensemble de données obtenu est ainsi le triplet $\mathbb{D} =< \mathbb{O}, \mathbb{P}, \mathbb{R} >$.

Voici quelques exemples d'illustration de la relation $\mathbb{R}$:

Exemple 2.4 *On considère la table de la figure II.9 :*

$$o_1\ \mathbb{R}\ (Produit = Lait)$$
$$o_1\ \mathbb{R}\ (Magasin = Auchan)$$
$$o_3\ \mathbb{R}\ (Adresse = Blois)$$

On ne peut pas avoir $o_1\mathbb{R}(Produit = Beurre)$ *s'il existe une dépendance fonctionnelle* $OID \rightarrow Produit$.

2.4.2 La forme des motifs

L'ensemble des motifs est l'ensemble des parties de $\mathbb{P}$. Le tableau de la figure II.9 pourrait être dessiné sous la forme suivante (figure II.10), où P_i représente la propriété $(Produit = i)$, M_i représente la propriété $(Magasin = i)$, et A_i représente la propriété $(Adresse = i)$. Cet exemple montre que l'on obtient ainsi une table similaire au cas transactionnel, où les items sont remplacés par des propriétés.
Nous distinguerons deux types de motifs : les $k - descripteurs$ qui sont des ensembles de k propriétés, et les règles d'association sont des implications entre ces ensembles.

Définition 2.20 *Un* $k - descripteur$ *est un sous-ensemble de* $\mathbb{P}$ *de* k *éléments.*

Définition 2.21 *Une règle d'association est une implication de la forme* $X \Rightarrow Y$, *où* $X \subseteq \mathbb{P}$ *et* $Y \subseteq \mathbb{P}$.

Voici quelques exemples de motifs : $(Produit = Lait, Magasin = Auchan)$, $(Magasin = Carrefour, Adresse = Paris)$ et de règle : $(Produit = Lait) \Rightarrow (Adresse = Blois)$. On peut dire ainsi qu'on obtient les mêmes motifs que dans le cas transactionnel et que les calculs du support et de la confiance se feront de la même manière. Les critères de sélection seront les mêmes. Il y a cependant une particularité que nous allons voir ci-dessous.

2.4.3 Discussion

Dans le cas où des dépendances fonctionnelles existent entre l'attribut OID et les attributs de $\mathbb{A}$, nous montrons que le nombre de candidats peut être réduit. Pour cela, nous rappelons la notion de partition d'un ensemble.

1	P_{Lait}	M_{Auchan}	A_{Blois}
2	P_{Lait}	$M_{Carrefour}$	A_{Paris}
3	P_{Beurre}	M_{SuperU}	A_{Blois}
4	P_{Lait}	M_{Auchan}	A_{Nantes}
5	P_{Sucre}	$M_{Carrefour}$	A_{Nantes}
6	$P_{Fromage}$	M_{Auchan}	A_{Paris}
7	P_{Lait}	$M_{Carrefour}$	A_{Nantes}
8	P_{Beurre}	M_{Auchan}	A_{Blois}
9	$P_{Fromage}$	$M_{Carrefour}$	A_{Paris}
10	P_{Lait}	$M_{Carrefour}$	A_{Blois}

FIG. II.10 – Table classique vue sous forme transactionnelle

Définition 2.22 Partition. *Soit E un ensemble. On appelle partition de E tout ensemble de parties E_i de $E(i = 1, \ldots, n)$ tel que $\bigcup_{i=1}^{n} E_i = E$ et $(\forall i, j)(i \neq j) \Rightarrow E_i \cap E_j = \emptyset$.*

Si l'on regroupe les propriétés par attribut, alors on obtient une partition de $\mathbb{P} = \bigcup_{i=1}^{n} \mathbb{P}_{A_i}$, où $\mathbb{P}_{A_i} = \{\ A_i = a | a \in dom(A_i)\ \}$.

Proposition 2.4

Pour tout $A_i \in \mathbb{A}$ tel que $OID \rightarrow A_i$, on a $: (\forall x, x' \in \mathbb{P}_{A_i})(x \neq x' \Rightarrow f(\{x\}) \cap f(\{x'\}) = \emptyset)$.

Cela veut dire que si on considère un motif X comportant deux propriétés d'une même partie $\mathbb{P}_{A_i}$, alors son support est nul. En effet, on a :
$sup(X) = \mid f(X) \mid / \mid \mathbb{O} \mid = \mid \bigcap_{x \in X} f(\{x\}) \mid / \mid \mathbb{O} \mid = 0$.
Il est ainsi inutile de considérer ces candidats là, ce qui réduit l'espace de recherche.
L'extraction des motifs dans une table classique est ainsi un cas particulier de l'extraction dans une base de données transactionnelles.

2.5 Cas d'une table multidimensionnelle

Une table multidimensionnelle est définie par un ensemble d'attributs appelés dimensions, qui sont des critères d'analyse (attributs catégoriques dans les bases de données statistiques) et un attribut numérique appelé mesure. Un exemple d'une telle table est donnée à la figure II.11. Le nombre de dimensions dans de telles tables est généralement très important, ce qui rend la découverte de motifs intéressants plus complexe. Les travaux [10, 57] que nous allons présenter effectuent un parcours du treillis en profondeur et non en largeur pour la découverte de motifs intéressants.

2.5.1 Les données

Nous étendons le modèle $\mathbb{ORP}$ au cas multidimensionnel. On ajoute à l'ensemble de données obtenu dans le cas classique une quatrième composante m, qui est une fonction de $\mathbb{O}$ dans l'ensemble des réels $\mathcal{R}$ appelée **mesure**. Cette mesure va permettre de prendre en compte différents critéres de sélection.
L'ensemble de données devient ainsi un quadruplet $< \mathbb{O}, \mathbb{P}, \mathbb{R}, m >$.

Exemple 2.5 *Pour être plus concret, nous allons donner un exemple sur la table multidimensionnelle représentée à la figure II.11, que nous utiliserons par la suite comme exemple de référence. Il s'agit du cas classique de ventes de produits dans un magasin situé dans une ville.*

OID	Produit	Magasin	Adresse	TotalVentes
1	Lait	Auchan	Blois	80
2	Lait	Carrefour	Paris	100
3	Beurre	Super U	Blois	30
4	Lait	Auchan	Nantes	120
5	Sucre	Carrefour	Nantes	60
6	Fromage	Auchan	Paris	30
7	Lait	Carrefour	Nantes	200
8	Beurre	Auchan	Blois	20
9	Fromage	Carrefour	Paris	80
10	Lait	Carrefour	Blois	220

FIG. II.11 – Table multidimensionnelle

Les attributs Produit, Magasin et Adresse sont les dimensions et la mesure représentée par l'attribut TotalVentes indique le total des ventes d'un produit dans un magasin.

2.5.2 La forme des motifs

Les motifs obtenus sont de la même forme que ceux dans le cas de la table classique. L'espace de recherche des motifs est exactement le même. Cependant, le sens de l'extraction dépendra de la mesure et du critère de sélection. Dans le cas classique, le motif $(Produit = Lait, Magasin = Auchan)$ était intéressant par sa fréquence d'apparition, alors qu'ici il pourrait l'être par rapport à un critère lié à la mesure. Dans notre cas, la mesure est la somme des ventes. Un motif peut par exemple être intéressant si la somme des ventes de produits laitiers à Auchan (i.e. correspondant à $((Produit = Lait), (Magasin = Auchan))$ est supérieure à un certain seuil.

2.5.3 Les critères de sélection

L'introduction de la mesure permet de prendre en compte d'autres critéres de sélection, ce qui généralise le problème de l'extraction des motifs.
La fonction f, qui fait correspondre à un motif X l'ensemble des objets qui le contiennent, est définie comme suit :

$$\begin{array}{rcl} f & : & 2^{\mathbb{P}} \rightarrow 2^{\mathbb{O}} \\ & & X \rightarrow f(X) = \{o \in \mathbb{O} \mid \forall x \in X, (o,x) \in \mathbb{R}\} \end{array}$$

La mesure m est une fonction qui fait correspondre à un objet un nombre réel :

$$\begin{array}{rcl} \mathrm{m} & : & \mathbb{O} \rightarrow \mathcal{R} \\ & & o \rightarrow m(o) \end{array}$$

Soit agg une fonction qui applique une aggrégation sur un ensemble de mesures :

$$\begin{array}{rcl} \mathrm{agg} & : & 2^{\mathcal{R}} \rightarrow \mathcal{R} \\ & & Y \rightarrow agg(Y) \end{array}$$

Un critère de sélection se définit comme suit :

Définition 2.23 - Critère de sélection.
Etant donné une mesure m, un opérateur $op \in \{ \leq, \geq \}$, une fonction d'aggrégation agg et un

seuil α, on note $\mathbf{agg(m, X)}$ **op** α *le critère de sélection q défini pour tout $X \subseteq 2^{\mathbb{P}}$ par :*

$$q(X) = true \text{ si } agg(\ \{\ m(o) \mid o \in f(X)\ \}\ )\ op\ \alpha.$$

Lorsque la fonction d'aggrégation est *count*, l'application du critère revient, lorsque l'opérateur *op* est $\geq$, à trouver l'ensemble des motifs dont le nombre d'occurrences dépasse le seuil α. On obtient ainsi le même critère de sélection que dans le cas transactionnel, $sup(X) \geq \alpha$. Autrement dit, le cas multidimensionnel est un cas plus général que le cas transactionnel.
Si l'on considère que la mesure représente des prix de vente, l'application du critère de sélection avec la fonction d'aggrégation *sum* et l'opérateur $\geq$ revient à trouver l'ensemble des motifs dont la somme des prix de ventes dépasse le seuil α.

Exemple 2.6 *Considérons la table de la figure II.11.*
Si on s'intéresse par exemple aux produits laitiers vendus à Auchan, on considère le motif $X = (Produit = Lait, Magasin = Auchan)$, et l'on a : $f(X) = \{1, 4\}$
Si la fonction d'aggrégation est la fonction count, on obtient

$$\begin{aligned} agg(m,X) &= count(\ m, (Produit = Lait, Magasin = Auchan)\ ) \\ &= count(\ \{m(o) \mid o \in f((Produit = Lait, Magasin = Auchan))\}\ ) \\ &= count(\ \{m(o) \mid o \in \{1,4\}\}\ ) \\ &= count(\ \{m(1), m(4)\}\ ) \\ &= 2 \end{aligned}$$

Il faut signaler que l'on considère des ensembles de mesures avec répétition, ce qui fait que $count(\ \{m(1), m(4)\}\ )$ est égal à 2, même si $m(1) = m(4)$.
Si la fonction d'aggrégation est la fonction sum, on obtient

$$\begin{aligned} agg(m,X) &= sum(\ m, (Produit = Lait, Magasin = Auchan)\ ) \\ &= sum(\ \{m(o) \mid o \in f(\ (Produit = Lait, Magasin = Auchan)\ )\}\ ) \\ &= sum(\ \{m(o) \mid o \in \{1,4\}\}\ ) \\ &= sum(\ \{m(1), m(4)\}\ ) \\ &= sum(\ \{80, 120\}\ ) \\ &= 200 \end{aligned}$$

On voit ainsi qu'on obtient plus de critères que dans le cas transactionnel.
Le parcours du treillis pour le calcul des motifs intéressants peut se faire du bas vers le haut ou du haut vers le bas du treillis selon la monotonie ou l'anti-monotonie du critère de sélection. Les définitions suivantes permettent d'étudier cette propriété du critère de sélection.

Définition 2.24 - fonction monotone croissante. *La fonction d'aggrégation agg est une fonction monotone croissante si pour tous $Y, Y' \in 2^{\mathcal{R}}$, $Y \subseteq Y' \Rightarrow agg(Y) \leq agg(Y')$.*

Définition 2.25 - fonction monotone décroissante. *la fonction d'aggrégation agg est une fonction monotone décroissante si pour tous $Y, Y' \in 2^{\mathcal{R}}$, $Y \subseteq Y' \Rightarrow agg(Y) \geq agg(Y')$.*

Proposition 2.5 *Le critère de sélection $agg(m, X) \geq \alpha$ est monotone si agg est une fonction décroissante.*
Le critère de sélection $agg(m, X) \geq \alpha$ est anti-monotone si agg est une fonction croissante.
Le critère de sélection $agg(m, X) \leq \alpha$ est monotone si agg est une fonction croissante.
Le critère de sélection $agg(m, X) \leq \alpha$ est anti-monotone si agg est une fonction décroissante.

Preuve : Nous allons faire la preuve pour le critère de sélection $agg(m, X) \geq \alpha$. Elle sera similaire pour les autres critères.
Soient X, X' deux motifs. Si $X \subseteq X'$, alors $f(X) \supseteq f(X')$ et $\{\ m(o) \mid o \in f(X)\} \supseteq \{\ m(o) \mid o \in f(X')\}$. Si la fonction agg est croissante, on a donc $agg(m, X) \geq agg(m, X')$. Ainsi, si $agg(m, X') \geq \alpha$, alors $agg(m, X) \geq \alpha$, d'où la monotonie du critère.
Par contre, si la fonction agg est décroissante, alors $agg(m, X) \leq agg(m, X')$. Ainsi, si $agg(m, X) \geq \alpha$, alors $agg(m, X') \geq \alpha$, d'où l'anti-monotonie du critère.

2.5.4 Les algorithmes

Les algorithmes s'intéressent généralement à l'extraction des motifs intéressants qui sont des ensembles de propriétés. C'est ce qui sera l'objet de notre étude. L'extraction des motifs revient ainsi à construire la théorie $\mathcal{T}h(2^{\mathbb{P}}, \mathbb{D}, agg(m, X)\ op\ \alpha) = \{X \subseteq \mathbb{P} \mid agg(m, X)\ op\ \alpha\}$. Etant donné que l'on peut avoir plusieurs fonctions et plusieurs mesures, un motif intéressant n'aura vraiment de sens qu'avec sa mesure associée, à savoir $agg(m, X)$. On définit ainsi :

Définition 2.26 - Iceberg Cube.
On appelle Iceberg Cube, noté $\mathcal{I}nt_{\alpha,agg}(< \mathbb{O}, \mathbb{P}, \mathbb{R}, m >)$ *l'ensemble des motifs intéressants munis de leurs supports :*

$$\mathcal{I}nt_{\alpha,agg}(< \mathbb{O}, \mathbb{P}, \mathbb{R}, m >) = \{(X, agg(m, X)) \mid X \in \mathcal{T}h(2^{\mathbb{P}}, \mathbb{D}, agg(m, X)\ op\ \alpha)\}.$$

Il existe deux approches principales pour la construction des Iceberg-Cubes : l'approche descendante et celle ascendante.
Lorsque le critère de qualité est monotone, on utilise une approche descendante [80]. En effet, la monotonie du critère de qualité implique que si un motif est intéressant, tous les motifs qui sont plus spécifiques que lui le sont aussi. Autrement dit, dès qu'un motif est non intéressant, aucun motif plus général que lui ne le sera. L'approche descendante, qui parcourt le treillis en partant des motifs les plus spécifiques, utilise cette propriété pour faire des coupures. Lorsque, par contre, le critère de qualité est anti-monotone, il vaut mieux utiliser une approche ascendante.
Les algorithmes proposés pour la construction d'Iceberg Cubes [10, 57] se distinguent par leur mode de parcours du treillis des parties de l'ensemble des dimensions. Contrairement à Apriori, le parcours du treillis pour le calcul de cette théorie va se faire en profondeur, et non en largeur, à cause du nombre important de candidats pouvant être généré à chaque niveau.
On distingue deux modes parcours en profondeur : le parcours selon l'ordre lexicographique et le parcours selon l'ordre lectique. Les deux algorithmes récents que nous allons présenter BUC et APIC [57, 10], qui utilisent une approche ascendante, effectuent respectivement un parcours lexicographique et lectique du treillis pour la construction de l'Iceberg-Cube.
On suppose que l'ensemble des dimensions est ordonné. Ainsi, on peut comparer deux motifs $(A_i = a_i)$ et $(A_j = a_j)$ comme suit :

$$\begin{gathered}(A_i = a_i) < (A_j = a_j) \\ Si\ \ A_i < A_j \\ Ou\ \ a_i < a_j\ (lorsque\ A_i = A_j).\end{gathered}$$

Si on considére les motifs écrits selon l'ordre croissant de leurs éléments, deux motifs X et X' sont alors comparables selon l'ordre lexicographique.

Exemple 2.7 *Considérons l'ensemble ordonné suivant : R = {A,B,C,D}. L'ensemble des combinaisons de* 2^R *suivant l'ordre lexicographique est présenté à la figure II.12. Les numéros indiquent l'ordre de parcours.*

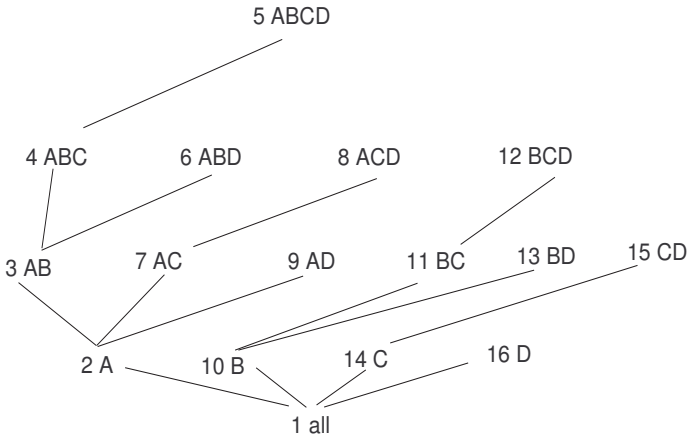


FIG. II.12 – BUC processing tree (Ordre lexicographique).

Algorithme BUC

L'algorithme BUC (Bottom Up Computation) a été proposé par Beyer et Ramakrishnan [10]. Il est l'opposé de tous les algorithmes qui avaient été proposées avant lui et qui relevaient de l'approche descendante [80, 81].
Comme indiqué, l'algorithme effectue un parcours du bas vers le haut du treillis des parties de $\mathbb{A}$ dans l'ordre lexicographique (voir figure II.12). Les numéros indiquent l'ordre de parcours. L'algorithme commence par examiner l'ensemble des motifs de la forme $\{A = a\}$, où $a \in dom(A)$. Supposons que les motifs $(A = a_1)$ et $(A = a_3)$ soient intéressants. Il s'intéresse alors aux motifs de la forme $(A = a_1) \cup \{B = b\}$, où $b \in dom(B)$. Si $(A = a_1, B = b_1)$ est intéressant, alors les motifs candidats de l'étape suivante seront ceux de la forme $(A = a_1, B = b_1) \cup \{C = c\}$, où $c \in dom(C)$ et ainsi de suite. Après avoir trouvé les motifs intéressants, spécialisations de $A = a_1$, l'algorithme passe aux spécialisations de $A = a_3$.
L'algorithme de calcul des motifs qui montre l'ordre de parcours du treillis est présenté à la figure II.13. La fonction $F(\mathbb{A}, \mathbb{O}, q)$ montre que l'algorithme part des motifs de la forme $\{A = a \mid a \in dom(A)\}$ et calcule récursivement l'ensemble des motifs intéressants noté *Sol* en appelant la fonction Fbis. La fonction *Fbis* décrit l'ordre de parcours du treillis comme indiqué plus haut.

Exemple 2.8 *Supposons que l'on cherche à construire, à partir de la table II.11 l'Iceberg-Cube suivant :*
$IC = \{ (X, agg(m, X)) \mid X \in \mathcal{T}h(2^{\mathcal{P}}, \mathbb{D}, agg(m, X) \geq \alpha)\}$*, où la fonction d'aggrégation est la fonction sum et* $\alpha = 200$*. BUC commence par calculer l'ensemble des motifs de la forme* $\{Produit = p \mid p \in dom(Produit)\}$*. Le motif* $(Produit = Lait)$ *étant intéressant (puisque la somme des ventes correspondante est supérieure à 200), il passe au motif* $(Produit = Lait, Magasin = Auchan)$*. Ce motif est aussi intéressant, mais il n'y a pas de motifs intéressants plus spécifiques que lui. L'algorithme passe alors au motif* $(Produit = Lait, Magasin = Carrefour)$*. Il continue ainsi en suivant ce parcours lexicographique jusqu'à la construction totale de l'Iceberg-Cube qui est montré à la figure* $II.14$*.*

Il faut souligner que l'algorithme *BUC* ne fait pas toutes les coupures possibles lorsqu'il passe d'un nœud à un autre. Supposons qu'après avoir calculé l'ensemble des motifs de la forme $A = a$, il trouve que $A = a_1$ est intéressant. L'algorithme passe alors au calcul des motifs plus spécifiques

Algorithme BUC :
– **Entrée :** l'ensemble des attributs $\mathbb{A} = \{A_1, A_2, \ldots, A_n\}$ l'ensemble des données $\mathbb{D} =< \mathbb{O}, \mathbb{P}, \mathbb{R}, m >$ et le critère de qualité $q = agg(m, X)$ op α, où $op \in \{\leq, \geq\}$;
– **Sortie :** l'Iceberg-Cube.

1. $Sol = \emptyset$
2. Pour i=1 à n
3. $\quad Cand = \{\ \{A_i = a_j\} \mid a_j \in dom(A_i)\ \}$
4. $\quad Sol = Sol \cup Fbis(Cand, \mathbb{O}, q)$
5. $I = \{(X, agg(m, X)) \mid X \in Sol\}$
6. **Retourner** I

Fonction Fbis(Cand, O, q)

1. D $= < O, \mathbb{P}, R, m >$, où $R = \mathbb{R}_{|O}$ (restriction de $\mathbb{R}$ à O)
2. $S = \mathcal{T}h(Cand, D, q)$
3. Pour tout $\varphi = \{A_{i_1} = a_{i_1}, \ldots, A_{i_p} = a_{i_p}\}$ appartenant à $\mathcal{T}h(Cand, D, q)$ tel que $i_1 < i_2 < \ldots < i_p$
4. $\quad O' = \{o \in O \mid (\forall x \in \varphi)(o\ \mathbb{R}\ x)\}$
5. $\quad$ Pour tout $j = i_p + 1$ à n
6. $\quad\quad Cand = \{\varphi \cup \{A_j = a_j\} \mid a_j \in dom(A_j)\}$
7. $\quad\quad S = S \cup Fbis(Cand, O', q)$
8. **Retourner S**

FIG. II.13 – Algorithme BUC

Motif	**TotalVentes**
$Produit = Lait$	720
$Produit = Lait, Magasin = Auchan$	200
$Produit = Lait, Magasin = Carrefour$	300
$Produit = Lait, Adresse = Blois$	300
$Produit = Lait, Adresse = Nantes$	320
$Magasin = Auchan$	250
$Magasin = Carrefour$	660
$Magasin = Carrefour, Adresse = Nantes$	260
$Magasin = Carrefour, Adresse = Blois$	220
$Adresse = Blois$	350
$Adresse = Paris$	210
$Adresse = Nantes$	380

FIG. II.14 – Iceberg-Cube

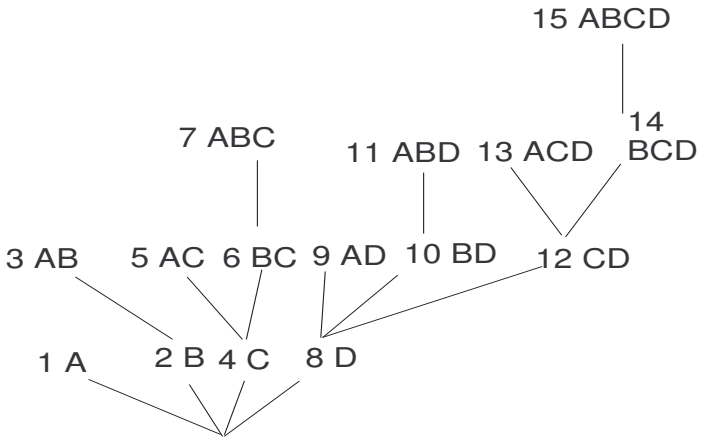


FIG. II.15 – Arbre lectique

que $A = a_1$, et les évalue, même si un de leurs sous-ensembles n'est pas fréquent. Par exemple $(Produit = Lait, Magasin = SuperU)$ sera évalué alors que $Magasin = SuperU$ n'est pas fréquent. Ceci est dû à l'ordre de parcours du treillis.

Algorithme APIC

Une autre approche pour le calcul des Iceberg-Cubes a été proposée par l'équipe de Lotfi Lakhal [57]. Cette approche se différencie de BUC par un autre ordre de parcours qui est comme indiqué plus haut, l'ordre lectique.

Définition 2.27 - Ordre lectique. *Soit $E = \{1, 2, \ldots, n\}$ ensemble fini et ordonné. Un sous-ensemble $A \subseteq E$ est lectiquement plus petit qu'un sous-ensemble $B \neq A$ si le plus petit élément qui distingue A et B appartient à B :*

$$A \preceq_l B \Leftrightarrow \exists\ (i \in B \backslash A) \mid A \cap \{1, 2, \ldots, i-1\} = B \cap \{1, 2, \ldots, i-1\}$$

Exemple 2.9 *Considérons l'ensemble ordonné suivant : R = {A,B,C,D}. L'ensemble des combinaisons de 2^R suivant l'ordre lectique est présenté à la figure II.15. Les numéros indiquent l'ordre de parcours selon l'ordre lectique.*

Pour le parcours du treillis, les auteurs définissent deux opérateurs. Etant donné un motif X :
- l'opérateur $Next(X)$ permet de trouver le motif suivant de X selon l'ordre lectique.
- l'opérateur $Skip(X)$ trouve le motif suivant de X qui n'est pas un sur-ensemble de X.

L'opérateur $Skip(X)$ est utilisé pour l'élagage. Ainsi Si X n'est pas intéressant $Skip(X)$ permet d'élaguer X et tous ses sur-ensembles.

Exemple 2.10 *Reprenons l'exemple de la figure II.15.*
Next(AB) = C ; Next(BC) = ABC.
Skip(C) = D ; Skip(BD) = CD.

La proposition suivante est une propriété fondamentale de l'ordre lectique.

Proposition 2.6 *Soit E un ensemble ordonné.*

$$(\forall X \subseteq E)(\forall Y \subseteq E)(Y \subset X \Rightarrow Y \preceq_l X).$$

Cet ordre résout le problème des coupures lié à l'ordre de parcours de BUC. En effet, si les motifs sont examinés selon l'ordre lectique, un motif X (d'après la proposition 2.6) ne sera examiné qu'après avoir examiné tous ses sous-ensembles Y. On pourra donc éviter d'évaluer son intérêt s'il a déjà été prouvé qu'un de ses sous-ensembles n'était pas intéressant.
Pour le calcul de l'Iceberg-Cube, les auteurs proposent un algorithme (APIC) qui parcourt l'arbre du bas vers le haut en suivant cet ordre lectique. D'abord les motifs de la forme $\{A = a\}$, où $a \in dom(A)$ sont examinés. Supposons qu'il trouve intéressants les motifs $(A = a_1)$ et $(A = a_3)$. L'algorithme passe alors aux motifs de la forme $\{B = b\}$, où $b \in dom(B)$. Supposons que seul $(B = b_2)$ soit trouvé intéressant. Les candidats sont maintenant les motifs $(A = a_1, B = b_2)$ et $(A = a_3, B = b_2)$. Il s'agit des candidats dont tous les sous-ensembles sont intéressants. L'algorithme qui montre l'ordre de parcours de APIC est présenté à la figure II.16. A chaque nœud de l'arbre lectique, l'algorithme calcule l'ensemble des motifs intéressants correspondants. Si cet ensemble n'est pas vide, l'algorithme passe par l'opérateur $Next(X)$ à l'ensemble des motifs suivant l'ordre lectique. Sinon, il passe directement par l'opérateur $Skip(X)$ aux motifs suivants qui ne sont des sur-ensembles des motifs considérés. On suppose ici que $Next(\mathbb{A}) = NULL$ et $Skip(\{A_i, A_{i+1}, \ldots, A_n\}) = NULL$ pour $i = 1 \ldots n$.

Exemple 2.11 *Reprenons le calcul de l'Iceberg-Cube de l'exemple 2.8 construit par BUC, mais cette fois-ci avec APIC. APIC commence par calculer l'ensemble des motifs de la forme* $\{(Produit = p) \mid p \in dom(Produit)\}$. *Seul le motif* $(Produit = Lait)$ *est intéressant (puisque la somme des ventes correspondante est supérieure à 200). Il passe alors aux motifs de la forme* $\{(Magasin = m) \mid m \in dom(Magasin)\}$. *Les motifs* $(Magasin = Auchan)$ *et* $(Magasin = Carrefour)$ *sont intéressants. Les prochains candidats seront alors les motifs* $(Produit = Lait, Magasin = Auchan)$ *et* $(Produit = Lait, Magasin = Carrefour)$. *Ces motifs sont intéressants. Il passe alors selon l'ordre lectique aux motifs de la forme* $\{(Adresse = a) \mid a \in dom(Adresse)\}$ *et ainsi de suite jusqu'à la construction totale de l'Iceberg-Cube.*

D'autre part, l'originalité de cette approche est une représentation inverse des données et des motifs, qui s'inspire du modèle partitionnel ([22],[83]). L'idée de cette représentation est d'associer à chaque motif l'ensemble des objets qui le contiennent. Une telle représentation rend facile la génération et l'évaluation des motifs de plus grandes tailles à partir de ceux de tailles plus petites et permet de remplacer des méthodes de tri complexe ou basées sur le hachage par de simples intersections d'ensembles.
L'exemple suivant montre que l'on peut évaluer le motif $A = a_1, B = b_2$ par l'intersection des ensembles des objets qui contiennent les motifs $A = a_1$ et $B = b_2$.

Exemple 2.12 *On associe à tout motif* X *l'ensemble* $f(X)$ *des objets qui le contiennent. Soient les motifs* $A = a_1$ *et* $B = b_2$.

$$f(A = a_1) = \{\ 1, 2\ \};$$
$$f(B = b_2) = \{\ 1, 2, 4, 5, 10\ \};$$
$$On\ note\ que\ f(A = a_1, B = b_2) = f(A = a_1) \cap f(B = b_2) = \{\ 1, 2\ \}.$$

Discussion

La première question que l'on pourrait se poser est la suivante : pourquoi ne pas utiliser l'algorithme Apriori pour le calcul des Iceberg-Cube ? D'autant plus que le critère d'élagage est le même que celui d'Apriori, d'une part, et d'autre part, qu'un candidat dans Apriori n'est généré que si tous ses sous-ensembles sont intéressants. La différence fondamentale entre Apriori et les algorithmes BUC et $APIC$ est que le premier parcourt le treillis par niveau, alors que les derniers effectuent un parcours en profondeur. Le problème qui se pose avec *Apriori* est le suivant :

Algorithme APIC :
- **Entrée :** l'ensemble des attributs $\mathbb{A} = \{A_1, A_2, \ldots, A_n\}$, l'ensemble des données $\mathbb{D} =< \mathbb{O}, \mathbb{P}, \mathbb{R}, m >$ et le critère de qualité q ;
- **Sortie :** l'Iceberg-Cube.

1. $Sol = \{\emptyset\}$
2. $X = \{A_1\}$
3. **Faire** {
4. $\quad Cand = \{\{A_{i_1} = a_1, ..., A_{i_p} = a_p\} \mid X = \{A_{i_1}, ..., A_{i_p}\}, (\forall i)(a_i \in dom(A_{i_p}))\}$
5. $\quad Cand = \{\varphi \in Cand \mid (\forall (A_i = a_i) \in \varphi)(\varphi \backslash \{A_i = a_i\} \in Sol)\}$
6. $\quad$ Calcul de $S = \mathcal{T}h(Cand, \mathbb{D}, q)$
7. $\quad Sol = Sol \cup S$
8. $\quad$ **Si** $S = \emptyset$ **alors** $X = Skip(X)$
9. $\quad\quad$ **sinon** $X = Next(X)$
10. } **Tant que** $(X \neq NULL)$
11. $I = \{(X, agg(m, X)) \mid X \in Sol\}$
12. **Retourner** I

FIG. II.16 – Algorithme APIC

le nombre de candidats durant les premières passes est très important du fait de la génération par niveau, ce qui fait qu'ils ne tiennent pas tous en mémoire. Pour BUC et APIC, ce nombre est beaucoup moins important du fait du parcours en profondeur, parce que seuls les motifs du nœud traité sont chargés en mémoire.

2.6 Extraction de motifs intéressants stockés dans plusieurs tables

Les approches classiques de découverte de règles d'association ont de sérieuses limites qui les rendent obsolètes dans un cadre relationnel :

- Limites du modèle de représentation de données : Les données sont stockées dans une table. Ceci implique la découverte de relations entre attributs d'une seule table.
- Pauvreté de la puissance d'expression des motifs : Les motifs découverts sont exprimés dans des langages attribut-valeur qui ont la puissance d'expression de la logique propositionnelle [32]. Ces langages ne permettent pas de représenter des structures complexes et des relations entre elles.

C'est ainsi que des approches multi-relationnelles, qui portent le nom de MRDM (Multi-Relational Data Mining) ont été proposées [33, 32, 27, 4]. MRDM est une forme de Data Mining où les données sont stockées dans plusieurs tables de bases de données relationnelles.
Une première approche de MRDM est d'effectuer l'extraction sur une table qui est le résultat d'une requête sur une base de données relationnelle. Ceci va permettre la découverte de relations entre attributs de tables différentes. Les approches que nous allons présenter ici [27, 34] et qui rentrent dans le cadre de la Programmation Logique Inductive (PLI), utilisent la logique du premier ordre pour représenter les données et les motifs.

2.6.1 Les données

Nous allons d'abord étudier quelques définitions de base de la logique du premier ordre pour comprendre comment les données sont représentées. Une présentation plus exhaustive de ce langage peut être trouvée dans [59].
Le langage logique du premier ordre est défini par : un ensemble de symboles de variables $v_1, \ldots, v_n$, un ensemble de symboles de constantes $c_1, \ldots, c_m$, un ensemble de symboles de fonctions $f_1, \ldots, f_k$, un ensemble de symboles de prédicats $p_1, \ldots, p_j$, des connecteurs logiques $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow, etc \ldots$, des quantificateurs $\forall, \exists$, des symboles de ponctuation.
A chaque symbole de prédicat ou de fonction est associée une arité qui désigne le nombre d'arguments auxquels ce symbole doit s'appliquer.
Toute constante est un **terme**, toute variable est un terme. Si f est un symbole de fonction d'arité n, et si $t_1, \ldots, t_n$ sont des termes, alors $f(t_1, \ldots, t_n)$ est un terme.
Il faut cependant signaler qu'en général, les fonctions ne sont pas considérées dans le cadre de MRDM.
Si p est un symbole de prédicat d'arité n, et si $t_1, \ldots, t_n$ sont des termes, alors $p(t_1 \ldots, t_n)$ est un **atome**. Un **fait** est un atome ne comportant aucune variable.

Définition 2.28 *Un ensemble de données est un ensemble de faits.*

Nous parlerons ainsi d'ensemble de données ou d'ensembles de faits.

Définition 2.29 - Domaine.
Soit p un prédicat d'arité m et $1 \leq k \leq m$. On note $dom(k, p)$ l'ensemble des valeurs prises par le k-ième terme d'un atome sur p. Ces ensembles de valeurs sont appelées domaines.

Un ensemble de faits peut être stocké dans une base de données relationnelle.

Exemple 2.13 *Soit $Pred = \{Cust, Sale, Prod\}$ un ensemble de prédicats d'arités 3, 2 et 2, définis comme suit :*
- *$Cust(Cid, Cprof, Caddr)$: le client Cid de profession $Cprof$ habite à l'adresse $Caddr$.*
- *$Sale(Cid, Pid)$: le client Cid a acheté le produit Pid.*
- *$Prod(Pid, Ptype)$: le produit Pid est de type $Ptype$.*

On suppose que les domaines sont définis comme suit :
- *dom(1,Cust) = dom(1,Sale) = $\{Abdou, Pierre, Jean\}$;*
- *dom(2,Cust) = $\{Enseignant, Avocat\}$;*
- *dom(3,Cust) = $\{Blois, Orléans\}$;*
- *dom(2,Sale) = dom(1,Prod) = $\{1, 2, 3, 4\}$; identifiants des produits.*
- *dom(2,Prod) = $\{Thé, Laitier, Alcool\}$.*

Voici un exemple d'ensemble de données :

$$S = \{\ Prod(1, Laitier), Prod(2, Laitier), Prod(3, Alcool), Prod(4, thé), Sale(Abdou, 1), Sale(Abdou, 2), Sale(Pierre, 1), Sale(Pierre, 3), Sale(Jean, 3), Sale(Jean, 4), Cust(Abdou, Enseignant, Blois), Cust(Pierre, Enseignant, Orléans), Cust(Jean, Avocat, Blois)\ \}.$$

Cet exemple sera utilisé par la suite comme exemple courant.

2.6.2 La forme des motifs

Il existe deux types de motifs qui sont respectivement les équivalents des itemsets et des règles dans le contexte relationnel : les requêtes conjonctives d'une certaine forme que nous allons préciser et les implications entre elles. Nous allons d'abord commencer par la présentation de quelques définitions de base.

Définition 2.30 - Requête conjonctive.
Une requête conjonctive Q est une formule de la forme

$$(\exists Y)(L1 \wedge L2 \wedge \ldots \wedge Lp)$$

où
- $L1 \wedge L2 \wedge \ldots \wedge Lp$ *est une conjonction d'atomes,*
- Y *est un ensemble de variables apparaissant dans* $L1 \wedge L2 \wedge \ldots \wedge Lp$.
- *Les variables de* $L1 \wedge L2 \wedge \ldots \wedge Lp$ *n'appartenant pas à* Y *sont appelées* **variables libres**.

Pour définir la réponse à une requête, nous avons besoin de la définition d'une substitution.

Définition 2.31 - Substitution.
Une substitution est une application de l'ensemble des variables dans l'ensemble des variables et des constantes.

Par exemple, $\theta = \{x/Avocat, y/z\}$ est la substitution définie par $\theta(x) = Avocat, \theta(z) = w$.
On note $p\theta$ le résultat de l'application de la substitution θ à l'ensemble des termes de l'atome $p(t_1, \ldots, t_n)$.
Une substitution θ est dite élémentaire si :
- Il existe une variable unique x_0 telle que $\theta(x_0)$ soit une constante et,
- Pour toute variable y différente de x_0, on a $\theta(y) = y$.

Définition 2.32 - Réponse à une requête conjonctive.
Soit Q une requête conjonctive. Pour tout ensemble de faits $\mathcal{S}$, la réponse à Q dans $\mathcal{S}$, notée $Q(\mathcal{S})$ est l'ensemble des n-uplets $K\theta$ où :
- K *est l'ensemble des variables libres de* Q *et,*
- θ *est une substitution telle que pour tout atome* $p(t_1, \ldots, t_n)$ *de* Q, $p\theta$ *appartient à* $\mathcal{S}$.

Un élément important lorsque l'on parle d'extraction de motifs est le sujet de l'extraction, c'est à dire ce que l'on compte. Dans le contexte d'Apriori, les motifs sont de la forme : *50% des transactions contiennent du pain et du beurre* ou *70% des transactions qui contiennent du lait contiennent du sucre.* On remarque qu'il ne peut pas y avoir d'équivoque puisque l'on s'intéresse toujours aux transactions. Par contre, dans une base de données, il y a plusieurs tables, et on peut compter par rapport à n'importe quel attribut. Par exemple, dans notre cas, on pourrait aussi bien s'intéresser aux clients qu'aux produits ou encore aux professions. C'est dire qu'il y a besoin de spécifier le sujet de l'extraction dans la requête. Cela se fait par une **requête de référence**. Une requête de référence est une requête conjonctive qui précise le sujet de l'extraction. Lorsque cette requête est composée d'un seul atome, on l'appelle aussi **atome de référence**. Par exemple, si on s'intéresse aux clients, la requête de référence est la suivante :
$(\exists x_2, x_3)(Cust(x_1, x_2, x_3))$.
La recherche des motifs se présente ainsi comme suit : étant donné une requête de référence R de variables libres K, on s'intéresse à des requêtes conjonctives de la forme $R \wedge Q$ où Q est une requête de même ensemble de variables libres K que R.

Exemple 2.14 *Soit la requête de référence suivante :*

$$R \ : \ (\exists y_1)(Sale(x_1, y_1))$$

La requête $Q \ : \ (\exists y_1, x_2)(Cust(x_1, x_2, Blois) \wedge Sale(x_1, y_1) \wedge Prod(y_1, Laitier))$ *permet de s'intéresser aux consommateurs blésois qui achètent des produits laitiers.*

Définition 2.33 - Implications entre requêtes
Soit R une requête de référence de variables libres K, Q_B et Q_H deux requêtes conjonctives de même ensemble de variables libres K. On s'intéresse alors aux implications logiques de la forme :

$$(\forall K)(R \wedge Q_B \Rightarrow R \wedge Q_H)$$

Les implications sont les équivalents des règles d'associations.

Exemple 2.15 *Soit la requête de référence suivante :*

$$R \ : \ (\exists y_1)(Sale(x_1, y_1))$$

La requête $Q1 \ : \ (\exists x_3, y_1)(Cust(x_1, Enseignant, x_3) \wedge Sale(x_1, y_1))$ définit l'ensemble des consommateurs enseignants.
La requête $Q2 \ : \ (\exists y_1)(Sale(x_1, y_1) \wedge Prod(y_1, Alcool))$ définit l'ensemble des consommateurs d'alcool.
L'implication $(\forall x_1)(Q1 \Rightarrow Q2)$ indique que tous les enseignants sont des consommateurs d'alcool.

Il faut signaler que dans [27], les mêmes motifs sont définis dans un formalisme Datalog.
Nous allons maintenant présenter les mesures d'intérêt des motifs. Les deux mesures les plus utilisées sont le support et la confiance.

Définition 2.34 - Support d'une requête.
Etant donnée une requête de référence R, soit Q une requête conjonctive de même ensemble de variables libres que R. Pour tout ensemble de faits S, le support de Q dans S relativement à R est défini par :

$$Sup(Q/R, S) = \mid [R \wedge Q](S)\mid \ / \ \mid R(S)\mid.$$

Définition 2.35 - Confiance d'une implication.
Etant donnée une requête de référence R, soit Q_B et Q_H deux requêtes conjonctives de même ensemble de variables libres que R.
Soit l'implication $I \ : \ (\forall K)(R \wedge Q_B \Rightarrow R \wedge Q_H)$. Pour tout ensemble de faits S, la confiance de I dans S relativement à R est définie par :

$$\begin{array}{rcl} Conf(I\ /R,S) & = & \mid [R \wedge Q_B \wedge Q_H](S) \mid \ / \ \mid [R \wedge Q_B](S) \mid \\ & = & Sup(Q_B \wedge Q_H \ / \ R, S) \ / \ Sup(Q_B/R, S) \end{array}$$

Exemple 2.16 *Considèrons notre exemple de référence (exemple 2.13) où l'ensemble de données S est défini. Soit la requête de référence suivante :*

$$R1 \ : \ (\exists y_1)(Sale(x_1, y_1))$$

La requête $Q1 \ : \ (\exists x_2, y_1)(Cust(x_1, x_2, Blois) \wedge Sale(x_1, y_1))$ défiit l'ensemble des consommateurs blésois.
La requête $Q2 \ : \ (\exists y_1)(Sale(x_1, y_1) \wedge Prod(y_1, alcool))$ définit l'ensemble des consommateurs d'alcool. Soit $I1 \ : \ (\forall x_1)(Q1 \Rightarrow Q2)$.

$$Sup(Q1/R, S) = 2/3;$$
$$Conf(I1/R, S) = (1/3)/(2/3) = 1/2.$$

Dans ce contexte, où nous avons des requêtes comme motifs à la place des itemsets, la relation de spécialisation est l'inclusion de requêtes.

Définition 2.36 - Inclusion de requêtes.
Soit P et Q deux requêtes conjonctives de même ensemble de variables libres. On dit que P est incluse dans Q et on note $P \sqsubseteq Q$, si $P(S) \subseteq Q(S)$ pour tout ensemble de données S. De plus, si $P \sqsubseteq Q$, alors l'implication $(\forall K)(P \Rightarrow Q)$ est vraie.

Voyons maintenant quelles sont les principales propriétés du support dont nous aurons besoin pour l'extraction des motifs.

Proposition 2.7 *Etant donnée une requête de référence R, soit Q une requête conjonctive de même ensemble de variables libres que R. Pour tout ensemble de faits S :*
- $0 \leq Sup(Q/R, S) \leq 1$.
- $Sup(Q/R, S) = 1$ *ssi S est un modèle de la formule* $(\forall K)(R \wedge Q)$.

Proposition 2.8 *Etant donnée une requête de référence R de variables libres K, soit Q une requête conjonctive de même ensemble de variables libres que R, et β une substitution des variables quantifiées existentiellement de Q. Alors :*
- $(\forall K)(Q\beta \Rightarrow Q)$ *est valide.*
- *Donc* $Sup(Q\beta/R, S) \leq Sup(Q/R, S)$.

Proposition 2.9 *Soient deux requêtes $Q1$ et $Q2$ conjonctives de mêmes variables de référence K. Alors :*
- $(\forall K)(Q1 \wedge Q2 \Rightarrow Q1)$ *est valide, donc* $Sup(Q1 \wedge Q2/R, S) \leq Sup(Q1/R, S)$.
- $(\forall K)(Q1 \wedge Q2 \Rightarrow Q2)$ *est valide, donc* $Sup(Q1 \wedge Q2/R, S) \leq Sup(Q2/R, S)$.

Les autres mesures (Couverture, Intérêt, Conviction) [12, 17], que nous avons présenté dans le cas transactionnel, peuvent aussi être définies dans le formalisme logique. Nous rappelerons leurs définitions dans le cas transactionnel et donnerons les définitions dans le cas logique toujours en nous basant sur le schéma de la figure II.4.
Dans le cas transactionnel, la couverture est définie par $Couv(X \Rightarrow Y) = P(X)$. Dans le formalisme logique cette définition peut être réecrite commme suit :

Définition 2.37 *Etant donné un ensemble de faits S, soit I une implication de la forme* $(\forall K)(R \wedge Q_B \Rightarrow R \wedge Q_H)$, *la couverture de l'implication notée $Couv(I \ / \ R, S)$ est définie par :*

$$Couv(I/R, S) = Sup(Q_B \ / \ R, S).$$

Dans le cas transactionnel, l'intérêt d'une règle d'association est défini par $Int(X \Rightarrow Y) = P(X, Y) \ / \ (P(X) * P(Y))$. Cela est traduit dans le formalisme logique comme suit :

Définition 2.38 *Etant donné un ensemble de faits S, soit I une implication de la forme* $(\forall K)(R \wedge Q_B \Rightarrow R \wedge Q_H)$, *l'intérêt de l'implication notée $Int(I \ / \ R, S)$ est défini par :*

$$Int(I \ / \ R, S) = Sup(Q_B \wedge Q_H \ / \ R, S) \ / \ (Sup(Q_B \ / \ R, S) * Sup(Q_H \ / \ R, S)).$$

La conviction est définie dans le cas transactionnel par $Conv(X \Rightarrow Y) = P(X) * P(\neg Y) \ / \ P(X, \neg Y)$. Dans le formalisme logique, on obtient la définition suivante :

Définition 2.39 *Etant donné un ensemble de faits S, soit I une implication de la forme* $(\forall K)(R \wedge Q_B \Rightarrow R \wedge Q_H)$, *la conviction de l'implication notée $Conv(I \ / \ R, S)$ est définie par :*

$$\begin{aligned} Conv(I \ / \ R,S) \quad = \quad & (Sup(Q_B \ / \ R, S) * (1 - Sup(Q_H \ / \ R, S)) \ / \\ & (Sup(Q_B \ / \ R, S) - Sup(Q_B \wedge Q_H \ / \ R, S)) \end{aligned}$$

2.6.3 Les critères de sélection

Soient *minsup* et $minconf$ deux seuils respectivement de support et de confiance fixés par l'utilisateur.

Définition 2.40 *Soit un ensemble de données S, une requête de référence R, et Q une requête conjonctive de même ensemble de variables libres que R.*
Q est fréquente ssi $Sup(Q \ / \ R, S) \geq minsup$.

Définition 2.41 *Soit un ensemble de données S, une requête de référence R, Q_B et Q_H deux requêtes conjonctives de même ensemble de variables libres que R. L'implication I : $(\forall K)(R \wedge Q_B \Rightarrow R \wedge Q_H)$ est intéressante si $Sup(Q_B \wedge Q_H \ / \ R, S) \geq minsup$ et $Conf(I \ / \ R, S) \geq minconf$.*

Le problème posé est de rechercher l'ensemble des implications intéressantes dans l'espace de recherche que nous allons définir ci-dessous.

2.6.4 Les algorithmes

Dans le cas transactionnel, l'espace de recherche était tout simplement $2^{\mathbb{I}}$, la collection de tous les sous-ensembles de l'ensemble $\mathbb{I}$ des items. Dans le cas relationnel, les motifs recherchés sont exprimés dans la logique du premier ordre, ce qui implique un espace de recherche de taille plus importante. Cet espace de recherche a d'ailleurs fait l'objet de beaucoup d'études en programmation logique inductive par l'intermédiaire des **biais** [1, 27, 73] ou **contextes d'extraction** dont l'objectif est de restreindre explicitement l'ensemble des motifs recherchés.
Il existe plusieurs sortes de biais, mais nous nous arrêterons principalement sur ceux qui sont liés à notre contexte de travail. Il s'agit des **biais de langages**.
Un biais de langage spécifie dans notre contexte le langage dans lequel les motifs recherchés sont définis. Il permet ainsi de restreindre le langage de description des motifs. Les biais de langage se scindent classiquement en deux groupes [88, 68] :

- **les biais syntaxiques**, qui sont, dans notre contexte, des restrictions sur la forme des motifs. On peut citer, comme exemples de biais syntaxiques la limitation du nombre de littéraux, ou la limitation du nombre de variables existentielles. Le biais QLB [35, 34] que nous allons présenter ci-dessous est un exemple de biais syntaxique.
- **les biais sémantiques**, qui utilisent des informations externes aux motifs. Ces informations prennent généralement la forme d'une *étiquette* pour chacun des arguments des prédicats qui constituent le motif. A l'aide de ces étiquettes, il est possible de définir des conditions sur ces arguments. Il existe deux types d'étiquetage : le premier exploite le typage des prédicats et celui des arguments, mais nous nous intéresserons surtout au typage des arguments. Le second, par la déclaration de modes *in* ou *out*, permet de restreindre le choix des variables dans les arguments des prédicats. Le biais W_{rmode} [27] que nous allons présenter ci-dessous est un exemple de biais sémantique.

Nous allons donner deux exemples de biais de langages ci-dessus mentionnés, notamment un exemple de biais syntaxique et un autre de biais sémantique, pour lesquels nous déterminerons à chaque fois, l'espace et la forme des motifs à rechercher, et l'opérateur de spécialisation utilisé.

Premier exemple de biais

Le premier, appelé *QLB(Query Language Bias)*, a été proposé par [35, 34]. Il est, comme indiqué plus haut un exemple de biais syntaxique que les auteurs appellent un contexte d'extraction :

Définition 2.42 Contexte d'extraction.
Un contexte d'extraction est un triplet $\mathcal{C} =< r, \mathcal{L}_C, \Gamma_C >$*, où :*
- r *est une requête définie par l'utilisateur. Cette requête apparaît dans toutes les requêtes et règles (implications) qui seront considérées : il s'agit de la requête de référence.*
- $\mathcal{L}_C$ *est un ensemble d'atomes à partir duquel les requêtes et règles seront construites.*
- Γ_C *est un ensemble de contraintes élémentaires de la forme "*X_i *cmp* a_i*", ou bien "*X_i *cmp* Y_i*", où* X_i *et* Y_i *sont des variables,* a_i *est une constante et cmp* $\in \{=, \leq, <, >, \geq\}$*.*

Précisons que dans ce contexte, les auteurs considèrent seulement les conjonctions d'atomes qui sont connexes, où la connexité est définie comme suit :

Définition 2.43 - Connexité *Soit* $\mathcal{L}$ *une conjonction de* n *atomes.* $\mathcal{L}$ *sera dit connexe s'il peut être écrit sous la forme* $\mathcal{L} = L_1 \wedge L_2 \wedge \ldots \wedge L_n$ *où* L_i*,* $i = 1 \ldots n$*, sont des atomes tels que : pour tout* k*,* $1 < k \leq n$*, il existe* $i < k$ *telle que* L_k *a au moins une variable en commun avec* L_i*.*

L'ensemble des requêtes définies par un contexte donné est appelé l'espace de recherche.

Définition 2.44 - Espace de recherche.
Etant donné un contexte $\mathcal{C} =< r, \mathcal{L}_C, \Gamma_C >$*, on s'intéresse à l'ensemble noté* $\mathcal{Q}(\mathcal{C})$ *des requêtes* $Q = (\exists Y)(r \wedge \mathcal{L} \wedge \Gamma)$ *telles que :*
- $\mathcal{L} \subseteq \mathcal{L}_C$ *et* $r \wedge \mathcal{L}$ *est connexe.*
- $\Gamma \subseteq \Gamma_C$ *et* $Var(\Gamma) \subseteq Var(\mathcal{L})$*, où* $Var(X)$ *représente l'ensemble des variables de l'ensemble* X*.*
- Y *est l'ensemble des variables quantifiées existentiellement de* $\mathcal{L}$*.*

De telles requêtes sont notées $(\mathcal{L} \wedge \Gamma)_r$*.*

On note $\mathcal{R}(\mathcal{C})$ l'ensemble des règles construites à partir de $\mathcal{C}$ de la forme $(\forall K)(\mathcal{B} \Rightarrow \mathcal{H})$, où $K = Var(r)$; $\mathcal{B} \in \mathcal{Q}(\mathcal{C})$, $\mathcal{H} \in \mathcal{Q}(\mathcal{C})$, $\mathcal{B} \wedge \mathcal{H} \in \mathcal{Q}(\mathcal{C})$.

Exemple 2.17 *Dans le contexte de notre exemple courant (exemple 2.13), soit* $\mathcal{C} =< r, \mathcal{L}_C, \Gamma_C >$ *le contexte défini comme suit :*
- *Requête de référence* : $(\exists x_1, x_2)(Cust(k_1, x_1, x_2))$.
- $\mathcal{L}_C : \{Cust(k_1, x_1, x_2), Sale(k_1, k_2), Sale(k_1, k_2'), Prod(k_2, y_2), Prod(k_2', y_2')\}$ *et,*
- $\Gamma_C : \{x_1 = Enseignant, x_2 > 30\} \cup \{y_2 = a, y_2' = a | a \in dom(2, Prod)\}$.

La requête Q_1 *suivante et les requêtes qui composent la règle* I_1 *sont dans* $\mathcal{Q}(\mathcal{C})$*.*

$$\begin{array}{rcl} Q_1 & : & (cust(k_1, Enseignant, x_2) \wedge (x_2 > 30) \wedge Sale(k_1, k_2) \wedge Sale(k_2, Alcool))_r \\ I_1 & : & (\forall k_1) \ \ (Sale(k_1, k_2) \wedge Sale(k_1, k_2') \wedge Prod(k_2, Laitier) \Rightarrow Prod(k_2', Thé)) \end{array}$$

En vue d'explorer l'espace de recherche, les auteurs définissent deux opérateurs de spécialisation (ou de raffinement) : un par ajout de littéraux (ρ_L) et un par ajout de contrainte (ρ_C). Avant de donner une définition plus formelle de ces opérateurs de spécialisation, nous nous arrêtons un instant sur les contraintes pour voir sous quelle forme elles seront ajoutées.

Définition 2.45 *Soit* Γ *une conjonction de contraintes élémentaires. On appelle solution de* Γ *et on note* $Sol(\Gamma)$ *l'ensemble des substitutions* θ *telles que* $\Gamma\theta$ *est vrai. Une contrainte* Γ *est plus restrictive qu'une contrainte* Γ' *si* $Sol(\Gamma) \subseteq Sol(\Gamma')$*.*

Ainsi une requête $Q = (\mathcal{L} \wedge \Gamma)_r$ est spécialisée soit en ajoutant un nouvel atome L à $\mathcal{L}$, soit en introduisant une contrainte Γ' plus restrictive que Γ.

Définition 2.46 - Opérateurs de spécialisation.
Soit $\mathcal{C} =< r, \mathcal{L}_C, \Gamma_C >$ *un contexte et* $Q = (\mathcal{L} \wedge \Gamma)_r$ *une requête de* $\mathcal{Q}(\mathcal{C})$. $\rho_L(Q)$ *est l'ensemble des requêtes spécialisées* $(\mathcal{L} \wedge L \wedge \Gamma)_r$ *telles que*
- L *est un atome de* $\mathcal{L}_G$ *non présent dans* $\mathcal{L}$ *et,*
- $(\mathcal{L} \wedge L)$ *est connexe.*

$\rho_C(Q)$ *est l'ensemble des requêtes spécialisées* $(\mathcal{L} \wedge \Gamma')_r$ *telles que :*
- Γ' *est un sous-ensemble de* Γ_C *plus restrictif que* Γ, *i.e.* $Sol(\Gamma') \subset Sol(\Gamma)$,
- *Il n'existe pas de contrainte* $\Gamma'' \subseteq \Gamma_C$ *telle que* $Sol(\Gamma') \subset Sol(\Gamma'')$ *et* $Sol(\Gamma'') \subset Sol(\Gamma)$.

Soit X un ensemble de requêtes. On note :

$$\gamma_L(X) = \bigcup_{Q \in X} \rho_L(Q) \text{ et } \gamma_C(X) = \bigcup_{Q \in X} \rho_C(Q)$$

Si on note $\overline{\mathcal{Q}(\mathcal{C})}$ l'ensemble des requêtes de $\mathcal{Q}(\mathcal{C})$ sans contrainte, alors :

$$\overline{Q(C)} = \gamma_L^*(\{r\}) \text{ et } Q(C) = \bigcup_{Q \in \overline{Q(C)}} \gamma_C^*(Q)$$

où $\gamma_L^*(\{r\})$ et $\gamma_C^*(Q)$ sont définis par :

$$\gamma_L^*(\{r\}) = \bigcup_{i=1}^{|\mathcal{L}_G|} \gamma_L^i(r) \text{ et } \gamma_C^*(\mathcal{L}_r) = \bigcup_{j=1}^{|\Gamma_G|} \gamma_C^j(Q)$$

Il faut noter ici que l'on est en présence de plusieurs semi-treillis. Il y a le semi-treillis des requêtes sans contrainte dont l'espace de recherche est $\overline{Q(C)}$, qui utilise l'opérateur de spécialisation ρ_L, et pour chaque requête Q, il y a un semi-treillis correspondant à l'espace de recherche $\gamma_C^*(Q)$, dont l'opérateur de spécialisation est ρ_C.

Proposition 2.10 ρ_L *est complet relativement à* $\overline{\mathcal{Q}(\mathcal{C})}$.
ρ_C *est complet relativement à l'espace* $\gamma_C^*(Q)$ *pour tout* $Q \in \overline{\mathcal{Q}(\mathcal{C})}$.

Etant donné une requête $Q = (\mathcal{L} \wedge \Gamma)_r \in \mathcal{Q}(\mathcal{C})$, on peut noter que pour toute requête $Q' = (\mathcal{L}' \wedge \Gamma')_r$ de $\rho_L(Q)$, on a $Q' \sqsubseteq Q$. De même, pour toute requête Q' de $\rho_C(Q)$, on a $Q' \sqsubseteq Q$. D'où le corollaire de la proposition 2.9 :

Corollaire 2.1 *Soit* $\mathcal{G} =< r, \mathcal{L}_G, \Gamma_G >$ *un contexte. Pour toute requête* Q *de* $Q(C)$, *on a :*
- *Si* Q' *est une requête de* $\rho_L(Q)$, *alors* $Sup(Q'/r, S) \leq Sup(Q/r, S)$.
- *Si* Q' *est une requête de* $\rho_C(Q)$, *alors* $Sup(Q'/r, S) \leq Sup(Q/r, S)$.

La découverte des motifs intéressants va se faire ainsi en deux temps : l'extraction des requêtes fréquentes et ensuite l'extraction des règles à partir de celles-ci.
L'extraction des requêtes fréquentes peut s'exprimer dans le cadre du formalisme général de Mannila comme suit :

$$\mathcal{T}h(\mathcal{Q}(\mathcal{C}), r, S, q) = \{Q \in \mathcal{Q}(\mathcal{C}) \mid q(Q, r, S) \text{ est vrai dans } S\} \text{ où}$$
$$q(Q, r, S) \text{ est vrai ssi } Sup(Q/r, S) \geq minsup.$$

Le calcul de cette théorie va se faire en plusieurs étapes : il faut d'abord calculer la théorie relative à $\overline{\mathcal{Q}(\mathcal{C})}$ pour laquelle les motifs doivent vérifier le critère de qualité q, et ensuite pour chaque requête fréquente, il faut calculer la théorie correspondant à l'application des contraintes, avec le même critère de qualité q. L'algorithme est présenté à la figure II.17.

L'algorithme générique de Mannila sera utilisé pour le calcul de chaque théorie. La phase de génération du semi-treillis des requêtes sans contrainte va se faire par l'intermédiaire de l'opérateur de spécialisation ρ_L et celle de chaque semi-treillis avec contrainte par l'opérateur ρ_C

– **Entrée :** $(\mathcal{C} =< r, \mathcal{L}_C, \Gamma_C >, q, S)$
– **Sortie :** $\mathcal{T}h(Q(C), r, S, q)$

1. Calculer $\mathcal{T}h(\gamma_L^*(\{r\}), r, S, q) = \bigcup_{i=1}^{|\mathcal{L}_G|} \mathcal{T}h(\rho_L^i(r), r, S, q)$
2. Pour i=1 à $|\mathcal{L}_G|$
3. Pour tout $Q \in \mathcal{T}h(\rho_L^i(r), r, S, q)$
4. calculer $\mathcal{T}h(\rho_C^*(Q), r, S, q)$.
5. **Retourner** $\mathcal{T}h(Q(C), r, S, q)$.

FIG. II.17 – Algorithme d'extraction des requêtes fréquentes

correspondant. L'évaluation dans le semi-treillis des requêtes sans contraintes se fait par niveau et par requête. Cela s'explique par le fait que les différentes requêtes candidates d'un niveau contiennent différents prédicats. Par contre, l'évaluation des requêtes candidates dans un semi-treillis avec contrainte pourra se faire en une passe par niveau puisqu'il s'agit là de spécialisations d'une requête (sans contrainte) donnée.

Deuxième exemple de biais

Le deuxième exemple de biais, appelé W_{rmode} a été proposé par [27]. Il s'agit là d'un exemple de biais sémantique qui prend ses sources dans le système $PROGOL$ [70]. Ce système permet de déclarer à la fois les types des arguments des prédicats, ainsi que leurs modes. Le biais W_{rmode} se définit comme suit :

Définition 2.47 - Contexte d'extraction.
Un contexte d'extraction est un couple $\mathcal{C} =< r, W >$*, où :*
– *r est un atome de référence.*
– *W est un ensemble d'atomes avec des contraintes de type et de modes.*
– *Chaque argument d'un atome de W peut être étiqueté par un "-", un "+", "-type", "+type", et finalement par un "a".*

Lorsque l'argument est est étiqueté par un "-", on parle de variable de sortie. Lorsqu'il est étiqueté d'un "+", on parle de variable en entrée. Les arguments étiquetés par "-type" et "+type" sont appelés arguments typés. Les arguments étiquetés par "a" correspondent à des constantes.
L'idée de cet etiquettage est, comme nous l'avons dit plus haut, de restreindre le choix des variables dans les arguments des prédicats. Cela se traduit de façon plus précise par l'aplication de contraintes de mode et de contraintes de types que les requêtes définies par ce contexte doivent respecter.

Définition 2.48 - Contrainte de mode.
Une requête vérifie les contraintes de mode s'il existe un ordre sur les atomes qui la constituent tel que pour chaque atome, chaque variable d'entrée ait au moins une occurrence parmi les atomes qui le précédent, et pour chaque atome, aucune variable de sortie n'ait une occurrence parmi les atomes qui le précédent.

Définition 2.49 - Contrainte de type.
Une requête vérifie les contraintes de type si les arguments typés (des différents atomes) qui partagent un même nom de variable ont des types identiques.

Dans l'exemple suivant, nous notons $\mathcal{Q}(\mathcal{C}_1)$ l'ensemble des requêtes définies par par le contexte $\mathcal{C}_1$. Une définition formelle de cet ensemble sera donné par la suite.

Exemple 2.18 *Soit le contexte défini par* $C_1 =< p(x), \{p(-), q(+,-)\} >$, *où* $p(x)$ *est l'atome de référence et* W *est l'ensemble* $\{p(-), q(+,-)\}$. *La requête* $(\exists z)(p(x) \wedge q(x,z)) \in \mathcal{Q}(\mathcal{C}_1)$, *alors que la requête* $(\exists z)(p(x) \wedge q(z,x) \notin \mathcal{Q}(\mathcal{C}_1)$, *parce que le premier argument de* q *est une variable en entrée et devrait avoir une occurrence parmi les arguments des atomes qui le précédent, ce qui n'est pas le cas de* z.
Soit maintenant le contexte défini par $C_2 =< p1(x), \{p1(-s), p2(+s,-t), p3(+t,a)\} >$.
$(\exists z)(p1(x) \wedge p2(x,z) \wedge p3(z,a)) \in Q(C_2)$, *mais* $(\exists z)(p1(x) \wedge p2(x,z) \wedge p3(x,a)) \notin Q(C_2)$ *car le deuxième argument de p2 et le premier de p3 doivent être de même type.*

Nous allons maintenant définir l'opérateur de spécialisation qui déterminera de façon plus formelle l'ensemble des requêtes qui appartiennent à l'espace de recherche.

Définition 2.50 - Opérateur de spécialisation *Etant donnée une requête* $Q : (\exists Y)(r \wedge L)$ *de variables libres les variables de l'atome de référence* r. *L'opérateur de spécialisation* $\rho(Q)$ *est l'ensemble des requêtes* $(\exists Y')(r \wedge L \wedge p(t_1, \ldots, t_n))$ *de variables libres les variables de l'atome de référence telles que :*

- $p(w_1, \ldots, w_n)$ *appartient à* W *et,*
- $p(t_1, \ldots, t_n)$ *et* $p(w_1, \ldots, w_n)$ *sont en accord relativement à* Q *et* W,

où $p(t_1, \ldots, t_n)$ *et* $p(w_1, \ldots, w_n)$ *sont en accord relativement à* Q *et* W *si pour tout* $i \in \{1, 2, ..., n\}$ *:*

- $w_i =$ " $-$ " $\Rightarrow t_i$ *est une variable sans occurrence dans* Q,
- $w_i =$ " $+$ " $\Rightarrow t_i$ *est une variable ayant une occurrence dans* Q,
- $w_i =$ " $+ type$" $\Rightarrow t_i$ *est une variable de type "type" ayant une occurrence dans* Q,
- $w_i =$ "a" $\Rightarrow t_i$ *est une constante dans le domaine* $dom(i,p)$.

On peut noter qu'il s'agit là d'une spécialisation par ajout de littéraux. L'idée est ainsi d'effectuer des spécialisations successives en partant de l'atome de référence.

Exemple 2.19 *Soit le contexte défini par* $\mathcal{C}_3 =< p1(x), \{p1(-C), p2(+C,-), p3(+C,-,-)\} >$. *Alors la spécialisation de l'atome de référence* $p1(x)$ *est définie par les deux requêtes suivantes :*

$$\begin{array}{rcl} \rho(p1(x)) & \rightarrow & \{(\exists y)(p1(x) \wedge p2(x,y)), \\ & & (\exists w,z)(p1(x) \wedge p3(x,w,z))\} \end{array}$$

Définition 2.51 - Espace de recherche.
Etant donné un contexte d'extraction $\mathcal{C} =< r, W >$, *l'espace de recherche* $Q(C)$ *est défini par les spécialisations successives de l'atome de référence :*

$$\mathcal{Q}(\mathcal{C}) = \rho^*(r)$$

Proposition 2.11 ρ *est un opérateur complet relativement à* $\mathcal{Q}(\mathcal{C})$.

Etant donné une requête $Q \in \mathcal{Q}(\mathcal{C})$, on peut noter que pour toute requête Q' de $\rho(Q)$, on a $Q' \sqsubseteq Q$. D'où le corollaire de la proposition 2.9

Corollaire 2.2 *Soit* $C =< r, W >$ *un contexte. Pour toute requête* Q *de* $Q(C)$, *on a :*

$$Si\ Q'\ est\ une\ requête\ de\ \rho(Q),\ alors\ Sup(Q'/r, S) \leq Sup(Q/r, S).$$

– **Entrée :** Ensemble de données S ; Ensemble de requêtes $\mathcal{Q}$; atome de référence r.
– **Sortie :** les supports des requêtes $Q_j \in \mathcal{Q}$.

1. Pour chaque requête $Q_j \in \mathcal{Q}$, **Faire** $numerateur_{Sup}(Q_j) = 0$;
2. Pour chaque $k_i \in r(S)$, **Faire :**
3. Isoler la portion S_i de S relative à k_i.
4. Pour chaque requête $Q_j \in \mathcal{Q}$, **Faire :**
5. Si $Q_j\theta_{k_i}(S_i)$ est non vide, incrémenter $numerateur_{Sup}(Q_j)$.
6. Pour chaque requête $Q \in \mathcal{Q}$, retourner $numerateur_{Sup}(Q_j)$ / $|r(S)|$.

FIG. II.18 – Algorithme d'évaluation du support

.

L'extraction des requêtes fréquentes peut s'exprimer dans le cadre du formalisme général de Mannila comme suit :

$$\mathcal{T}h(Q(C), r, S, q) = \{Q \in Q(C) \mid q(Q, r, S) \textit{ est vrai dans } S\} \textit{ où}$$
$$q(Q, r, S) \textit{ est vrai ssi } Sup(Q/r, S) \geq minsup.$$

Le calcul de cette théorie se fait par l'algorithme $WARMR$ qui est une instance de l'algorithme générique de Mannila et Toivonen. La phase de génération est effectuée par l'intermédiaire de l'opérateur de spécialisation, qui à chaque niveau, permet de générer les requêtes candidates du niveau supérieur. L'évaluation des supports des candidats se fait par niveau et par valeurs de K, l'ensemble des variables de référence. A un niveau donné, l'algorithme isole les portions S_i de l'ensemble des données relatives aux valeurs k_i des variables de référence, et pour chaque S_i, il incrémente les supports des requêtes candidates Q pour lesquelles $Q\theta_{k_i}(S_i)$ est non vide, où θ_{k_i} est une substitution sur les valeurs de k_i. Cette méthode suppose que les ensemble de données S_i peuvent être stockés en mémoire. L'algorithme d'évaluation est donné à la figure II.18.
On peut facilement caractériser les ensembles S_i dans le cas particulier où tout atome d'un motif candidat contient la ou les variables de référence.
Soit $K = (x_1, x_2, \ldots, x_n)$ le n-uplet des variables de référence $x_1, x_2, \ldots, x_n$. Pour tout $k_i = (k_{i_1}, k_{i_2}, \ldots, k_{i_n}) \in dom(1, r) \times \ldots \times dom(n, r)$, la portion S_i de S relative à k_i

$$S_i = \bigcup_{L \in W} \{L\theta \in S \mid \theta(x_1) = k_{i_1} \wedge \ldots \theta(x_n) = k_{i_n}\},$$

où r est l'atome de référence, et S l'ensemble des faits.

Exemple 2.20 *L'ensemble de données (de faits) est défini comme suit :*
$S = \{p1(1), p1(2), p1(3),$
$p2(1,'algo'), p2(1,'maths'), p2(2,'algo'), p2(2,'prog.'), p2(3,'prog'), p2(3,'se'),$
$p3(1,'iut','Blois', p3(2,'iut','Paris'), p3(3,'deug','Paris')\}$. *Considérons le contexte* C_4 *défini comme suit :*
$C_4 =< p1(x), \{p1(-C), p2(+C, -), p3(+C, a, a)\} >$.
Considèrons maintenant quelques candidats de niveau 2 définis par $\rho(p1(x))$ *:*
$Q1 : (\exists y)(p1(x) \wedge p2(x, y))$, $Q2 : p1(x) \wedge p3(x,'iut','Blois')$, $Q3 : p1(x) \wedge p3(x,'deug','Blois')$, $Q4 : p1(x) \wedge p3(x,'deug','Paris')$.
Nous allons faire une évaluation des requêtes candidates en considèrant les sous-ensembles S_i *de* S.

- $S[x = 1] = \{p1(1), p2(1,'algo'), p2(1,'maths'), p3(1,'iut','Blois')\}$.
  Si on applique la substitution $\{x/1\}$ *aux différentes requêtes, seuls les supports de* $Q1$ *et* $Q2$ *seront incrémentés.*
- $S[x = 2] = \{p1(2), p2(2,'algo'), p2(2,'prog.'), p3(2,'iut','Paris')\}$.
  L'application de la substitution $\{x/2\}$ *conduit à l'incrémentation du support de* $Q1$ *seulement.*
- $S[x = 3] = \{p1(3), p2(3,'prog.'), p2(3,'se'), p3(3,'deug','Paris')\}$.
  Incrémentation des supports de $Q1$ *et* $Q4$.

Ainsi $Sup(Q1/r, S) = 3/3 = 1,\ Sup(Q2/r, S) = 1/3,\ Sup(Q3/r, S) = 0,\ Sup(Q4/r, S) = 1/3.$

2.6.5 Conclusion

Les approches multi-relationnelles pour l'extraction des motifs intéressants dans les bases de données, que nous avons présentées, utilisent la logique de premier ordre pour la représentation des données et des motifs. Cette représentation permet d'obtenir une puissance d'expression des motifs plus importante, et par conséquent un espace de recherche plus grand que dans le cas transactionnel, d'où la nécessité de définir des biais, et notamment des biais de langage. Ces biais permettent de focaliser la recherche sur les motifs d'une certaine forme. Nous avons présenté un exemple de biais syntaxique et un exemple de biais sémantique. L'approche que nous allons présenter plus loin, et qui constitue notre contribution dans ce cadre, propose un autre exemple de biais syntaxique.

3 Représentations condensées

3.1 Introduction

L'Extraction de motifs fréquents est une des tâches fondamentales du Data Mining. De nombreux algorithmes ont été proposés pour améliorer ses performances. Cependant, le nombre de motifs découverts est généralement très important. C'est donc dire que le besoin de représentations plus petites qui permettent de retrouver ces motifs ainsi que leurs supports sans accéder à la base de données est manifeste. D'autre part, l'expérience a montré que l'extraction de motifs fréquents n'est pas toujours possible du fait de leur nombre et de leur taille, par exemple dans le cas de données corrélées. En effet, dans ce cas il y a un très grand nombre de motifs de très grande taille, ce qui fait qu'ils ne tiennent pas tous en mémoire. Il serait alors, dans de telles situations, judicieux de disposer de représentations qui, à défaut de trouver des valeurs exactes des supports des motifs, permettent d'avoir une approximation de ces derniers. Il s'agit là de représentations condensées.
Le concept de représentation condensée a été introduit dans [62] et formalisé dans le contexte de représentation $\epsilon - adequate$. Intuitivement, une représentation $\epsilon - adequate$ est une représentation de données pouvant être remplacée par une autre pour répondre aux mêmes types de requêtes, mais avec une perte de précision ϵ.
Il existe cependant d'autres représentations qui permettent de reconstituer entièrement l'ensemble des motifs fréquents.
Avant de donner une définition de représentation condensée des motifs fréquents, précisons que nous travaillerons avec le formalisme $\mathbb{O}, \mathbb{R}, \mathbb{P}$ présenté dans la partie 2 sur la découverte de motifs intéressants. L'ensemble $\mathbb{P}$ des motifs est ici l'ensemble $\mathbb{I}$ des items. L'ensemble des données est ainsi le triplet $\mathbb{D} = (\mathbb{O}, \mathbb{I}, \mathbb{R})$.
Rappelons que f est la fonction qui fait correspondre à un motif X l'ensemble des objets qui le contiennent, et g sa fonction duale :

$$f(X) = \{\ o \in \mathbb{O} \mid \forall x \in X, (o, x) \in \mathbb{R}\}$$
$$g(O) = \{\ x \in X \mid \forall\ o \in O, (o, x) \in \mathbb{R}\}$$

Le couple (f, g) est une connection de Galois entre $2^{\mathbb{O}}$ et $2^{\mathbb{I}}$. Les opérateurs gof et fog sont des fermetures de Galois. Le support d'un itemset X se calcule par la formule suivante :

$$sup(X) = card(f(X)) \ / \ card(\mathbb{O}) \ = \ |f(X)| \ / \ |\mathbb{O}|$$

Pour un seuil $minsup \in [0, 1]$, un itemset X est dit fréquent si $sup(X) \geq minsup$. On note :

- $Freq(\mathbb{D})$ l'ensemble des itemsets fréquents.
- $Sup(\mathbb{D}) = \{\ (X, sup(X) \mid X \subseteq \mathbb{I}\ \}$ l'ensemble de tous les couples (un itemset, son support).
- $FreqSup(\mathbb{D}) = \{\ (X, sup(X) \mid X \in Freq(\mathbb{D})\ \}$ l'ensemble de tous les couples (un fréquent, son support).

Nous allons passer en revue les différentes représentations condensées, et à chaque fois, nous ferons le lien avec les itemsets fréquents. Nous parlerons dans un premier temps de représentations condensées exactes [75, 6, 18, 19] et ensuite de représentation condensée approximative [15, 16].

3.2 Représentations condensées exactes

Définition 3.1 *Etant donné un ensemble de données $\mathbb{D}$, soit $X_1, ..., X_n$ et Y des sous-ensembles de $Sup(\mathbb{D})$. $R = \{\ X_1, ..., X_n\ \}$ est une représentation condensée exacte de Y si :*

1. $X_1 \cup ... \cup X_n \subseteq Y$.

2. Il existe une fonction F indépendante de $\mathbb{D}$ permettant de calculer Y à partir de R.

L'item 1 de la définition indique que R est un sous-ensemble de Y (d'où le terme de représentation condensée). L'item 2 indique que F permet de calculer Y à partir de R sans accéder aux données.

3.2.1 Les maximaux

Les itemsets fréquents maximaux sont les itemsets dont tous les sur-ensembles sont non fréquents et tous les sous-ensembles fréquents. Ils constituent une bordure des itemsets les plus spécifiques que l'on appelle la *frontière positive* [64]. Les auteurs donnent d'ailleurs une définition plus générale des frontières.

Définition 3.2 *Soit S un ensemble de motifs, et $\preccurlyeq$ une relation de spécialisation, où $\varphi \preccurlyeq \psi$ veut dire que φ est plus spécifique que ψ .*
La frontière positive de S, notée $Bd^+(S)$, est l'ensemble des motifs les plus spécifiques de S.

$$Bd^+(S) = \{\ \varphi \in S \mid \nexists\ \psi \in S,\ \psi \prec \varphi\}$$

La frontière négative de S, notée $Bd^-(S)$, est l'ensemble des motifs les plus généraux qui ne sont pas dans S.

$$Bd^-(S) = \{\ \varphi \in S \mid \nexists\ \psi \notin S,\ \varphi \prec \psi\}$$

Dans notre cadre, l'ensemble des motifs est l'ensemble des itemsets et la relation de spécialisation est l'inclusion ensembliste ($\subseteq$). On obtient alors les formules suivantes :

$$Bd^+(S) = \{\ \varphi \in S \mid \nexists\ \psi \in S,\ \varphi \subset \psi\}$$

$$Bd^-(S) = \{\ \varphi \in S \mid \nexists\ \psi \notin S,\ \psi \subset \varphi\}$$

Définition 3.3 *Etant donné un ensemble de données $\mathbb{D}$:*

$$BdSup^+(\mathbb{D}) = \{\ (X, sup(X) \mid X \in Bd^+(\mathbb{D}))\ \}$$
$$BdSup^-(\mathbb{D}) = \{\ (X, sup(X) \mid X \in Bd^-(\mathbb{D}))\ \}$$

Si on applique les définitions ci-dessus à l'ensemble des itemsets fréquents, on obtient les formules suivantes :

$$FreqM(\mathbb{D}) = Bd^+(Freq(\mathbb{D}) = \{X \in Freq(\mathbb{D}) \mid \nexists\ Y \in Freq(\mathbb{D}), X \subset Y\}$$
$$Bd^-(Freq(\mathbb{D})) = \{X \subseteq \mathbb{I}\ | X \notin Freq(\mathbb{D}) \wedge\ \forall\ Y \subset X : Y \in Freq(\mathbb{D})\}.$$

où $FreqM(\mathbb{D})$ est l'ensemble des fréquents maximaux.
On peut facilement montrer que tous les itemsets fréquents peuvent être dérivés à partir de ces frontières. Un algorithme efficace qui utilise cette approche est $Max - Miner$ [9] qui trouve d'abord la frontière positive et en déduit les fréquents.
Si l'ensemble des itemsets fréquents peut être déduit de la frontière positive (tous les itemsets qui sont plus généraux) ou de la frontière négative (tous les itemsets plus spécifiques), il n'en est pas de même pour leurs supports, d'où la proposition suivante :

Proposition 3.1 *Soit $FreqMSup(\mathbb{D})$ l'ensemble des itemsets fréquents maximaux munis de leurs supports :*

$$FreqMSup(\mathbb{D}) = \{\ (X, sup(X)) \mid X \in FreqM(\mathbb{D})\}$$
$\{\ FreqMSup(\mathbb{D})\ \}$ *ne constitue pas une représentation condensée de $FreqSup(\mathbb{D})$.*

3.2.2 Les fermés

Les itemsets fermés sont définis à partir de la fermeture de Galois. Ils ont été présentés dans le contexte des règles d'association par [75].

Définition 3.4 *Soit $X \subseteq \mathbb{I}$ un itemset. On appelle $gof(X)$ la fermeture de X. X est fermé s'il est égal à sa fermeture, c'est à dire si $gof(X) = X$.*

Proposition 3.2 *Le plus petit itemset fermé qui contient un itemset X est égal à $gof(X)$.*

On peut introduire une relation d'équivalence en utilisant la fermeture.

Définition 3.5 - Relation d'équivalence [7].
Deux itemsets X et Y sont équivalents s'ils ont la même fermeture dans $\mathbb{D}$:

$$X \ \theta \ Y \Leftrightarrow gof(X) = gof(Y)$$

Les auteurs [7] ont montré que la relation d'équivalence implique la proposition suivante :

Proposition 3.3
- *Deux itemsets équivalents ont le même support, i.e. $X \ \theta \ Y \Rightarrow sup(X) = sup(Y)$.*
- *Si deux itemsets X et Y ont même support et que $X \subseteq Y$, alors ils sont dans la même classe d'équivalence, i.e. si $sup(X) = sup(Y)$ et $X \subseteq Y$, alors $X \ \theta \ Y$.*
- *Chaque classe d'équivalence a exactement un itemset maximal qui est fermé.*

Définition 3.6 *Soit $Closed$ l'ensemble des itemsets fermés de $\mathbb{D}$. La paire $(Closed, \subseteq)$ est un treillis appelé treillis des itemsets fermés.*

L'algorithme d'extraction des itemsets fermés, *Close* [75], se distingue notamment des autres, parce qu'il est basé sur l'élagage du treillis des fermés pour trouver les itemsets fréquents. L'utilisation de ce treillis qui est un sous-ensemble du treillis des parties, améliore sensiblement la performance de l'extraction parce qu'il y a beaucoup moins de fermés fréquents que d'itemsets fréquents. A partir de ce treillis, les auteurs déduisent un certain nombre de propriétés :

Proposition 3.4 *Tous les sous-ensembles fermés d'un fermé fréquent sont fréquents.*

Proposition 3.5 *Tous les sur-ensembles fermés d'un fermé non fréquent sont non fréquents.*

Proposition 3.6 *Soit $FreqC(\mathbb{D})$ l'ensemble de tous les fermés fréquents. L'ensemble $FreqMC(\mathbb{D})$ des fréquents fermés maximaux est défini comme suit :*

$$FreqMC(\mathbb{D}) = \{ \ X \in FreqC(\mathbb{D}) \mid \nexists \ Y \in FreqC(\mathbb{D}), \ X \subset Y\}$$

L'ensemble des maximaux fréquents est égal à l'ensemble des maximaux fermés fréquents, i.e. $FreqMC(\mathbb{D}) = FreqM(\mathbb{D})$.

Proposition 3.7 *Le support d'un itemset fréquent X non fermé est égal au support du plus petit fermé fréquent qui le contient.*

En se basant sur ces propriétés, l'algorithme *Close* commence par trouver les fermés fréquents, et dérive à partir d'eux les itemsets fréquents, pour finalement générer les règles d'association. L'algorithme est particulièrement efficace lorsque les données sont corrélées, contrairement aux algorithmes de type Apriori. On dit des données qu'elles sont corrélées lorsque la présence d'un attribut ou d'un ensemble d'attributs implique la présence d'un autre (en d'autres termes, il y a beaucoup de règles avec une confiance de 1).

1. **Entrée** : $FreqCSup(\mathbb{D})$;
2. **Sortie** : $FreqSup(\mathbb{D})$;
3. k = 0 ;
4. Pour $k = 1$ à $|\mathbb{P}|$
5. $L_k = \emptyset$;
6. **Pour tout** $c \in FreqCSup(\mathbb{D})$ **Faire**
7. $L_{\|c\|} = L_{\|c\|} \cup \{c\}$;
8. **Si** $(k < \|c\|)$ **alors** $k = \|c\|$;
9. **Fin**
10. **Pour** $(i = k; i > 1; i - -)$ **Faire**
11. **Pour tout** itemset $c \in L_i$ **Faire**
12. **Pour tous** (i-1)-sous-ensembles s de c **Faire**
13. **Si** $(s \notin L_{i-1})$ **Faire**
14. $\sup(s) = \sup(c)$;
15. $L_{i-1} = L_{i-1} \cup \{s\}$;
16. **Fin**
17. **Fin**
18. **Fin**
19. **Fin**

FIG. II.19 – Algorithme de reconstitution des itemsets fréquents à partir des fermés fréquents.

Proposition 3.8 *Pour tout ensemble de données* $\mathbb{D}$ *:*

$$FreqM(\mathbb{D}) \subseteq FreqC(\mathbb{D}) \subseteq Freq(\mathbb{D}).$$

Proposition 3.9 *Soit* $FreqCSup(\mathbb{D})$ *l'ensemble des itemsets fermés fréquents munis de leurs supports, i.e.* $FreqCSup(\mathbb{D}) = \{ (X, sup(X)) \mid X \in FreqC(\mathbb{D})\}$. $\{ FreqCSup(\mathbb{D}) \}$ *est une* **représentation condensée** *exacte de* $FreqSup(\mathbb{D})$.

En effet, on peut grâce à la proposition 3.6 retrouver tous les itemsets fréquents, et grâce à la proposition 3.7 recalculer tous les supports des non fermés sans accéder à la base de données. D'ailleurs, les auteurs proposent dans [75] un algorithme qui permet de faire cette reconstitution. Le pseudo-code de cet algorithme est présenté à la figure II.19.

L'algorithme se résume comme suit : l'ensemble des itemsets fermés fréquents est découpé en ensembles L_i, $i \in \{1, \ldots, k\}$, où i représente la taille d'un itemset, ce qui permet de trouver en même temps la taille k des itemsets maximaux. Ce découpage est décrit dans l'algorithme de la ligne 3 à la ligne 7, où $L_{\|c\|}$ représente l'ensemble des itemsets fréquents de taille c. Ensuite, les itemsets de L_i sont construits en partant de L_k jusqu'à L_1. A chaque itération, l'ensemble L_{i-1} est construit en utilisant les itemsets de L_i. Pour chaque itemset c de taille i, on génére tous les itemsets de taille $i - 1$. Tous les sous-motifs qui ne sont pas dans L_{i-1} sont ajoutés avec pour valeur de support le support de c.

3.2.3 Les clés

Le concept de clé a été introduit par l'équipe de Lakhal [6]. Les clés sont les minimaux des classes d'équivalence définies par la relation θ.

Définition 3.7 *Les itemsets clés d'un ensemble de données $\mathbb{D}$ sont les minimaux (au sens de l'inclusion) de toutes les classes d'équivalence définies sur $\mathbb{D}$ par la relation θ. On note $Key(\mathbb{D})$ l'ensemble des clés.*

Les clés sont aussi appelés *ensembles libres* [16] ou *générateurs* [55].

Proposition 3.10 *Tout sur-ensemble d'un itemset non clé est non clé.*

Proposition 3.11 *Tout sous-ensemble d'un itemset clé est un itemset clé.*

Cette approche utilise comme pour les fermés les propriétés de la relation d'équivalence θ que nous rappelons ici ([6]) :

Proposition 3.12 *Soient deux itemsets X et Y.*
- *$X\ \theta\ Y \Rightarrow sup(X) = sup(Y)$;*
- *$X \subseteq Y$ et $sup(X) = sup(Y) \Rightarrow X\ \theta\ Y$*

Cela veut dire qu'il suffit de connaître le support d'un élément de la classe d'équivalence pour connaître celui des autres. Seulement, la classe d'équivalence n'est pas connue d'avance, et un algorithme, *Pascal* qui permet de la construire, a été proposé dans [6]. C'est un algorithme de type Apriori qui fait une extraction par niveau. A chaque niveau, il y a génération et évaluation des candidats.
L'algorithme *Pascal* est principalement basé sur le comptage par inférence : lors de la génération des itemsets candidats, s'il apparaît un itemset non clé, un itemset clé appartenant à la même classe d'équivalence a déjà été traité lors d'une itération précédente. Ceci est traduit par la propriété suivante :

Proposition 3.13 - Comptage par inférence.
X est un itemset non clé si et seulement si $sup(X) = min_{x \in X}(sup(X \setminus \{x\}))$

L'algorithme fonctionne ainsi comme Apriori, mais ne compte pas le support des itemsets non clés, ce qui permet de réduire considérablement le nombre de calculs, et donc le nombre d'accès à la base de données.
Il est important de souligner que l'algorithme *Pascal* est un algorithme très efficace. Dans [7], les auteurs font une comparaison des algorithmes *Apriori*, *Pascal*, $Max - Miner$ et *Close*. Leurs résultats montrent que *Pascal* donne des temps de réponse équivalents à ceux de *Apriori* et *Max-Miner* et inférieurs à ceux de *Close* dans le cas de données faiblement corrélées, et qu'il est le plus efficace parmi les quatre dans le cas de données corrélées.
Soit $FreqKey(\mathbb{D})$ l'ensemble des clés fréquents. L'ensemble $FreqKeySup(\mathbb{D})$ des clés fréquents munis de leurs supports est défini comme suit :

$$FreqKeySup(\mathbb{D}) = \{\ (X, sup(X) \mid X \in FreqKey(\mathbb{D})\ \}$$

Voyons maintenant le lien qui existe entre $FreqKeySup(\mathbb{D})$ et $FreqSup(\mathbb{D})$. Comme le montre la proposition 3.13, tous les supports des itemsets non clés fréquents peuvent être déduits de ceux des clés fréquents. Seulement, étant donné un itemset quelconque, on ne peut pas savoir s'il est fréquent ou pas. Pour avoir cette information, il faut rajouter la bordure négative des clés fréquents.

Proposition 3.14 *Soit* $Bd^-(FreqKey(\mathbb{D})) = \{X \subseteq \mathbb{I} \mid X \notin FreqKey(\mathbb{D}) \wedge \forall Y \subset X : Y \in FreqKey(\mathbb{D})\}$ *la frontière négative des clés fréquents.*
L'ensemble $\{ FreqKeySup(\mathbb{D}), BdSup^-(FreqKey(\mathbb{D})) \}$ *est une* **représentation condensée exacte** *de* $FreqSup(\mathbb{D}) \cup BdSup^-(FreqKey(\mathbb{D}))$.

On notera que $\{ FreqKeySup(\mathbb{D}), BdSup^-(FreqKey(\mathbb{D})) \}$ n'est pas une représentation condensée de $FreqSup(\mathbb{D})$ tout seul, sinon le point 1 de la définition 3.1 ne serait pas vérifié.
L'algorithme de reconstitution des itemsets fréquents se résume comme suit : étant donné un itemset candidat, on vérifie si un de ses sous-ensembles n'est pas élément de la frontière négative. Si tel est le cas, alors il n'est pas fréquent. Sinon, il s'agit d'un itemset fréquent et son support est calculé en utilisant la proposition 3.13.

Proposition 3.15 *Soit* $Bd^+(FreqKey(\mathbb{D})) = \{X \in FreqKey(\mathbb{D}) \mid \nexists Y \in FreqKey(\mathbb{D}), X \subset Y\}$ *la frontière positive des clés fréquents.*
L'ensemble $\{ FreqKeySup(\mathbb{D}), BdSup^+(FreqKey(\mathbb{D})) \}$ *est une* **représentation condensée exacte** *de* $FreqSup(\mathbb{D})$.

Ici le point 1 de la définition 3.1 est vérifié, puisque $BdSup^+(FreqKey(\mathbb{D}))$ est bien un sous-ensemble de $FreqSup(\mathbb{D})$.

L'ensemble des itemsets peut ainsi être découpé en classes d'équivalence. Dans chaque classe d'équivalence, il y a un maximal qui est un fermé, et les minimaux qui sont les clés. Cette composition des classes explique la proposition suivante :

Proposition 3.16 *L'ensemble des fermés fréquents est plus petit que celui des clés fréquents :*

$$|FreqC(\mathbb{D})| < |FreqKey(\mathbb{D})|$$

On peut donc dire que la représentation condensée de la proposition 3.9 est plus condensée que les représentations condensées des propositions 3.14 et 3.15.

3.2.4 Ensembles libres disjonctifs

Les ensembles libres disjonctifs (disjunction-free sets) sont une autre représentation proposée par [18]. Ce sont essentiellement des extensions des ensembles libres, donc des clés. La définition d'ensemble libre disjonctif repose sur celle de règle disjonctive simple.

Définition 3.8 *Soit* X *un itemset. Une règle disjonctive simple basée sur* X *est une expression de la forme* $Y \Rightarrow A \vee B$, *où* $Y \subset X$ *et* $A, B \in X \backslash Y$.

Les auteurs notent que A et B sont des items pas forcément différents, ainsi la règle $Y \Rightarrow A \vee A$ est un cas particulier de règle simple disjonctive. Par la suite, on note $S(X) = card(f(X)) = |f(X)|$
.
Etant donné un ensemble de données $\mathbb{D}$, une règle simple disjonctive $Y \Rightarrow A \vee B$ est valide dans $\mathbb{D}$ si sa confiance est égale à 1, i.e. $f(Y) = f(Y \cup \{A\}) \cup f(Y \cup \{B\}$. A partir de la définition ci-dessus, les auteurs proposent un lemme fondamental pour la suite.

Lemme 1 *Soit* X *un itemset, et* A, B *des items dans* X. *Alors, il existe* $Y \subset X$ *tel que* $Y \Rightarrow A \vee B$ *soit une règle valide si et seulement si* $S(X) = S(X \backslash \{A\}) + S(X \backslash \{B\}) - S(X \backslash \{A, B\})$.

Définition 3.9 *Un itemset* X *est un ensemble libre disjonctif dans* $\mathbb{D}$ *s'il n'existe pas de règle disjonctive simple valide basée sur* X *dans* $\mathbb{D}$. *L'ensemble des itemsets libres disjonctifs est noté* $DFree(\mathbb{D})$.

Cette définition implique que si un itemset n'est pas un ensemble libre disjonctif, alors son support peut être déduit de ceux de ses sous-ensembles par le lemme 1. Ce lemme montre aussi que les ensembles libres représentent un cas particulier des ensembles libres disjonctifs, i.e. lorsque $A = B$. Il montre par la même occasion que si un itemset n'est pas libre, alors il n'est pas un ensemble libre disjonctif. En effet, si un itemset Y n'est pas libre, alors il existe un sous-ensemble X de Y avec le même support, ce qui veut dire que la règle $Y \Rightarrow X \vee X$ est valide. Par conséquent, Y n'est pas un ensemble libre disjonctif, d'où la proposition :

Proposition 3.17 *Pour tout ensemble de données* $\mathbb{D}$ *:*

$$DFree(\mathbb{D}) \subseteq Key(\mathbb{D})$$

Proposition 3.18 - Anti-monotonie. *Soit X un itemset. Pour tout* $Y \subseteq X$*, si* $X \in DFree(\mathbb{D})$*, alors* $Y \in DFree(\mathbb{D})$*.*

C'est ainsi que les auteurs proposent deux algorithmes d'extraction des ensembles libres disjonctifs fréquents : l'un (HLinEx) est une instance de l'algorithme générique de Mannila et Toivonen, et l'autre (VLinEx) est un algorithme en profondeur.
Notons $FreqDFree(\mathbb{D})$ l'ensemble des itemsets libres disjonctifs fréquents, $FreqDFreeSup(\mathbb{D})$ celui avec les supports et $Bd^{-}(FreqDFree(\mathbb{D}))$ la frontière négative des ensembles libres disjonctifs fréquents :

$$Bd^{-}(FreqDFree(\mathbb{D})) = \{\ X\ \subseteq \mathbb{I} \mid X \notin FreqDFree(\mathbb{D}) \wedge\ \forall Y \subset X : Y \in FreqDFree(\mathbb{D})\ \}.$$

Proposition 3.19 $\{\ FreqDFreeSup(\mathbb{D}), BdSup^{-}(FreqDFree(\mathbb{D}))\}$ *est une* **représentation condensée exacte** *de* $FreqSup(\mathbb{D}) \cup BdSup^{-}(FreqDFree(\mathbb{D}))$*.*

L'algorithme de reconstitution des fréquents est le suivant : soit X un itemset quelconque, s'il existe un sous-ensemble de X dans $Bd^{-}(FreqDFree(\mathbb{D}))$, alors X n'est pas fréquent. Sinon X est un itemset fréquent non disjonctif, et alors son support peut être calculé par le lemme 1.

3.2.5 Les itemsets non dérivables

Ce concept a été introduit par T. Calders et B. Goethals [19]. Les auteurs présentent des règles (système d'inégalités), appelées règles déductives, qui permettent de borner les supports des itemsets candidats sans accéder à la base de données. Cette approche est basée sur le principe de l'inclusion-exclusion. Nous allons utiliser un exemple qui part de ce principe pour produire deux règles déductives et généralise les travaux sur les ensemnles libres.

Exemple 3.1 *Soient X, Y et Z trois ensembles. Le principe d'inclusion-exclusion s'énonce comme suit :*

$$|X \cup Y \cup Z| = |X| + |Y| + |Z| - |X \cap Y| - |X \cap Z| - |Y \cap Z| + |X \cap Y \cap Z|$$

Supposons qu'on dispose d'un ensemble de n transactions T et d'un ensemble d'items A, B, C... Alors $f(A)$*,* $f(B)$ *et* $f(C)$ *sont respectivement les ensembles des transactions qui contiennent A, B, ou C.*
Pour $X = f(A)$*,* $Y = f(B)$*,* $Z = f(C)$*, le principe d'inclusion-exclusion devient :*
$|f(A) \cup f(B) \cup f(C)| = |f(A)| + |f(B)| + |f(C)| - |f(A) \cap f(B)| - |f(A) \cap f(C)| - |f(B) \cap f(C)| + |f(A) \cap f(B) \cap f(C)|$
$|f(A) \cup f(B) \cup f(C)| = |f(A)| + |f(B)| + |f(C)| - |f(AB)| - |f(AC)| - |f(BC)| + |f(ABC)|$

Puisque d'une part $|f(A) \cup f(B) \cup f(C)| \leq n$ *, et d'autre part,* $S(X) = |f(X)|$*, on obtient :*

$$S(ABC) \leq S(AB) + S(AC) + S(BC) - S(A) - S(B) - S(C) + n$$

ce que l'on peut réecrire comme suit :

$$S(ABC) \leq S(AB) + S(AC) + S(BC) - S(A) - S(B) - S(C) + S(\emptyset)$$

Cette règle est notée $R_\emptyset(ABC)$*. De la même façon :*

$$|f(AB) \cup f(AC)| = |f(AB)| + |f(AC)| - |f(ABC)|$$

Puisque $|f(AB) \cup f(AC)| \leq |f(A)|$*, on en déduit que :*

$$S(ABC) \geq S(AB) + S(AC) - S(A)$$

Celle-ci est la règle $R_A(ABC)$*.*

Nous avons là un exemple de règles déductives qui montrent que l'on peut utiliser les supports des sous-ensembles de ABC pour borner le support de ce dernier. Les auteurs généralisent cette remarque par le théorème suivant :

Théoreme 3.1 *Soit* Y *un itemset. Etant donné les supports de* X *pour tout* $X \subset Y$*, les bornes inférieure et supérieure du support de* Y *peuvent être calculées à partir des règles de déduction.*

Mieux, on peut à partir d'un système de règles déductives trouver la valeur *exacte* du support d'un itemset X à partir des supports de ses sous-ensembles sans accéder à la base de données.

Définition 3.10 *Un itemset* X *est dit dérivable (***DI***), si on peut à partir d'un système de règles déductives trouver la valeur exacte de son support à partir des supports de ses sous-ensembles sans accéder à la base de données.*

Définition 3.11 *Un itemset* X *dont on ne peut pas à partir d'un système de règles déductives trouver la valeur exacte de son support à partir des supports de ses sous-ensembles sans accéder à la base de données est dit non dérivables(***NDI***). L'ensemble des NDI est noté* $NDI(\mathbb{D})$

Les auteurs montrent qu'à partir de l'ensemble des NDI fréquents, on peut calculer les supports de tous les autres fréquents. Pour cela, ils démontrent que lorsqu'un itemset X est non dérivable, alors tous ses sous-ensembles le sont aussi, en énonçant la proposition suivante :

Proposition 3.20 - Anti-monotonie. *Soit* X *et* Y *deux itemsets. Si* $X \subset Y$*, et* X *dérivable, alors* Y *est aussi dérivable.*

Les auteurs proposent par conséquent un algorithme de calcul des fréquents NDI. C'est un algorithme d'extraction par niveau de type Apriori dont l'idée principale est la suivante : au niveau i, on évalue le support d'un itemset candidat seulement si tous ses sous-ensembles sont fréquents et si les bornes inférieure et supérieure ne permettent pas de dériver exactement son support.
On peut par conséquent énoncer la proposition suivante :

Proposition 3.21 *Soit* $FreqNDISup(\mathbb{D})$ *l'ensemble des NDI fréquents munis de leurs supports.* $\{\ FreqNDISup(\mathbb{D})\ \}$ *est une* **représentation condensée exacte** *de* $FreqSup(\mathbb{D})$*.*

Nous allons maintenant montrer comment générer l'ensemble des itemsets fréquents à partir des NDI fréquents. Pour cela, nous partons de la remarque suivante : si un itemset X n'appartient pas à l'ensemble des NDI fréquents, alors il est :

1. soit non fréquent, soit
2. soit dérivable.

Pour savoir à quel cas correspond X, on dérive, par le calcul des règles déductives, les bornes inférieure et supérieure $[l, u]$ de son support. Si $l = u$, alors on connaît exactement le support de X, et on peut dire si X est fréquent ou pas. Si $l \neq u$, alors on sait que X n'est pas fréquent. En effet, s'il l'avait été, il serait dans l'ensemble des NDI fréquents.
Pour reconstituer l'ensemble des itemsets fréquents à partir des NDI fréquents, on peut utiliser une approche ascendante. On commence par l'ensemble des NDI fréquents. Le niveau suivant sera constitué des itemsets X tels que tous les sous-ensembles soient dans l'ensemble des NDI fréquents. A chaque étape, on calcule les bornes inférieure (l) et supérieure (u) des supports des candidats. Si $l \neq u$, on sait que le candidat n'est pas fréquent. Si $l = u$, l'itemset est dérivable, et il est fréquent si $l \geq minsup$. Il faut peut-être souligner que les supports des NDI fréquents stockés seront utilisés dans ce calcul. Afin d'éviter l'évaluation de toutes les règles pour chaque itemset, les auteurs proposent une optimisation qui se résume comme suit : chaque fois que l'on rencontre un DI, on garde la régle qui a permis de trouver son support exact. Cela va permettre de choisir la règle à utiliser pour le calcul du support des sur-ensembles de cet itemset, au lieu d'évaluer toutes les règles.
Dans [19], les auteurs démontrent la proposition suivante qui fait la connexion entre les représentations condensées présentées dans les propositions 3.19 et 3.21.

Proposition 3.22 $FreqNDISup(\mathbb{D}) \subseteq FreqDFreeSup(\mathbb{D})$.

Les auteurs montrent dans [19] que leur approche est une généralisation des ensembles libres disjonctifs, et par conséquent des clés. Si on reprend le lemme 1 et la définition 3.9, on peut dire qu'un itemset X est un ensemble libre disjonctif si il n'existe pas deux items A et B dans X tels que $S(X) = S(X - \{A\}) + S(X - \{B\} - S(X - \{A, B\})$. On peut remarquer, comme les auteurs le montrent dans ([19]), que cette expression correspond à une de leurs règles déductives. Les ensembles libres disjonctifs, et par conséquent les ensembles libres peuvent ainsi être trouvées à partir des règles déductives.

3.3 Représentations approximatives

Définition 3.12 *Etant donné un ensemble de données $\mathbb{D}$, soit $X_1, ..., X_n$ et Y des sous-ensembles de $Sup(\mathbb{D})$. R est une représentation condensée approximative de Y si :*

1. *$X_1 \cup ... \cup X_n \subseteq Y$.*
2. *Il existe une fonction F indépendante de $\mathbb{D}$ permettant de calculer Y à partir de R à ϵ-prés, i.e. de calculer sans accéder aux données les motifs de Y et une approximation de leurs supports à ϵ-prés.*

Le concept d'ensembles $\delta - libres$ a été introduit par Boulicaut et al. ([15, 16]). La définition des $\delta - libres$ ($\delta-$free sets) proposée dans [16] passe par celle des règles $\delta - fortes$.

Définition 3.13 *Soit R un ensemble d'items. Une règle d'association basée sur R est une expression de la forme $X \Rightarrow Y$, où $X, Y \subseteq R$ et $X \cap Y = \emptyset$.*
Une règle $\delta - forte$ ($\delta - strong\ rule$) est une règle d'association $X \Rightarrow Y$, telle que $S(X) - S(X \cup Y) \leq \delta$, c'est à dire que la règle n'est pas violée dans plus de δ transactions.

Définition 3.14 *Un itemset X est appelé $\delta - libre$ s'il n'existe pas de règle $\delta - forte$ basée sur X dans $\mathbb{D}$. Notons $Free(\mathbb{D}, \delta)$ l'ensemble des $\delta - libres$.*

Puisque δ est supposé être très petit, un ensemble δ-libre est un itemset dont les sous-ensembles n'ont pas de forte corrération.
On peut noter que les ensembles δ-libres représentent une extension des clés, et plus précisément lorsque $\delta = 0$, on obtient une clé :

$$Free(\mathbb{D}, 0) = Key(\mathbb{D}).$$

Proposition 3.23 - Anti-monotonie. *Soit X un itemset. Pour tout $Y \subseteq X$, si $X \in Free(\mathbb{D}, \delta)$, alors $Y \in Free(\mathbb{D}, \delta)$.*

Notons $FreqFree_\delta(\mathbb{D})$ l'ensemble des $\delta - libres$ fréquents et $FreqFreeSup_\delta(\mathbb{D})$ l'ensemble des $\delta - libres$ avec leur support. $Bd^-(FreqFree_\delta(\mathbb{D})) = \{X \subseteq \mathbb{I} \mid X \notin FreqFree_\delta(\mathbb{D}) \wedge \forall Y \subset X : Y \in FreqFree_\delta(\mathbb{D})\}$ la frontière négative des $\delta - libres$.

Proposition 3.24 $\{ FreqFreeSup_\delta(\mathbb{D}), BdSup^-(FreqFree_\delta(\mathbb{D})) \}$ *est une représentation condensée approximative de $FreqSup(\mathbb{D}) \cup BdSup^-(FreqFree_\delta(\mathbb{D}))$.*

En utilisant cette représentation, l'approximation du support d'un itemset X se fait comme suit : Si X a un sous-ensemble Y libre mais non fréquent, alors le support de X est considéré comme égal à 0. Sinon, le support de X est égal au minimum des supports des sous-ensembles de X qui sont libres et fréquents.

3.4 Conclusion

Nous avons différentes représentations condensées des motifs fréquents, et notamment les représenations condensées exactes et les représentations condensées approximatives. Les représentations condensées exactes permettent de reconstituer l'ensemble des motifs fréquents avec leurs supports exacts, tandis que pour les représentations condensées approximatives, les supports sont regénérés avec une certaine erreur.
L'ensemble $\{ FreqMSup(\mathbb{D}) \}$ des maximaux fréquents, munis de leurs supports, ne constitue pas une représentation condensée de l'ensemble $FreqSup(\mathbb{D})$ des motifs fréquents munis de leurs supports . L'ensemble des fermés fréquents, munis de leurs supports, $\{ FreqCSup(\mathbb{D}) \}$ est une représentation condensée exacte $FreqSup(\mathbb{D})$, et on a la relation suivante entre l'ensemble des maximaux fréquents, des fermés fréquents et fréquents $Freq(\mathbb{D})$:

$$FreqM(\mathbb{D}) \subseteq FreqC(\mathbb{D}) \subseteq Freq(\mathbb{D}).$$

Par ailleurs, l'ensemble des fermés fréquents est de taille plus petite que l'ensemble des motifs clés fréquents. Il faut enfin rajouter à l'ensemble des motifs clés fréquents munis de leurs supports, sa frontière négative pour obtenir une représentation condensée exacte des motifs fréquents. De même, l'ensemble des ensembles libres disjonctifs munis de leurs supports, auquel on ajoute sa frontière négative, constitue une représentation condensée exacte des motifs fréquents.
L'ensemble des itemsets non dérivables, munis de leurs supports constitue également une représentation condensée de $FreqSup(\mathbb{D})$.

Les $\delta - libres$ qui constituent des représentations condensées approximatives sont des généralisations des itemsets libres (ou clés). Selon [50], le paramètre δ permet de faire un compromis entre

le niveau de précision souhaité (plus δ est petit, plus l'erreur est faible) et l'efficacité et la taille de la représentation (plus δ est grand et plus l'extraction est rapide et la représentation petite). Cependant toutes ces représentations sont des représentations de la réponse à une requête. Dans un environnement multi-utilisateurs, plusieurs utilisateurs exécutent des requêtes d'extraction. Comment alors trouver la représentation condensée d'un ensemble de réponses à des requêtes d'extraction ? La réponse à cette question que nous développerons dans la deuxième partie est une des contributions de cette thése.

4 Extraction itérative de motifs

4.1 Introduction

Le processus d'extraction de connaissances comprend plusieurs étapes : la compréhension du domaine, la préparation des données, la découverte de connaissances et leur exploitation. Il s'agit là d'un processus interactif et itératif complexe pour lequel plusieurs théories doivent être calculées [13]. Pour cela, il est nécessaire de disposer d'un langage de requêtes qui permet à l'utilisateur de sélectionner des sous-ensembles de données, mais aussi de spécifier différentes tâches d'extraction avec la sélection des motifs de la théorie correspondante. C'est ainsi qu'est apparu le concept de *Base de données inductive*, introduit pour la première fois par Mannila et al. [48] et ensuite repris dans [60, 61]. Intuitivement, une base de données inductive est une base de données classique à laquelle on associe un ensemble de motifs obtenus par extraction sur cette même base, et une fonction d'évaluation qui détermine la qualité des motifs sur les données. Les bases de données inductives proposent un cadre dans lequel pourrait se dérouler l'intégralité du processus d'extraction.
D'autre part, les requêtes d'extraction de ces motifs sont des requêtes complexes et prennent ainsi du temps à s'exécuter. Ce problème est d'autant plus crucial que l'utilisateur ne sachant pas à priori ce qu'il cherche, doit poser un certain nombre de requêtes pour trouver ce qui l'intéresse, d'où une itération du processus. Lorsqu'on se trouve dans un environnement multi-utilisateurs, où chaque utilisateur pose indépendamment ses requêtes, le problème se pose davantage. Etant donné que différents utilisateurs peuvent poser des requêtes similaires ou même identiques, il apparaît ainsi un besoin manifeste d'utiliser les résultats des extractions antérieures pour optimiser le calcul des extractions futures.
Cette partie commence ainsi par une présentation des bases de données inductives à la section 4.2. A la section 4.3, nous donnons quelques exemples de langages de bases de données inductives existants. La section 4.4 porte sur les techniques de réutilisation de résultats déjà calculés pour l'optimisation des extractions futures et la section 4.5 conclut cette partie.

4.2 Bases de données inductives

Définition 4.1 *Un schéma de base de données inductive (BDI) est un couple* $\mathcal{R} = (R, (\mathcal{L}, \varepsilon, \nu))$ *où :*

- R *est un schéma classique de base de données,*
- $\mathcal{L}$ *est un ensemble de motifs,*
- ν *est un ensemble de valeurs résultats,*
- $\varepsilon : (\theta, r) \rightarrow \nu$ *est une fonction d'évaluation des motifs sur les données, cette fonction associe à chaque paire* (θ, r) *un élément de* ν*, où* r *est une instance de* R *et* $\theta \in \mathcal{L}$ *est un motif.*

Une instance d'une base de données inductive (BDI) de schéma $(R, (\mathcal{L}, \varepsilon, \nu))$ est un couple (r, s) où r est une instance de R et $s \subseteq \mathcal{L}$. Dans le formalisme général de la découverte des motifs intéressants proposé par Mannila [63], une requête et son résultat peuvent être définis comme suit.
Soit $\mathcal{R}$ l'ensemble des instances possibles d'une base de données de schéma R. Considérons une instance r du schéma R d'une base de données, une collection de motifs $s \subseteq \mathcal{L}$ et une fonction d'évaluation ε de $\mathcal{L} \times \mathcal{R}$ dans $\{vrai, faux\}$. Une requête sur une BDI de schéma $(R, (\mathcal{L}, \varepsilon, \nu))$ est une fonction qui, à une instance (r_1, s_1) de schéma $(R, (\mathcal{L}, \varepsilon, \nu))$ associe une nouvelle instance

(r_2, s_2) de même schéma. Par exemple, la fonction $\tau(r_1, s_1) = (r_1, \mathcal{T}h(r_1, s_1, \epsilon))$, où :

$$\mathcal{T}h(r_1, s_1, \varepsilon) = \{\theta \in s_1 \mid \varepsilon(\theta, r_1) = vrai\}$$

est un exemple de requête sur une BDI qui effectue une sélection sur les motifs.

Exemple 4.1 *Considérons la relation de schéma* $R =< TID, A, B, C, D, E, F >$ *dont une instance* r *est donnée à la figure II.3 de la partie sur la découverte de motifs intéressants.* $\mathbb{I} = \{A, B, C, D, E, F\}$, TID *est l'identifiant de transaction et A,B,...,F sont des attributs binaires. Rappelons qu'un itemset est un sous ensemble de* $\mathbb{I}$ *et qu'une règle d'association est une implication de la forme* $X \Rightarrow Y$, *où* $X \subseteq \mathbb{I}$, $Y \subseteq \mathbb{I}$ *et* $X \cap Y = \emptyset$.
On peut alors considérer la base de données inductive suivante :

$$\mathcal{R} = (\{R\}, (\mathcal{L}, \varepsilon, \nu))$$

où :
- R *est le schéma indiqué plus haut,*
- $\mathcal{L}$ *est l'ensemble des règles d'associations,*
- $\nu = [0, 1]^2$,
- $\varepsilon(X \Rightarrow Y, r) = (sup(X \Rightarrow Y, r), conf(X \Rightarrow Y, r))$, *où sup et conf sont respectivement le support et la confiance de la règle* $X \Rightarrow Y$.

Dans [13], les auteurs proposent un cadre de recherche pour la conception de langages de requêtes pour les bases de données inductives. Leur proposition peut se résumer ainsi :
- les requêtes des bases de données inductives peuvent être formulées par une extension de l'algèbre relationnelle offrant la possibilité de faire référence aux motifs et aux valeurs correspondantes de la fonction d'évaluation ε ;
- le processus d'extraction de connaissances doit être vu comme un processus interactif et itératif dont le calcul nécessite la prise en compte de plusieurs théories et peut ainsi prendre du temps, ce qui implique un important problème d'optimisation.

L'utilisateur peut ainsi poser des requêtes sur les données comme sur une base de données classique, mais aussi sur les motifs, ou encore sur les deux. La possibilité doit ainsi être offerte de sélectionner des données relatives à certains motifs par exemple, et de ne pas recalculer des motifs déjà calculés. Il apparaît ainsi à nos yeux deux besoins fondamentaux :
- Besoin de langages de requêtes pour les bases de données inductives.
- Besoin de réutilisation des résultats stockés pour l'optimisation des calculs.

Nous allons ainsi d'une part, montrer par l'exemple les aspects interactifs et itératifs de quelques langages existants qui rentrent dans ce cadre et, d'autre part étudier quelques techniques de réutilisation des résultats antérieurs pour l'optimisation des calculs futurs.

4.3 Quelques exemples de langages existants

Les bases de données inductives ont besoin de langages qui permettent à l'utilisateur de sélectionner des sous-ensembles de données, mais aussi de spécifier des tâches d'extraction et de sélectionner des motifs à partir des théories correspondantes. Pour cela, ces langages doivent inclure différents critères qui prennent en compte toutes les étapes du processus d'extraction ([11], BBMM1) :
- La possibilité de sélectionner, de manipuler et d'interroger les données de la base.
- La spécification du type de motifs à extraire, étant donné qu'il existe différents types de motifs.

Cid	Cprof	Cadr
0	Avocat	Blois
1	Plombier	Orléans
2	Avocat	Orléans
3	Professeur	Blois
4	Plombier	Tours

Cid	Pid	qte
0	0	assez
0	1	beaucoup
1	0	peu
1	1	beaucoup
2	1	beaucoup
3	2	peu
4	1	assez

Pid	Ptype
0	Lait
1	Pain
2	Beurre
3	Fromage

FIG. II.20 – Base de données Ventes : Table Client(Cid,Cprof,Cadr) à gauche, table Vente(Cid,Pid) au milieu, table Produit(Pid,Ptype) à droite.

Cid	Cprof	Cadr	Ptype	qte
0	Avocat	Blois	Lait	assez
0	Avocat	Blois	Pain	beaucoup
1	Plombier	Orléans	Lait	peu
1	Plombier	Orléans	Pain	beaucoup
2	Avocat	Orléans	Pain	beaucoup
3	Professeur	Blois	Beurre	peu
4	Plombier	Tours	Pain	assez

FIG. II.21 – Table Achat

- La définition de contraintes que les motifs extraits doivent satisfaire. Il peut s'agir de contrainte de support minimum ou de contrainte sur les éléments de la règle.
- La satisfaction de la propriété de fermeture : étant donné qu'une instance d'une BDI est un couple (r, s), où r est une instance du schéma R et s un sous-ensemble de l'ensemble des motifs, le résultat d'une requête sur cette base devrait être un objet de même type que ses arguments, c.a.d un couple (r', s'). Cependant les langages présentés par la suite distinguent en fait plusieurs types de requêtes sur les BDI :
  - des requêtes d'extraction portant uniquement sur les données, et dont la réponse est un nouvel ensemble de motifs,
  - des requêtes sur des ensembles de motifs, et dont la réponse est un ensemble de motifs,
  - des requêtes sur des couples (r, s) et dont le résultat est un ensemble de données.
- Un post-traitement des résultats, qui permettrait d'effectuer des opérations sur les motifs, de sélectionner la partie des données pour lesquelles les motifs sont intéressants.

Il existe un certain nombre de langages que l'on présente comme langages de bases de données inductive. On peut citer notamment les langages MSQL [49], MINE-RULE [65, 66] qui extraient principalement des règles d'association, le langage MineSQL [69] qui extrait des itemsets fréquents et des règles d'associations, et le langage DMQL [45] qui prend en compte plusieurs types de motifs et de modèles. Nous allons voir par quelques exemples, dans quelle mesure les langages MSQL, MINE-RULE et DMQL satisfont ces critères. Le langage MineSQL sera présenté plus loin.

On considère les informations stockées dans la base de données $Ventes$ (figure II.20) composée des trois tables $Client$, $Vente$ et $Produit$. L'extraction se fait sur la table $Achat$, jointure de ces trois tables, présentée à la figure II.21. On s'intéresse aux règles d'association entre la profession des clients et le type de produits qu'ils achètent.

4.3.1 MSQL

Dans [48], les auteurs soulignent le besoin urgent, du fait du nombre important de règles d'associations générées par les systèmes d'extraction de connaissances, de mettre au point un langage qui permettrait aussi bien de trouver des règles sur une partie des données que de poser des requêtes sur la base de règles générée. Le langage MSQL propose plusieurs primitives parmi lesquelles : **GETRULES** pour la génération des règles et **SELECTRULES** pour la sélection des règles. Les auteurs [49] parlent respectivement de *data-mining* et de *rule-mining*. Avant de donner un exemple qui montre l'aspect itératif et interactif de ce langage, nous allons voir quelques définitions de base. Un descripteur est, comme nous l'avons indiqué dans la partie sur la découverte de motifs intéressants, une propriété $A = a$, où A est un attribut et $a \in dom(A)$.

Définition 4.2 *On dit qu'un tuple d'une relation R satisfait un descripteur $A = a$ si la valeur de A dans le tuple est égale à a. Un tuple satisfait une conjonction de descripteurs, s'il satisfait tous les descripteurs qui composent la conjonction.*

Regardons maintenant la forme des règles que les auteurs appellent *règles propositionnelles*. Une règle propositionnelle est une règle d'association telle que définie dans la partie sur la découverte de motifs intéressants, c'est à dire de la forme :

$$Body \Rightarrow Consequent \qquad [support, confiance]$$

où $Body$ (le corps) est une conjonction de descripteurs, et $Consequent$ (la tête) est un descripteur. Il faut souligner que le support est défini ici comme le nombre de tuples qui satisfont le corps de la régle. La confiance est le rapport entre le nombre de tuples qui satisfont le corps et la tête, et le nombre de tuples qui satisfont le corps.

Définition 4.3 *On dit qu'un tuple satisfait la règle $X \Rightarrow Y$ s'il satisfait $X \wedge Y$, et il viole la règle $X \Rightarrow Y$ s'il satisfait X mais pas Y.*

Selon les auteurs, les principales caractéristiques de MSQL se résument comme suit :
- MSQL est un langage qui prend sa source dans SQL.
- Le langage comprend une primitive qui permet de manipuler le résultat des extractions précédentes.
- Il y a une primitive qui permet d'identifier l'ensemble des données qui satisfont ou qui violent un ensemble de règles.

Nous allons voir maintenant des exemples qui mettent en relief ces différentes caractéristiques. Les auteurs soulignent qu'un des points les plus importants lors de la conception du langage est de permettre la représentation et la manipulation des régles, notamment le corps et la tête, qui sont des ensembles et ne peuvent pas être représentés en SQL standard. Une approche courante pour l'extraction de règles est d'exécuter la primitive $GETRULES$ qui permet de trouver toutes les règles, de stocker le résultat, et ensuite d'exécuter une série de $SELECTRULES$, pour sélectionner des sous-ensembles de la base de règles.
Le premier problème qui se pose est le suivant : la primitive $GETRULES$ s'attend à recevoir en entrée une table dans laquelle l'attribut qui sert de référentiel (Cid, dans notre cas) est une clef. Pour cela, il faut créer une table avec autant d'attributs booléens qu'il y a de valeurs pour les attributs $Cprof$ et $Ptype$. On obtient la table $Achat1$ à la figure II.22.

Cette transformation met en évidence une faiblesse importante de MSQL. La table sur laquelle s'effectue l'extraction aura autant d'attributs qu'il y a de valeurs d'attributs possibles dans le corps et la tête de la règle. La géneration des règles avec $GETRULES$ se fait comme suit :

Cid	Avocat	Plombier	Professeur	Lait	Pain	Beurre
t1	1	0	0	1	0	0
t2	1	0	0	0	1	0
t3	0	1	0	1	0	0
t4	0	1	0	0	1	0
t5	1	0	0	0	1	0
t6	0	0	1	0	0	1
t7	0	1	0	0	1	0

FIG. II.22 – Table *Achat*1

Corps	Tête	Support	Confiance
(Avocat=1)	(Ptype=Pain)	3	66 %
(Plombier=1)	(Ptype=Pain)	3	66 %

FIG. II.23 – Base de règles *BaseR*

GETRULES(*Achat*1) into BaseR
WHERE BODY has {(Avocat=1) OR (Plombier=1) OR (Professeur=1) OR (Electricien=1) } AND CONSEQUENT is {(Lait=1) OR (Beurre=1) OR (Pain=1) } AND support > 2 AND confidence > 0.5

Cette requête permet de trouver à partir de la table *Achat*1 toutes les règles de la forme $Cprof = a \Rightarrow Ptype = b$ qui satisfont les critères de support et de confiance, et de les stocker dans la base de règles *BaseR*. Cette base de règles est représentée à la figure II.23.
La primitive $SELECTRULES$ permet de manipuler le résultat des extractions précédentes. La requête suivante permet d'extraire les règles dont le corps contient *Plombier* :

SELECTRULES(BaseR)
WHERE BODY has (Plombier=1).

MSQL propose aussi une primitive $VIOLATES$ qui permet de trouver des exceptions sur les données d'origine. La requête suivante permet de sélectionner l'ensemble des tuples de *Achat*1 qui violent toutes les règles extraites ayant $Avocat = 1$ dans le corps (voir figure II.24) :

INSERT INTO *Achat*2 AS
SELECT * FROM *Achat*1
WHERE VIOLATES ALL (
SELECTRULES(BaseR) WHERE BODY has (Avocat=1))

MSQL propose aussi une primitive $SATISFY$ qui permet de trouver des tuples qui satisfont un ensemble de règles. Ces deux dernières primitives sont trés intéressantes puisqu'elles permettent de faire le lien entre les règles trouvées et les données d'origine.

Cid	Avocat	Plombier	Professeur	Lait	Pain	Beurre
t1	1	0	0	1	0	0
t3	0	1	0	1	0	0
t4	0	1	0	0	1	0
t6	0	0	1	0	0	1
t7	0	1	0	0	1	0

FIG. II.24 – Table Achat2

4.3.2 MINE RULE

L'autre extension de SQL, présenté dans [65, 66], propose un opérateur MINE-RULE pour la génération de règles d'associations. L'objectif des auteurs est ici de présenter un modèle pour la description uniforme des différentes règles d'association proposées jusque-là. Ils reformulent ainsi par l'intermédiaire de cet opérateur différents problèmes liés à l'extraction de règles d'associations telles que les règles d'associations généralisées [84, 85] et les motifs séquentiels [3]. Ainsi, les règles extraites par MINE RULE sont plus expressives que celles de MSQL. Dans MSQL, le référentiel est forcément la clef de la table sur laquelle s'effectue l'extraction. Cependant, dans le cas général, les données peuvent être regroupées par un autre attribut précisant le sujet de l'extraction ou référenciel (c.f partie sur la découverte de motifs intéressants). Les règles d'association sont de la forme $X \Rightarrow Y$, où X (le corps) et Y (la tête) sont des ensembles de descripteurs.
Selon les auteurs, les principales caractéristiques de MINE RULE se résument comme suit :

- La possibilité de sélectionner une partie des données pour le processus d'extraction. D'une part, MINE RULE dispose de la clause FROM qui permet de spécifier la source de données. D'autre part, étant donné que le WHERE associé à la clause FROM est exécutée avant le GROUP BY, on peut dire que c'est le couple (GROUP BY, WHERE) qui constitue le référentiel et précise ainsi le sujet de l'extraction.
- La définition de la structure des règles. Les auteurs parlent de règles d'association mono-dimensionnelles (lorsque les composantes de la règle sont différentes valeurs de la même dimension ou attribut) ou de règles multi-dimensionnelles (lorsque différents attributs interviennent dans la règle).
- La définition de différentes contraintes liées à l'extraction.

Nous allons voir maintenant quelques exemples avec MINE RULE. La requête suivante (Requête m1) permet de trouver les règles d'association entre les avocats et les types de produits qu'ils achètent :

```
MINE RULE TableR AS
SELECT DISTINCT 1..1 Cprof AS BODY,
1..1 Ptype AS HEAD, SUPPORT, CONFIDENCE
FROM Achat WHERE Cprof = Avocat
GROUP BY Cid
EXTRACTING RULES WITH SUPPORT : 0.1, CONFIDENCE : 0.3
```

La clause SELECT définit la structure des règles : le corps est une profession (1..1), et la tête un type de produit. La clause FROM définit la source de données. Ici, elle précise qu'il faut s'intéresser uniquement aux achats des avocats. Il faut signaler que dans MSQL, une telle clause n'existe pas. C'est ainsi que la sélection d'une source de données n'est pas prise en compte par MSQL et doit être traitée à part. Les clauses GROUP BY et WHERE indiquent que le référen-

tiel est l'attribut Cid, et ne seront pris en compte que les identifiants des avocats. Finalement la clause EXTRACTING RULES WITH indique que seules les règles avec un support et une confiance respectivement supérieurs à 0.1 et 0.3 seront retenues. Cet exemple extrait des règles d'associations multi-dimensionnelles. La requête suivante est un exemple d'extraction de règles mono-dimensionnelles :

```
MINE RULE IntraType AS
SELECT DISTINCT 1..n Ptype AS BODY,
1..1 Ptype AS HEAD, SUPPORT, CONFIDENCE
FROM Achat
GROUP BY Cid
EXTRACTING RULES WITH SUPPORT : 0.1, CONFIDENCE : 0.2.
```

La clause SELECT précise qu'on peut avoir plusieurs types (1..n) de produits dans le corps et un type pour la tête. Ainsi, on pourrait avoir ici des règles du genre $Ptype = Pain, Ptype = Lait \Rightarrow Ptype = Beurre$. On peut aussi faire une sélection sur la source de données en utilisant la clause HAVING de SQL associée à la clause GROUP BY. La clause HAVING est utilisée avec les fonctions d'aggrégats (telles que COUNT, MIN, MAX, AVG). Ainsi la requête suivante n'extrait que les règles concernant les consommateurs ayant effectué au plus 5 achats :

```
MINE RULE MoinsDeCinq AS
SELECT DISTINCT 1..n Ptype AS BODY,
1..1 Ptype AS HEAD, SUPPORT, CONFIDENCE
FROM Achat
GROUP BY Cid
HAVING COUNT(*) <= 5
EXTRACTING RULES WITH SUPPORT : 0.1, CONFIDENCE : 0.2
```

Il est possible, avec MINE RULE de partionner les groupes de tuples définis via le GROUP BY en sous-groupes appelés *clusters* et de trouver des règles d'association entre *clusters* d'un même groupe. MINE RULE propose pour cela la clause CLUSTER BY. Pour plus de détails sur cette clause, le lecteur peut se référer à [65]. Comme MSQL, MINE RULE permet de spécifier différentes contraintes sur les composantes de la règle (le corps, la tête ou sur leur cardinalité). Il faut cependant souligner que MINE RULE a été conçu en prenant en compte principalement la phase d'extraction. Ainsi, les auteurs ne proposent pas de façon explicite d'outils pour le post-traitement des résultats de l'extraction. Cependant, dans [11], les auteurs montrent comment stocker l'ensemble des règles trouvées lors de l'extraction dans trois tables : une pour le corps, une pour la tête et une pour faire le lien entre le corps et la tête. A partir de ces tables, ils montrent enfin quelles requêtes SQL permettent de trouver des exceptions sur les données source, et notamment comment trouver l'ensemble des données qui violent des règles précises.

4.3.3 DMQL

Le langage DMQL a été proposé par l'équipe de Jiawei Han à l'Université de Simon Fraser [44, 45]. Les auteurs insistent sur le besoin pour le data mining de disposer de langages de requêtes. Ils soulignent que les systèmes d'extraction de connaissances doivent être interactifs et itératifs pour permettre une découverte efficace et une manipulation flexible des motifs. La conception des langages de requêtes doit impérativement répondre à un tel besoin. Pour cela, ils proposent un certain nombre de primitives qui doivent être au cœur de ces langages de requêtes,

primitives qui sont spécifiées comme suit :

- L'ensemble des données relatives au processus d'extraction doit être spécifié dans la requête d'extraction. Etant donné qu'un utilisateur peut être intéressé seulement par une partie des données, la possibilité doit lui être offerte de ne travailler que sur ces données là. Il faut noter que ce critère est le même que celui proposé pour les langages dans le cas des bases de données inductives.
- Le type de motifs à rechercher doit être spécifié dans la requête d'extraction. Etant donné que l'utilisateur peut s'intéresser à des règles d'associations ou de classification, ou encore à des corrélations ou à d'autres modèles, il est important de préciser le type de motifs que l'on recherche.
- L'intégration de la connaissance du domaine. L'intégration dans le processus d'extraction de la connaissance du domaine lorsqu'elle est disponible, et surtout lorsqu'elle aide à trouver des motifs intéressants, est importante. Cette connaissance du domaine peut être donnée sous la forme d'une hiérarchie de concepts, utilisée pour représenter les attributs ou leurs valeurs en différents niveaux d'abstraction.
- La mise à disposition de différentes mesures et seuils pour l'extraction des motifs intéressants. Il s'agit, par exemple dans le cas des règles d'association, du support et de la confiance.

Le langage DMQL a été conçu sur la base de ces diffèrentes primitives. La syntaxe du langage est proche de celle de SQL. C'est ainsi que l'on peut retrouver les traditionnelles clauses telles que FROM, WHERE, GROUP BY, HAVING, ORDER BY.
Une règle d'association est de la forme $A \Rightarrow B$, où A et B sont des conjonctions de prédicats. On peut citer comme exemple dans notre table *Achat* la règle suivante : $Cprof(X, Avocat) \wedge Ptype(Y, Pain) \Rightarrow qte(X, Y, beaucoup)$. Cette règle veut dire que les avocats qui achètent du pain en achètent beaucoup. Il faut signaler que cette syntaxe ne fait pas apparaître de manière explicite le référentiel de la règle (s'agit-il d'une règle sur les consommateurs ou sur les produits ?) Si on considère le formalisme défini par [35], une telle règle peut être réecrite comme suit :

$$(\forall X)[(\exists Y, Z)(Client(X, Avocat, Z) \wedge Prod(Y, Pain)) \Rightarrow (\exists Y)(Vente(X, Y,' beaucoup'))]$$

Les auteurs de DMQL définissent deux types de règles qu'ils appellent règles mono-dimensionnelles et règles multi-dimensionnelles. Une règle mono-dimensionnelle contient plusieurs occurrences du même prédicat alors que dans une règle multi-dimensionnelle, plusieurs prédicats apparaissent, et chacun une seule fois. Cependant, la syntaxe de DMQL prévoit d'avoir des règles dans lesquelles apparaissent plusieurs instances d'un même prédicat en ajoutant un signe + au nom des prédicats devant apparaître plusieurs fois. DMQL permet également de guider le processus d'extraction par l'utilisation de méta-règles. Les méta-règles sont des modèles qui permettent de restreindre de façon syntaxique l'espace de recherche des règles d'associations. Par exemple, si on s'intéresse aux relations entre la profession, le type de produits achetés et la quantité, on peut définir la méta-règle suivante :

$$Cprof(X : Client, Y) \wedge P(Y, W) \Rightarrow Q(X : Client, Y : Produit, Z)$$

où P et Q sont des variables prédicats qui peuvent être instanciées par les attributs des relations impliquées. X est une clé de la table $Client$, Y une clé de la table $Produit$, W et Z sont des variables. Une règle que l'on peut trouver avec cette méta-règle est celle que nous avons donnée comme exemple. Nous allons maintenant donner des exemples de requêtes DMQL à partir de la table *Achat*. Le premier exemple est la requête qui utilise la méta-règle définie ci dessus :

USE DATABASE Ventes

MINE ASSOCIATIONS AS BASER1
MATCHING WITH $Cprof(X : Client, Y) \wedge P(Y, W) \Rightarrow Q(X : Client, Y : Produit, Z)$
FROM *Achat*
GROUP BY Cid
WITH SUPPORT threshold=30%
WITH CONFIDENCE threshold=30%

La clause USE indique la base de données utilisée. Le type de motifs recherchés est spécifié dans la clause MINE. Ici, il s'agit de règles d'associations. Avec le mot-clé MATCHING, on indique la forme des règles que l'on veut obtenir. Les clauses FROM et GROUP BY indiquent respectivement la ou les relations sources et le référentiel. Une autre requête, qui permet de trouver des relations entre les différents produits, est la suivante :

USE DATABASE Ventes
MINE ASSOCIATIONS AS BASER2
MATCHING WITH $Ptype^{+}(X, Y) \Rightarrow Ptype^{+}(X, Z)$
FROM *Achat*
GROUP Cid
WITH SUPPORT threshold=30%
WITH CONFIDENCE threshold=30%

Dans ce cas, on peut avoir plusieurs occurrences du prédicat *Ptype*, aussi bien dans le corps que dans la tête. Notons qu'il n'existe pas à notre connaissance de description complète du langage DMQL et de sa sémantique, ne serait-ce que pour les règles d'association.

4.3.4 Discussion

Nous passons en revue les différents critères que doivent prendre en compte les langages de requêtes de bases de données inductive afin de voir dans quelle mesure ils sont satisfaits par les langages que nous venons de présenter.
Le premier critère est la possibilité de sélectionner, de manipuler et d'interroger les données. Le langage MSQL n'intégre pas la possibilité de sélectionner une partie des données. Pour travailler sur une partie des données, il faut exécuter une requête SQL et utiliser le résultat comme entrée de MSQL. Par contre, les langages MINE RULE et DMQL disposent des clauses FROM et WHERE qui leur permettent de faire cette sélection.
Le deuxième critère est la spécification du type de motifs à extraire. Le langage DMQL est le seul qui prend véritablement en compte différents types de motifs, même si MINE RULE permet de manipuler en plus des règles d'associations, des motifs séquentiels élémentaires.
Le troisième critère est la définition de contraintes que les motifs doivent satisfaire. Les trois langages intégrent la définition de différentes contraintes, telles que le support, la confiance, des contraintes sur le corps ou la tête de la règle.
Le quatrième critère est la satisfaction de la propriété de fermeture. C'est le langage MSQL qui dispose de primitives dont les propriétés s'approchent le plus de la propriété de fermeture. En effet, ses primitives $SATISFY$ et $VIOLATE$ permettent de faire le lien entre les motifs trouvés et les données d'origine. MINE RULE ne dispose pas de telles primitives, mais on peut, comme nous l'avons déjà dit, par des requêtes complexes, trouver par exemple des exceptions sur les données sources. Le cinquième critère est le post-traitement des résultats. Seul MSQL propose une primitive pour le post-traitement des résultats.
Finalement, il faut souligner que dans MSQL, le référentiel est forcément la clef de la table sur

laquelle s'effectue l'extraction. Ceci n'est pas le cas pour MINE RULE et DMQL, même si la syntaxe de DMQL ne définit pas de manière explicite le référentiel.

4.4 Techniques de réutilisation de résultats déjà calculés.

Un des sérieux problèmes qui se pose à l'extraction des règles d'association et à ses applications est le temps de réponse requis par les algorithmes d'extraction. Celui-ci est trop élevé pour des systèmes qui, de par leur conception et leur nature, se veulent interactifs. Dans le but de réduire ce temps de réponse, un certain nombre de techniques ont été proposées. L'idée fondamentale qui est à la base de ces techniques est d'utiliser les résultats déjà calculés et stockés pour optimiser les calculs futurs. Parmi ces techniques, deux approches se distinguent. La première consiste, étant donné deux requêtes, à identifier une relation entre elles, et selon cette relation à proposer un algorithme de calcul de la deuxième en fonction de la première. La seconde approche consiste à ajouter au système conventionnel d'extraction un cache qui garde certains motifs, ce qui évite de les recalculer lors d'une prochaine extraction.

4.4.1 Techniques basées sur les relations entre les requêtes d'extraction.

Nous allons présenter deux approches [5, 69] qui rentrent dans ce cadre. Pour les deux, nous verrons comment les auteurs définissent une requête d'extraction. Ensuite nous étudierons les relations entre les requêtes, et finalement nous verrons l'algorithme proposé.

Première technique de comparaison

Dans [5], E. Baralis et G. Psaila expriment leurs requêtes d'extraction par l'opérateur MINE-RULE que nous avons vu à la sous-section 4.3.2. Ainsi, les différents exemples que nous avons vus sont des requêtes d'extraction. Le résultat d'une requête d'extraction est un ensemble de règles avec pour chaque règle, un support et une confiance.

Relations entre les requêtes

Les auteurs identifient trois types de relations entre les requêtes :

- **Equivalence** : deux requêtes d'extraction sont équivalentes si les règles obtenues par les résultats de ces requêtes sont les mêmes, avec les mêmes valeurs de support et de confiance, pour toute instance de la base de données. Soit la requête suivante que nous appelerons $Q1$:

  MINE RULE TableR AS
  SELECT DISTINCT 1..1 Cprof AS BODY,
  Ptype AS HEAD, SUPPORT, CONFIDENCE
  FROM Achat WHERE Cprof = Avocat
  GROUP BY Cid
  EXTRACTING RULES WITH SUPPORT : 0.1, CONFIDENCE : 0.3

  On peur créer une autre table qui est le résultat de la sélection sur les avocats de la table *Achat* et appliquer l'opérateur MINE RULE à cette table. Cette requête est équivalente à la requête $Q1$.
- **Inclusion** : une requête d'extraction Q_1 inclut une requête d'extraction Q_2, noté $Q_1 \supseteq Q_2$ si, pour toute instance de la base de données, toute règle présente dans le résultat de Q_2 est aussi présente dans le résultat de Q_1 avec les mêmes valeurs de support et de confiance.

Soit la requête $Q3$ suivante :

MINE RULE TableR AS
SELECT DISTINCT 1..1 Cprof AS BODY,
Ptype AS HEAD, SUPPORT, CONFIDENCE
FROM Achat WHERE Cprof = Avocat
GROUP BY Cid
EXTRACTING RULES WITH SUPPORT : 0.4, CONFIDENCE : 0.5

Alors, on a : $Q1 \supseteq Q3$. On remarquera qu'il existe une relation entre l'inclusion et l'équivalence : $Q1 \equiv Q2$ ssi $Q1 \subseteq Q2$ et $Q2 \subseteq Q1$.

– **Dominance** : une requête d'extraction Q_1 domine une requête Q_2, noté $Q_1 \rhd Q_2$ si, pour toute instance de la base de données, chaque règle présente dans le résultat de Q_2 avec un support s_2 et une confiance c_2 est aussi présente dans le résultat de Q_1 avec $s_2 \leq s_1$ et $c_2 \leq c_1$.

Ensuite, les auteurs montrent comment trouver la relation d'inclusion ou de dominance entre deux requêtes. Pour cela, ils procèdent à un raffinement de requêtes en modifiant les caractéristiques de bases des règles d'associations, i.e. les seuils de support et de confiance, les cardinalités du corps et de la tête, et différentes contraintes de sélection. Le théorème suivant permet par exemple de déterminer la relation d'inclusion entre deux requêtes d'extraction.

Théoreme 4.1 *Soit Q_1 et Q_2 deux requêtes d'extraction, qui différent seulement par leurs seuils minimum de supports (resp. $minsup_1, minsup_2$) et de confiances (resp. $minconf_1, minconf_2$). Si $minsup_1 \leq minsup_2$ et $minconf_1 \leq minconf_2$, alors $Q_1 \supseteq Q_2$.*

Lorsque $Q_1 \supseteq Q_2$, le résultat de la deuxième requête peut être calculé par filtrage du résultat de la première.

Deuxieme technique de comparaison

La deuxième approche basée sur les relations entre les requêtes a été proposée dans [69]. Les auteurs introduisent les vues sur des requêtes d'extraction auxquelles ils veulent faire jouer le même rôle que les vues pour les systèmes de gestion de bases de données. Ils proposent d'utiliser les vues pour l'optimisation de la réponse à une requête d'extraction "similaire" à la requête définie par la vue. La comparaison entre les requêtes permet de détecter cette similarité.
Avant de donner la définition de vues sur des requêtes d'extraction, revenons sur la notion de vues dans les bases de données classiques. Une vue est une table virtuelle ou matérialisée qui stocke les résultats d'une requête SQL. Les vues sont utilisées pour faciliter l'accès à des données fréquemment utilisées qui sont le résultat de requêtes complexes. Une vue sur une requête d'extraction est une collection de motifs (règles d'associations et itemsets fréquents dans le cas présent) qui est le résultat de la requête.
Pour définir une requête d'extraction, les auteurs proposent un langage déclaratif MineSQL, une extension de SQL pour l'extraction de règles d'association. A la différence de MINE RULE, MineSQL permet la découverte et la manipulation des itemsets fréquents. Le langage MineSQL définit ainsi un ensemble de nouveaux types pour le stockage et la manipulation des règles d'associations et des itemsets. SET OF (SET OF NUMBER, SET OF CHAR, etc.) permet de représenter des ensembles d'items. La fonction SET permet de convertir de simples items en ensembles d'items.
Le type ITEMSET OF est utilisé pour représenter les itemsets fréquents. Le support d'un itemset

est stocké en même temps que les items qui le composent. SUPPORT(x) retourne le support de l'itemset x.
MineSQL prend aussi en compte des règles. Le type RULE OF permet de représenter des règles d'associations. Les auteurs définissent un certain nombre de fonctions et d'opérateurs qui manipulent les composantes des règles. L'extraction des itemsets fréquents ou des règles d'associations est effectuée par la clause MINE. Ainsi, la requête suivante extrait de la table *Achat* les itemsets fréquents, i.e. dont le support est supérieur ou égal à 0.1.

MINE ITEMSET, SUPPORT(ITEMSET)
FOR X FROM (SELECT SET(Ptype) AS X
FROM Achat GROUP BY Cid)
WHERE SUPPORT(ITEMSET) ≥ 0.1 ;

La primitive CREATE VIEW permet de définir une vue sur une requête d'extraction. L'exemple suivant crée la vue sur la requête précédente.

CREATE VIEW $VUE1$
AS MINE ITEMSET
FOR X FROM (SELECT SET(Ptype) AS X FROM Achat GROUP BY Cid)
WHERE SUPPORT(ITEMSET) ≥ 0.1 ;

Supposons maintenant qu'un utilisateur s'intéresse à l'ensemble des motifs définis par la requête suivante :

MINE ITEMSET
FOR X FROM (SELECT SET(Ptype) AS X FROM Achat GROUP BY Cid)
WHERE SUPPORT(ITEMSET) ≥ 0.3 ;

Le résultat de cette requête peut être calculé en effectuant simplement un filtrage du résultat de la vue. Cet exemple montre que l'on peut utiliser une vue pour optimiser le calcul de la réponse à une requête. Il faut, bien sûr, pour cela, pouvoir comparer des requêtes. Les auteurs, comme dans [5], comparent les requêtes en se basant sur les contraintes définies sur celles-ci. C'est ainsi qu'ils définissent deux classes de contraintes :

- **les contraintes sur les données** et,
- **les contraintes d'extraction**.

Les contraintes sur les données se trouvent dans la clause FROM de l'instruction SELECT. Elles représentent des conditions de sélection sur la source de données à traiter. Cela veut dire que dans ce cas, l'extraction se fait sur une table différente de la table de départ, et ainsi les valeurs des supports des motifs changent.
Les contraintes d'extraction se retrouvent dans la clause WHERE. Elles représentent des conditions de sélection sur les motifs à découvrir. Dans ce cas, les valeurs des supports ne changent pas.
Suivant ces types de contraintes, les auteurs identifient différentes relations entre la requête qui définit une vue et une nouvelle qui aurait des contraintes plus restrictives ou moins restrictives.

Relations entre les requêtes

Quatre relations sont définies entre la requête qui définit la vue et la nouvelle requête.

– La nouvelle requête a des contraintes sur les données plus rectrictives que la requête définissant la vue ; on parle alors d'extension de contraintes sur les données ;
Exemple 4.2 *Dans La requête $Q1$ suivante, on ne s'intéresse qu'aux consommateurs qui ont acheté plus de 5 articles. Elle étend les contraintes sur les données de la requête $Q2$:*

$Q1$:	$Q2$:
MINE ITEMSET	*MINE ITEMSET*
FOR X	*FOR X*
FROM (SELECT SET(Ptype) AS X	*FROM (SELECT SET(Ptype) AS X*
FROM Achat	*FROM Achat*
GROUP BY Cid	*GROUP BY Cid)*
HAVING COUNT(Ptype) ≥ 5*)*	
WHERE SUPPORT(ITEMSET) $\geq$ *0.3*	*WHERE SUPPORT(ITEMSET)* $\geq$ *0.3*

Intuitivement, l'extension des contraintes sur les données correspond à une réduction de l'ensemble de données à traiter.
– La nouvelle requête a des contraintes sur les données moins rectrictives que la requête définissant la vue. On parle de réduction des contraintes sur les données : si une requête $Q1$ étend les contraintes sur les données d'une requête $Q2$, alors $Q2$ réduit les contraintes de $Q1$. Intuitivement, la réduction des contraintes sur les données correspond à une augmentation de l'ensemble de données à traiter.
On notera que dans les cas de l'extension comme de la réduction de contraintes, étant donné que la découverte de motifs d'une requête à l'autre se fait sur des tables différentes, il est difficile de trouver une relation entre les deux requêtes.
– Les contraintes d'extraction de la nouvelle requête sont plus rectrictives que celles de la requête définissant la vue. On parle d'extension de contraintes d'extraction.
Exemple 4.3 *La requête $Q1$ suivante étend les contraintes d'extraction de la requête $Q2$:*

$Q1$:	$Q2$:
MINE ITEMSET	*MINE ITEMSET*
FOR X	*FOR X*
FROM (SELECT SET(Ptype) AS X	*FROM (SELECT SET(Ptype) AS X*
FROM Achat	*FROM Achat*
GROUP BY Cid	*GROUP BY Cid*
WHERE SUPPORT(ITEMSET) $\geq$ *0.3*	*WHERE SUPPORT(ITEMSET)* $\geq$ *0.1*

Notons que dans ce cas, le résultat (ou la réponse) de $Q1$ est inclus dans celui de $Q2$. On peut ainsi dire que si une requête $Q1$ étend les contraintes d'extraction de $Q2$, alors $Q1 \subseteq Q2$. Intuitivement, l'extension des contraintes d'extraction correspond à une réduction de l'ensemble des motifs découverts.
– Les contraintes d'extraction de la nouvelle requête sont moins rectrictives que celles de la requête définissant la vue. On parle de réduction de contraintes d'extraction. Si $Q1$ étend les contraintes d'extraction de $Q2$, alors $Q2$ réduit les contraintes d'extraction de $Q1$.

Algorithme

Les auteurs ne proposent pas de nouvel algorithme mais montrent plutôt quel algorithme existant utiliser selon la relation entre la nouvelle requête et celle qui définit la vue :
– Extraction complète lorsqu'il n'y a pas de relations.
– Extraction incrémentale si la source de données de la nouvelle requête contient des tuples supplémentaires par rapport à celle utilisée dans la vue. Il existe un certain nombre d'al-

gorithmes que l'on appelle algorithmes incrémentaux (Incremental mining) qui permettent de faire une mise à jour de l'ensemble des motifs découverts lorsque la base de données change. Un exemple est l'algorithme [31]. Ce type d'algorithme peut être utilisé lorsqu'il y a réduction des contraintes de base de données.
- L'extraction complémentaire consiste à découvrir des itemsets fréquents en se basant sur ceux présents dans la vue, qui restent fréquents. Les auteurs proposent d'utiliser l'agorithme de [72] (que nous allons présenter ci-dessous) dans le cas où il y a réduction de contraintes d'extraction.
- Filtrage dans le cas d'extension des contraintes d'extraction.

Les auteurs montrent finalement que s'il existe deux relations (une sur les contraintes sur les données et une autre sur les contraintes d'extraction) entre la nouvelle requête et celle qui définit la vue, alors l'extraction peut se faire en deux étapes, où chaque étape utilise l'algorithme correspondant.

Techniques basées sur l'utilisation de caches

Les approches [72, 51] que nous présentons dans cette partie ajoutent au système conventionnel de découverte de motifs un cache pour l'accéleration du processus d'extraction. Nous étudierons le contenu du cache et les algorithmes proposés.

Première technique de cache : knowledge cache

Dans le but d'optimiser l'extraction de règles d'association, Nag et al. [72] proposent d'ajouter au système conventionnel d'extraction un cache appelé *knowledge cache*. Précisons qu'il s'agit du système standard d'extraction de règles d'association qui, rappelons le, passe par la découverte des itemsets fréquents.

Contenu du cache

Dans cette approche, pour toute nouvelle requête d'extraction calculée, on ajoute dans le cache tous les itemsets dont on a calculé le support, qu'ils soient fréquents ou non (en général les itemsets non fréquents dont on a calculé le support sont les itemsets de la frontière négative). Notons que le fait d'ajouter au cache des itemsets non fréquents pour une extraction donnée peut être utile pour deux raisons :
- Ces itemsets peuvent être fréquents relativement à une nouvelle requête d'extraction (par exemple, si le seuil est plus petit).
- Pour ces itemsets, il n'est pas besoin de recalculer le support.

Le cache est un ensemble de buffers (buckets), où le k-ième buffer stocke seulement les k-itemsets sous la forme d'enregistrements : $< Itemset, Support >$.

Algorithmes

Les auteurs proposent trois algorithmes de cache. L'algorithme le plus simple est **l'algorithme NR(No Replacement)**. L'utilisation du cache se fait avec l'algorithme A-priori. D'abord la phase de génération au niveau k permet de créer les k-itemsets candidats, ensuite le cache est utilisé pour trouver le support de tous les candidats qui s'y trouvent. Si tous les supports sont trouvés, alors il n'y aura pas d'accès à la base pour ce niveau, ce qui représente un gain de temps considérable. Sinon, l'algorithme construit un arbre de hachage des itemsets dont le support est

inconnu, fait une passe sur la base pour le calcul des supports et ne retient pour la génération suivante que les itemsets dont le support est supérieur ou égal au seuil. L'algoritme insère alors dans le cache tous les itemsets dont le support a été évalué. Une fois qu'un itemset est inséré dans le cache, il n'est jamais remplacé. L'algorithme est ainsi exécuté jusqu'à ce que le cache soit rempli. Il n'effectue pas de remplacement, d'où son nom.

Le deuxième algorithme est **l'algorithme SR(Simple Replacement)**. Les auteurs considèrent que tous les itemsets ne sont pas d'égale importance, qu'il est plus intéressant de garder des itemsets de plus grand support car ils sont plus souvent utilisés et le calcul de leurs supports est plus coûteux. L'algorithme SR fonctionne comme NR jusqu'à ce que le cache soit rempli. Dans ce cas, les itemsets de plus faible support sont remplacés par ceux de plus grand support. Les auteurs soulignent que les algorithmes NR et SR peuvent aussi fonctionner dans le cas de requêtes avec des contraintes sur les items.

Le troisième algorithme qui est présenté comme le plus efficace est **l'algorithme BR (Benefit Replacement)**. Il est basé sur la notion de support garanti. Par la suite, on note $buffer(k)$ tous les k-itemsets du cache considéré.

Définition 4.4 *Pour tout k, le support garanti d'un buffer k, noté gsup(k), est la valeur maximale de support telle que tous les k-itemsets de support supérieur ou égal à gsup(k) sont dans* $buffer(k)$.

Les auteurs proposent de stocker dans le cache la valeur $gsup(k)$ pour chaque buffer k. Cela permet d'améliorer la performance de l'extraction lorsque la nouvelle requête a un seuil de support supérieur ou égal à $gsup(k)$. Dans ce cas, il est garanti que tous les k-itemsets fréquents sont dans $buffer(k)$ et l'algorithme doit alors juste faire un scan de ce buffer pour les rechercher (pas d'évaluation de supports ni de génération de candidats à ce niveau). L'algorithme complet est présenté à la figure II.25. Si le seuil minimum de support est inférieur à $gsup(k)$, l'algorithme effectue la génération des candidats, calcule les supports des itemsets qui ne se trouvent pas dans le cache, et les insère dans ce dernier, en effectuant des remplacements si nécessaire. Finalement, il ne retient pour la prochaine génération que les candidats dont le support est supérieur ou égal au seuil minimum.

Examinons maintenant comment les auteurs proposent de gérer le cache pour que sa taille ne dépasse pas une borne maximale. Dans le cas de l'algorithme SR, les auteurs proposaient en cas de saturation du cache de favoriser l'insertion des itemsets ayant un grand support (en supprimant si nécessaire, des itemsets avec un support plus faible). Dans le cas de l'algorithme BR, étant donné un cache C, les auteurs proposent un algorithme permettant de maximiser la fonction de bénéfice définie par :

$$B(C) = \sum_{k} B(k)$$

où pour tout buffer k, on a :

$$B(k) = \frac{100 - gsup(k)}{|buffer(k)|} * AccessFrequency(k)$$

Dans cette fonction, $AccessFrequency(k)$ correspond à la fréquence d'accès du buffer k. La maximisation d'une telle fonction permet de favoriser la conservation en cache des itemsets :

- les plus souvent utilisés dans le processus d'extraction itératif,
- qui minimisent les valeurs des supports garantis $gsup(k)$. La minimisation d'un support garanti $gsup(k)$ permet en effet de maximiser le nombre de situations où il ne sera pas nécessaire d'accéder à la base de données pour calculer les supports d'un ensemble de candidats de niveau k.

– **Entrée :** un ensemble de données $\mathbb{D}$, un seuil de support $minsup$;
– **Sortie :** l'ensemble $Freq_{minsup}(\mathbb{D})$.

1. k = 1 ;
2. **Faire** {
3. **Si** $(minsup \geq gsup(k))$
4. $L_k = \{< Itemset, support > \mid Itemset \in buffer(k)$ et $support \geq minsup\}$;
5. **Sinon**
6. $C_k = \{k - itemsets\ candidats\ en\ utilisant\ Apriori\ Gen\}$;
7. $K_k = \{itemsets\ dans\ C_k\ dont\ le\ support\ est\ dans\ le\ cache\}$;
8. $U_k = C_k - K_k$;
9. **Si** $(U_k \neq \emptyset)$
10. Calculer le support de tous les itemsets de U_k ;
11. Ajouter U_k au cache (effectuer des remplacements si nécessaires) ;
12. $C_k = K_k \cup U_k$;
13. $L_k = \{itemsets \in C_k / \ support \geq minsup\}$;
14. $k = k + 1$;
15. } **Tant que** $(L_{k-1} \neq \emptyset\}$;
16. Retourner $\bigcup_k L_k$;

FIG. II.25 – Algorithme A-priori avec l'algorithme BR

Il faut souligner que l'algorithme BR, contrairement aux algorithmes NR et SR, ne peut pas être facilement adapté à des requêtes d'extraction comportant des critères de sélection autres que des contraintes de support minimal. Par exemple, si une première requête d'extraction est posée pour rechercher tous les itemsets X tels que $sup(X) > \alpha$ et $X \supseteq A$, on ne trouvera pas dans $buffer(k)$ tous les k-itemsets de support supérieur à $gsup(k) = \alpha$, mais seulement les k-itemsets fréquents contenant A.

Deuxieme technique de cache

La deuxième approche est présentée par Boulicault et al. [51]. Les auteurs proposent une approche itérative permettant de calculer de manière efficace les fermés fréquents d'une requête d'extraction. Les auteurs utilisent la notion de représentation condensée pour réduire la taille du cache. Au lieu de garder tous les motifs dont le calcul du support a été fait, ils ne gardent qu'une représentation condensée de ceux-ci, et notamment les itemsets libres, mais sous forme d'enregistrements dont les champs sont la fréquence (voir définition ci-dessous) et la fermeture (ce qui leur permet de calculer les fermés). C'est ainsi qu'ils proposent une extension de l'algorithme CLOSE [75] à une séquence de requêtes en utilisant des contraintes anti-monotones. Cet algorithme utilise les ensembles libres et leurs fermetures comme cache pour le stockage des informations sur les différentes requêtes. Les itemsets fermés (fréquents) constituent une représententation condensée des itemsets fréquents, particulièrement intéressante dans le cas de données corrélées. Leur choix repose ainsi sur deux arguments : l'utilisation de représentations condensées permet d'une part de faire une extraction sur des données fortement corrélées et d'autre part de réduire la taille du cache.
Dans un premier temps, nous nous arrêtons sur quelques rappels et définitions de base, et ensuite sur l'extension de l'algorithme CLOSE à des contraintes anti-monotones sans l'utilisation d'un cache. Ceci nous permettra de comprendre l'algorithme qui utilise le cache, et plus particulièrement le contenu de ce dernier pour optimiser une séquence de requêtes.
Pour la représentation des données et des motifs, nous reprenons le formalisme $\mathbb{O}, \mathbb{R}, \mathbb{P}$ que nous avons présenté dans la partie sur la découverte de motifs intéressants. L'ensemble $\mathbb{P}$ des motifs est ici l'ensemble $\mathbb{I}$ des items, l'ensemble de données étant $\mathbb{D} =< \mathbb{O}, \mathbb{I}, \mathbb{R} >$. Rappelons que nous notons f la fonction qui fait correspondre à un motif l'ensemble des objets qui le contiennent. Soit X un itemset, on appelle *fréquence* de X et on note *freq(X)* la cardinalité de $f(X)$: $freq(X) = |f(X)|$.

Définition 4.5 *Une contrainte C est un critère de qualité défini de l'ensemble $\mathbb{I}$ des items vers $\{true, false\}$.*
Une requête d'extraction est un couple (C, Db) où Db est une base de données transactionnelle et C une contrainte. La réponse à une requête $Q = (C, Db)$ est définie par l'ensemble :

$$R(Q) = \{\ (X, freq(X)) \mid C(X) = true\ \}$$

Ainsi la contrainte $C_{\gamma - freq}(X) \equiv (freq(X) \geq \gamma)$ est la contrainte de fréquence minimale, où un seuil de fréquence γ est défini. Un autre exemple de contrainte est $C_{\neg A}(X) \equiv (A \notin X)$.
Nous rappelons maintenant quelques définitions et propositions.

Définition 4.6 *Une contrainte C est anti-monotone si pour tous itemsets X et X' : $(X' \subseteq X \wedge C(X) = true) \Rightarrow C(X') = true$.*

Comme nous l'avons déjà vu dans la partie sur les représentations condensées, un itemset X est fermé s'il est égal à sa propre fermeture. Ainsi, si on note $ferm(X)$ la fermeture de X, un itemset fermé est un itemset tel que $X = ferm(X)$.

Proposition 4.1 *Soient X et Y deux itemsets.*
- $X \subseteq ferm(X)$.
- *Si* $X \subseteq Y$, *alors* $ferm(X) \subseteq ferm(Y)$.
- $freq(X) = freq(ferm(X))$.

Au vu de la proposition 4.1, les auteurs définissent également les notions de fermeture héritée et de fermeture propre :

Définition 4.7 *Pour tout itemset X :*
- *Etant donné un ensemble de données $\mathbb{D} =< \mathbb{O}, \mathbb{I}, \mathbb{R} >$, soient f et g les fonctions définies pour tout $o \subseteq \mathbb{O}$ et $X \subseteq \mathbb{I}$ par :*

$$f(X) = \{\ o \in \mathbb{O} \mid \forall x \in X, (o,x) \in \mathbb{R}\}$$
$$g(O) = \{\ x \in X \mid \forall\ o \in O, (o,x) \in \mathbb{R}\}$$

  La fermeture de X, notée $ferm(X)$, est définie par :

$$ferm(X) = gof(X)$$

- *La fermeture héritée de X, notée $hferm(X)$, est définie par :*

$$hferm(X) = \bigcup_{Y \subset X,\ |Y|=|X|-1} ferm(Y) \setminus X$$

- *La fermeture propre de X, notée $pferm(X)$, est définie par :*

$$pferm(X) = ferm(X) \setminus (hferm(X) \cup X)$$

Par ailleurs, les auteurs démontrent la proposition suivante :

Proposition 4.2 *Un itemset est libre s'il n'est inclus dans la fermeture d'aucun de ses sous-ensembles.*

Exemple 4.4 *Considérons la base de données transactionnelle représentée à la figure II.28. $ferm(A) = AB$ et $ferm(C) = CD$. Donc AC est libre puisqu'il n'est inclus dans la fermeture d'aucun de ses sous-ensembles A et B.*

Nous pouvons maintenant présenter l'extension de l'algorithme CLOSE à des contraintes anti-monotones proposée par les auteurs. Cet algorithme calcule les itemsets libres et leurs fermetures qui satisfont une contrainte de la forme $C_{\gamma-freq} \wedge C_{am}$, où C_{am} est une contrainte syntaxique, c.a.d. une contrainte qui peut être être calculée sans accéder aux données.
Chaque itemset X est stocké dans un enregistrement aussi noté X avec quatre champs : $X.items$ est la liste des items dans X, $X.hferm$ est la fermeture héritée de X, $X.pferm$ est la fermeture propre de X et $X.freq$ est la fréquence de X. En utilisant la proposition 4.1, on voit que $ferm(X)$ se décompose en l'ensemble $X.items \cup X.hferm \cup X.pferm$. L'algorithme d'extraction, qui est un algorithme par niveau, est présenté à la figure II.26.
A chaque itération, les candidats qui ne vérifient pas la contrainte anti-monotone sont élagués. Ensuite, il y a une passe sur les données pour calculer la fermeture propre et la fréquence. Les itemsets qui ne vérifient pas la contrainte de fréquence sont élagués et on recupère les fermetures des itemsets restants. Ensuite les candidats pour l'itération suivante sont générés par la fonction CandGen. Cette fonction est similaire à la fonction de génération de A-priori. C'est durant cette génération que la fermeture héritée est initialisée par la définition 4.7, et que ne sont finalement

– **Entrée :** Une requête $Q = (C_{\gamma-freq} \wedge C_{am}, \mathbb{D})$ où C_{am} est une contrainte anti-monotone .
– **Sortie :** l'ensemble $\mathcal{O} = \{(X.ferm, X.freq) \mid X$ est libre et $C_{\gamma-freq} \wedge C_{am}(X) = true\}$. Par construction, $\mathcal{O}$ est une représentation condensée de $R(Q)$.

1. $Cand = \{(\emptyset, \emptyset, \emptyset, 0)\}$
2. **while** $Cand \neq \emptyset$ {
3. $\quad Cand = \{X \in Cand, C_{am}(X.items) = true\}$
4. $\quad DBPass(Cand, \mathbb{D})$
5. $\quad Cand = \{X \in Cand, X.freq \geq \gamma\}$
6. $\quad Afficher(\{(X.ferm, X.freq)\ /\ X \in Cand\})$
7. $\quad Cand = CandGen(Cand)$
8. $\quad Cand = \{X \in Cand, X\ est\ libre\}$
9. }

FIG. II.26 – Extension de l'algorithme CLOSE à des contraintes anti-monotones

retenus que les itemsets libres.
On peut noter que c'est le calcul de la fermeture propre et de la fréquence des itemsets qui consomme du temps dans cet algorithme. Si cette information avait été calculée par une autre requête, il serait intéressant de la stocker et de la réutiliser. C'est cela qui justifie l'utilisation d'un cache.

Contenu du cache

La fermeture héritée, comme nous l'avons vu, peut être calculée lors de la génération des candidats en utilisant les fermetures des sous-ensembles de X de taille $|X| - 1$. Il n'est ainsi pas nécessaire de la stocker dans le cache. Les auteurs stockent plutôt la fermeture propre et la fréquence qui pourront être utilisées lors des futures extractions. Le cache est ainsi un ensemble d'enregistrements de la forme *(X.items, X.pferm, X.freq)*, où X est un itemset libre.
Les auteurs posent ensuite une contrainte sur le cache : celui-ci doit être fermé vers le bas. Cela veut dire que si un itemset libre X est dans le cache, alors tout sous-ensemble de X (qui est forcément libre) est aussi dans le cache. Par conséquent, si un itemset libre n'est pas dans le cache, aucun de ses sur-ensembles ne peut y être. Autrement dit, la propriété d'être dans le cache est anti-monotone. Cette propriété est utilisée pour accélérer la recherche d'un itemset dans le cache. Ainsi, on ne recherchera pas dans le cache un itemset dont un des sous-ensembles n'y est pas.

Algorithme

Nous présentons maintenant l'algorithme qui utilise le cache pour optimiser le calcul d'une séquence de requêtes. Les auteurs ajoutent pour chaque itemset X un autre champ booléen $X.InCache$ qui permet de savoir si l'itemset X est dans le cache. Plus précisément, $X.InCache$ permet de savoir si l'itemset est susceptible de se trouver dans le cache. Ainsi, lorsqu'il est à false, ce n'est pas la peine d'aller chercher l'itemset dans le cache. Au départ, $X.InCache$ est initialisé à true pour tout X. Par rapport à l'algorithme II.26 présenté avant, ce nouvel algorithme a en plus comme entrée un cache et comme sortie le cache modifié (voir figure II.27).

– **Entrée :** Une requête $Q = (C_{\gamma-freq} \wedge C_{am}, \mathbb{D})$ où C_{am} est une contrainte anti-monotone et un cache.
– **Sortie :** l'ensemble $\{(X.ferm, X.freq) \mid X$ est libre et $C_{am}(X) = true\}$ et un nouveau cache $C_{new}\}$.

1. $Cand = \{(\emptyset, \emptyset, \emptyset, 0)\}$
2. $C_{new} = C$
3. **while** $Cand \neq \emptyset$ {
4. $\quad Cand = \{X \in Cand, C_{am}(X.items) = true\}$
5. $\quad Cache - Pass(Cand, X)$
6. $\quad DBPass2(Cand, \mathbb{D})$
7. $\quad Cand = \{X \in Cand, X.freq \geq \gamma\}$
8. $\quad Afficher(\{(X.ferm, X.freq)\ /\ X \in Cand\})$
9. $\quad Cand = CandGen(Cand)$
10. $\quad Cand = \{X \in Cand, X\ est\ libre\}$
11. }

FIG. II.27 – Extension de l'algorithme CLOSE à des contraintes anti-monotones

Après avoir supprimé les candidats qui ne vérifient pas la contrainte anti-monotone (cf. ligne 4 de l'algorithme), tous les itemsets candidats X sont parcourus pour tester la valeur du booléen $X.InCache$.
Si la valeur de $X.InCache$ est à false, alors l'itemset X ne peut pas être dans le cache. Si elle est à true, alors l'itemset est cherché dans le cache (ligne 5). S'il y est trouvé, il y a mise à jour de sa fréquence et de sa fermeture propre, sinon son champ $X.InCache$ est mis à false. Ensuite, la fréquence et la fermeture propre des itemsets qui n'ont pas été trouvés dans le cache (i.e. $X.InCache = false$) sont calculées lors d'une passe sur la base de données qui représente l'ensemble de données $\mathbb{D}$ (ligne 6) . Ces itemsets sont ensuite insérés dans le nouveau cache (ligne 7).
C'est durant la phase de génération des candidats (ligne 10) que le champ $X.InCache$ de chaque nouveau candidat X est initialisé avec la conjonction des champs $Y.InCache$ de chaque sous-ensemble Y de X de taille $|X| - 1$. Cela veut dire que ce champ est à false si et seulement si il existe un sous-ensemble de X qui n'est pas dans le cache. Ceci implique, puisque le cache est fermé vers le bas, que X ne peut pas être dans le cache. Cette procédure initialise aussi la fermeture héritée des itemsets candidats.

Exemple 4.5 *Supposons qu'on dispose de la base de données transactionnelle représentée à la figure II.28, et supposons que l'on pose une première requête* $Q_1 = ((C_{2-freq} \wedge C_{\neg B}), \mathbb{D})$, *où* $C_{2-freq}(X) \equiv freq(X) \geq 2$ *et* $C_{\neg B}(X) \equiv (B \notin X)$. *Nous alons exécuter l'algorithme en commençant par les candidats de niveau 1. Dans un premier temps, il n'y a rien dans le cache. Le premier niveau est constitué par les 1-itemsets A, C, D et E. On notera que B n'apparaît pas au niveau 1, parce que* $C_{\neg B}(B) = false$ *(cf. algorithme II.26, ligne 3).*
$A.freq = 3$ *;*
$A.pferm = A.ferm \backslash (A.hferm) \cup A) = AB \backslash A = B$ *; L'enregistrement (A,B,3) sera donc stocké dans le cache aprés la passe sur les données. De même,*

A	B	C	D	E
A	B			
A	B	D		
C	D			

FIG. II.28 – Base de données exemple

$C.pferm = D$; $C.freq = 2$; $D.pferm = \emptyset$; $D.freq = 3$; $E.pferm = ABCD$; $E.freq = 1$;
An niveau 2, les candidats sont AC, AD et CD, mais CD n'est pas libre. On va donc refaire les mêmes calculs pour les candidats restants. Il n'y a pas de candidats de niveau 3 puisque AC n'est pas fréquent. Le cache est donc composé comme suit :
$\{(A, B, 3), (C, D, 2), (D, \emptyset, 3), (E, ABCD, 1), (AC, E, 1)(AD, \emptyset, 2)\}$.
Considérons maintenant une seconde requête $Q_2 = (C2 - freq \wedge C\neg E, \mathbb{D})$, *où* $C_{\neg E}(X) = (E \notin X)$, *et reprenons l'algorithme.*
A est recherché dans le cache et puisqu'il s'y trouve, sa fréquence et sa fermeture sont mises à jour. B ne se trouve pas dans le cache, son champ InCache est mis à faux, et ainsi le calcul de la fréquence et de la fermeture sera effectué lors de la passe sur les données, et il sera inséré dans le cache. C et D se trouvent dans le cache. Leur traitement est le même que celui de A. Au niveau 2, les candidats libres sont AC, AD, BC, BD. AC et AD se trouvant dans le cache, les calculs seront faits lors de la passe pour BC et BD. Pour ce cas aussi, il n'y a pas de candidats de niveau 3. Le cache est finalement composé comme suit :
$\{(A, B, 3), (B, A, 3), (C, D, 2), (D, \emptyset, 3), (E, ABCD, 1), (AC, E, 1), (AD, \emptyset, 2), (BC, E, 1), (BD, \emptyset, 2)\}$.
On peut ainsi constater que beaucoup de calculs n'ont pas été faits sur la base de données, ce qui réduit le temps de calcul de la deuxième requête. De même, la taille n'a pas beaucoup augmenté. On peut néanmoins constater que A et B ayant les mêmes fermetures, on pourrait ne garder qu'un des deux enregistrements.

4.5 Conclusion

Les langages de requêtes que nous avons vus jouent un rôle prépondérant dans la mise en œuvre de bases de données inductives. Elles sont généralement des extensions de SQL, ce qui veut dire qu'ils rentrent dans le cadre de ces systèmes qui veulent intégrer le système d'extraction de motifs dans les bases de données. D'autre part, ils permettent de définir des requêtes d'extraction (tâches d'extraction), ce qui fait des résultats de ces dernières des objets de la base de données que l'on souhaite manipulables comme tous les autres. Etant donné que le processus d'extraction est très coûteux, le besoin de réutilisation de résultats déjà stockés pour l'optimisation des requêtes est manifeste.
Nous avons étudié deux approches qui rentrent dans ce cadre. La première définit des requêtes d'extraction par le biais des langages que nous avons vus dans la section 4.4.1 et se base sur les relations entre deux requêtes pour trouver la réponse de l'une en utilisant les résultats de l'autre. Les techniques présentées utilisent des langages qui permettent de travailler avec plusieurs relations, mais au niveau itératif, on compare des requêtes d'extraction qui portent sur une même table.
L'autre approche utilise un cache pour l'optimisation des requêtes. Nag et al [72] proposent plusieurs techniques de cache parmi lesquelles se distingue celle basée sur la notion de support garanti. Celui-ci permet dans beaucoup de cas d'éviter les accès à la base parce que le cache peut garantir qu'il contient tous les itemsets fréquents recherchés. La technique utilisée par Boulicaut et al. [51] qui se rapproche plutôt de la première proposée par [72] (Algorithme *NR*),

utilise les représentations condensées (notamment les itemsets libres et leurs fermetures) comme cache pour effectuer une extraction sous contraintes d'une séquence de requêtes. Cette dernière approche, selon les auteurs, est particulièrement intéressante dans le cas de données fortement corrélées, puisque la taille du cache est alors réduite. Il faut souligner que ces approches basées sur l'utilisation de caches considèrent le cas où la base de données source contient une seule relation.

5 Conclusion

Dans cette première partie, nous avons fait l'état de l'art sur la découverte de motifs intéressants, les représentations condensées et l'extraction itérative de motifs.
Dans la découverte de motifs intéressants, nous avons montré que le formalisme général de découverte de motifs intéressants défini par Mannila et al [63] s'exprime aussi bien dans le contexte de données stockées dans une table que dans le contexte d'une base de données pouvant contenir plusieurs tables relationnelles. Dans le cas de données stockées dans une table (table classique ou table multidimensionnelle), nous avons utilisé le formalisme $\mathbb{ORP}$ présenté par Lakhal et al. [57] pour la représentation des données et des motifs. Dans le cas de données stockées dans plusieurs tables (base de données), nous avons présenté des approches qui utilisent la logique de premier ordre pour la représentation des données et des motifs. Nous présenterons dans notre contribution une approche itérative pour l'extraction de requêtes conjonctives, qui utilise aussi la logique de premier ordre pour la représentation des données et des motifs.

Nous avons passé en revue les différentes formes de représentations condensées et les différentes approches de leur mise en œuvre qui ont proposées jusqu'à maintenant. Cependant il s'agit là de représentations condensées d'une réponse à une requête d'extraction. Notre contribution consistera à présenter une représentation condensée non pas d'une réponse à une requête d'extraction, mais à un ensemble de réponses.

Nous avons ensuite montré le lien qui existe entre l'extraction itérative de motifs et les bases de données inductives. Dans ce sens, nous avons d'une part, présenté différents langages de bases de données inductives et d'autre part, différentes techniques de réutilisation de résultats déjà stockés d'une extraction pour l'optimisation d'une nouvelle extraction. Nous présenterons dans la partie contribution une approche itérative d'extraction de requêtes qui, d'une part, propose un langage de manipulation d'un ensemble de requêtes, et d'autre part réutilise la représentation condensée d'une réponse à une requête d'extraction pour optimiser l'extraction d'une nouvelle requête.

Chapitre III

Contribution

1 Introduction

Le processus d'extraction de connaissances est un processus itératif et interactif complexe. Etant donné que, d'une part, un utilisateur, après une première requête d'extraction, peut raffiner sa requête et que, d'autre part, plusieurs utilisateurs peuvent exécuter des requêtes similaires voire identiques, requêtes dont les temps d'exécution sont très importants, nous proposons une approche itérative de l'extraction des motifs. Elle consiste à utiliser le résultat des requêtes d'extractions antérieures pour optimiser le calcul de la nouvelle requête.
Cette approche est présentée dans le cadre de l'extraction des motifs dans le cas relationnel, où, comme nous l'avons vu dans la partie sur la découverte de motifs intéressants, il y a une explosion de l'espace de recherche. Il apparaît ainsi nécessaire, comme nous l'avons vu dans cette même partie, de proposer un biais ou contexte d'extraction pour réduire la taille de cet espace.
L'approche itérative est présentée sous deux formes : la première approche, qui utilise un formalisme logique, effectue une extraction itérative de requêtes conjonctives basée sur des vues sur ces requêtes. La seconde approche est une extraction itérative par combinaison de contextes d'extractions en utilisant les opérations de l'algèbre relationnelle.
Le premier problème qui se pose à cette approche est le stockage de la réponse à une requête, puisque cette réponse est de taille importante. Pour cela, nous stockons la représentation condensée de la réponse à une requête, qui sera utilisée pour l'optimisation du calcul de la réponse à la requête future.
Cependant, étant donné que nous avons plusieurs requêtes, donc plusieurs réponses non indépendantes, l'autre problème qui se pose est le stockage optimal d'un ensemble de réponses à des requêtes.
Une autre contribution de ce mémoire est de proposer des représentations condensées d'un ensemble de réponses à des requêtes.
Nous parlerons dans la première partie de l'extraction itérative de requêtes conjonctives basée sur des vues. Dans la deuxième partie, nous présenterons l'extraction itérative par combinaison de contextes d'extractions. La troisième partie portera sur l'implémentation de l'approche itérative par combinaison de contextes d'extraction. Finalement, la quatrième partie est consacrée aux représentations condensées d'un ensemble de réponses à des requêtes.

2 Extraction itérative de requêtes conjonctives à partir de vues

Les travaux exposés dans cette partie ont été présentés dans [39, 40].

2.1 Introduction

Le processus d'extraction de connaissances est un processus interactif et itératif. Cela signifie qu'un utilisateur, après une première extraction, peut décider de raffiner celle-ci, ou de formuler une requête d'extraction similaire. Pourtant, la plupart des approches de découvertes de connaissances n'utilisent pas les résultats des premières requêtes d'extraction pour le calcul des requêtes suivantes. Il en est de même lorsque plusieurs utilisateurs effectuent une extraction à partir des mêmes données.
L'approche que nous proposons utilise les résultats des requêtes d'extractions antérieures pour optimiser le calcul de la nouvelle requête.
Cette approche utilise les frontières des théories ([63]) des premières requêtes d'extractions pour optimiser le calcul de la nouvelle requête. Rappelons que nous avons défini les frontières positives et négatives de la réponse à une requête dans la partie consacrée aux représentations condensées.
Le formalisme que nous utilisons pour la représentation des motifs est proche de celui introduit

dans [25, 27]. Cependant, notre approche se différencie principalement de [25, 27] par l'introduction de vues dans l'expression des motifs et des requêtes. Les vues permettent d'une part de formuler simplement des requêtes d'extractions très complexes. D'autre part, l'optimisation du calcul de la réponse à la nouvelle requête va se faire par des tests d'inclusions de requêtes, tests qui seront facilités par la nature conjonctive des requêtes et des vues. L'exemple suivant donne une idée de notre approche.

Exemple 2.1 *Soit S un ensemble de données contenant des faits sur les clients, produits, magasins et ventes, c'est à dire des faits sur les prédicats* $Cust(Cid, Cjob, Caddr)$, $Prod(Pid, Ptype)$, $Store(Sid, Sname, Saddr)$ *et* $Sales(Cid, Pid, Sid, Date)$.

D'abord, supposons que l'on s'intéresse aux caractéristiques de l'ensemble des consommateurs. Pour cela, on introduit une vue AllCust, appelée la vue de référence, *et définie sous la forme d'une requête datalog comme suit :*
$AllCust(Cid) : -Cust(Cid, Cjob, Caddr)$.
L'intérêt des motifs sera calculé par rapport à cette vue.

Supposons maintenant que l'on s'intéresse aux relations entre les professions des consommateurs et les types de produits qu'ils achètent. Nous introduisons la requête de base *suivante :*

$$\overline{Q}_1 : \quad (\exists\ Cjob, Caddr, Pid, Sid, Date, Ptype)(AllCust(Cid) \wedge Cust(Cid, Cjob, Caddr) \wedge Sales(Cid, Pid, Sid, Date) \wedge Prod(Pid, Ptype))$$

Finalement, supposons que l'on veut extraire des relations spécifiques entre certaines professions et certains types de produits, par exemple entre les professeurs et les avocats d'une part, et le lait et la bière d'autre part. Ceci est déclaré sous la forme d'un ensemble de substitutions $\Sigma_1 = \{Cjob = Professeur, Cjob = Avocat, Ptype = Lait, Ptype = Bière\}$, *que nous appelons* le domaine d'intérêt. *Signalons que pour la suite, nous utilisons toujours des majuscules pour les constantes.*

Dans notre approche, un contexte d'extraction *est défini par le triplet* $C_1 = \langle AllCust, \overline{Q}_1, \Sigma_1 \rangle$, *où AllCust est la vue de référence,* $\overline{Q}_1$ *est la requête de base, et* Σ_1 *est le domaine d'intérêt. Les motifs que nous considérons sont obtenus par spécialisation de la requête de base* $\overline{Q}_1$ *en utilisant les substitutions dans le domaine d'intérêt* Σ_1. *Par exemple, si on instancie* $Cjob$ *par* $professeur$ *et Ptype par Lait, on obtient la requête suivante :*

$$Q : \quad (\exists\ Caddr, Pid, Sid, Date)(AllCust(Cid) \wedge Cust(Cid, Professeur, Caddr) \wedge Sales(Cid, Pid, Sid, Date) \wedge Prod(Pid, Lait))$$

La réponse à cette requête contient alors tous les noms des consommateurs qui sont professeurs et qui ont acheté du lait.
Etant donné un ensemble de données S, une requête Q est extraite si elle est fréquente, c'est à dire si le rapport de la cardinalité de sa réponse dans S sur la cardinalité de la réponse à la vue de référence est supérieur ou égal à un seuil. Le problème posé est le calcul de l'ensemble des requêtes fréquentes définies par un contexte donné.

Supposons maintenant qu'un autre utilisateur s'intéresse aux relations entre les professions des consommateurs et les types de produits qu'ils achètent, mais que son domaine d'intérêt soit défini par $\Sigma_2 = \{Cjob = Professeur, Cjob = Chercheur, Ptype = Lait, Ptype = Pain\}$. *Alors, cet utilisateur considère le contexte d'extraction* $C_2 = \langle AllCust, \overline{Q}_1, \Sigma_2 \rangle$, *et l'on voit que le calcul de l'ensemble des requêtes fréquentes définies par* C_2 *peut être optimisé en utilisant les résultats de celui défini par* C_1. *En effet,* C_1 *et* C_2 *ont la même vue de référence, la même requête de base et leurs domaines d'intérêt concernent les professeurs et le lait.*

Dans notre formalisme, nous introduisons aussi des vues pour formuler des requêtes d'extraction plus complexes. Supposons que l'on s'intéresse aux achats des consommateurs faits dans leur propre ville. Pour cela, on introduit la vue SLocal suivante :

$$SLocal(Cid, Pid) :- \quad Cust(Cid, Cjob, Caddr) \wedge Sales(Cid, Pid, Sid, Date) \wedge Store(Sid, Sname, Caddr)$$

On peut alors considérer le contexte d'extraction défini par $C_3 = \langle AllCust, \overline{Q}_3, \Sigma_1 \cup \Sigma_2 \rangle$ *où* $\overline{Q}_3$ *est la requête de base suivante :*

$$\overline{Q}_3 : \quad (\exists\, Cjob, Caddr, Pid, Ptype)(AllCust(Cid) \wedge Cust(Cid, Cjob, Caddr) \wedge SLocal(Cid, Pid) \wedge Prod(Pid, Ptype))$$

Nous montrons qu'en se basant sur le fait que $\overline{Q}_3$ *est incluse dans* $\overline{Q}_1$ *(au sens de [21]), le calcul de l'ensemble des requêtes fréquentes définies par* C_3 *peut être optimisé en utilisant les résultats de ceux définis par* C_1 *et* C_2*. Considérons par exemple la requête suivante :*

$$Q' : \quad (\exists\, Caddr, Pid, Sid, Date)(AllCust(Cid) \wedge Cust(Cid, Professeur, Caddr) \wedge SLocal(Cid, Pid) \wedge Prod(Pid, Lait))$$

On peut noter, étant donné que Q' *est incluse dans* Q*, que si* Q *n'est pas intéressante, alors* Q' *ne peut pas être intéressante.*

Finalement, considérons le cas où la vue de référence change. Par exemple, supposons que l'on s'intéresse seulement aux consommateurs qui sont professeurs. Pour cela, on peut définir la vue de référence $CProf$ *suivante :*

$$CProf(Cid) :- Cust(Cid, Professeur, Caddr)$$

Dans ce cas, nous montrons que, puisque $CProf$ *est incluse dans AllCust, le calcul de l'ensemble des requêtes fréquentes définies par* $C_4 = \langle CProf, \overline{Q}_1, \Sigma_1 \rangle$ *peut être optimisé en utilisant celui défini par* C_1*.*

Le reste de cette partie est organisé comme suit : A la section 2.2, nous donnons les définitions formelles des contextes d'extraction et des motifs que nous considérons. L'approche itérative est présentée à la section 2.3. Nous montrons comment les notions d'inclusion de requêtes et de frontières peuvent être utilisées pour optimiser la découvete de requêtes fréquentes et proposons un algorithme itératif. La section 2.4 est une conclusion qui souligne les différents problèmes posés par cette approche avec des remarques et suggestions.

2.2 Extraction à partir de vues

2.2.1 Définitions de base et notations

Nous supposons que nous disposons d'un ensemble fixe de prédicats d'arités données, appelés *prédicats de base*. Les vues et requêtes d'extraction sont définies à partir de ces prédicats de base. Nous supposons également qu'à toute entrée d'un prédicat de base est associé un ensemble prédéfini de valeurs, appelé *domaine.*[1]

Rappelons que si p est un prédicat de base d'arité n, alors $p(t_1, t_2, \ldots, t_n)$ est un atome, où chaque t_i est un terme, i.e. soit une constante, soit une variable. Dans notre contexte, nous considérons uniquement des *vues* définies sous la forme de *requêtes conjonctives* correspondant à

[1]La notion de domaine utilisée ici, correspond à celle de domaine actif dans les bases de données relationnelles.

la définition classique donnée dans [91]. Plus précisément, toute vue considérée v est définie par une seule règle Datalog de la forme :

$$v(Y) :- p_1(X_1), p_2(X_2), \ldots, p_m(X_m)$$

où p_1, p_2, ..., p_m sont des prédicats de base, X_1, X_2, ..., X_m sont des ensembles de termes (constantes ou variables) ; Y est un sous-ensemble des variables ayant une occurrence dans le corps de la règle. Les vues considérées sont donc définies par des règles *saines* [91]. Rappelons qu'une règle est dite saine si elle ne contient pas de variable qui apparaît en tête mais pas dans le corps. Par la suite, nous notons par v, aussi bien le nom de la vue correspondante que l'atome de la tête de la règle la définissant. Nous notons enfin par $Var(v)$ l'ensemble des variables de v, i.e. $Var(v) = Y$.

De plus, nous supposons que nous disposons d'un ensemble de données, c'est à dire, comme nous l'avons dit dans la partie sur la découverte de motifs intéressants, d'un ensemble fixe de *faits*, i.e. des atomes de la forme $p(a_1, a_2, \ldots, a_n)$, où p est un prédicat de base d'arité n et chaque a_i est une constante. C'est à partir de cet ensemble que les motifs intéressants seront découverts. De plus, nous supposons :

- que tout ensemble de données est compatible avec les domaines des prédicats de base, i.e. que si $p(a_1, a_2, \ldots, a_n) \in S$, alors pour tout $i = 1 \ldots n$, a_i est dans le domaine de la i-ième entrée du prédicat de base p,
- que si un fait n'est pas dans S, alors ce fait est faux. Nous nous plaçons donc dans le cadre de l'hypothèse du *monde clos* [79].

Soit Var et $Const$ l'ensemble des variables et constantes. Etant donné un ensemble de faits S, et une vue v définie par la règle Datalog :

$$v(Y) :- p_1(X_1), p_2(X_2), \ldots, p_m(X_m)$$

nous notons par $v[S]$ l'ensemble des n-uplets $Y\theta$ où θ est une substitution définie de Var dans $Const$ telle que $p_i(X_i)\theta \in S$ pour tout $i = 1 \ldots m$.

Exemple 2.2 *Dans le cadre de l'exemple 2.1, un exemple d'ensemble de données est présenté ci-dessous. Cet exemple sera utilisé par la suite comme exemple de référence :*

$$\begin{aligned} S = \{ \; & Client(Eric, Avocat, Blois), && Client(Ivan, Professeur, Tours), \\ & Client(Claire, Avocat, Blois), && Client(Anne, Professeur, Orleans), \\ & Produit(1, Lait), && Produit(2, Fromage), \\ & Produit(3, Lait), && Produit(4, Pain), \\ & Magasin(1, Casino, Blois), && Magasin(2, Auchan, Tours), \\ & Magasin(3, Casino, Tours), && Magasin(4, Carrefour, Orleans), \\ & Vente(Eric, 1, 2, Mars), && Vente(Eric, 2, 1, Mars), \\ & Vente(Eric, 4, 1, Avril), && Vente(Ivan, 1, 3, Mars), \\ & Vente(Claire, 2, 3, Juin), && Vente(Anne, 3, 4, Mars)\} \end{aligned}$$

Cet ensemble indique par exemple qu'Anne est professeur et vit à Orléans, qu'elle a acheté du lait au magasin Carrefour d'Orléans au mois de Mars. Par ailleurs, la vue PCasino définie par :

$$PCasino(Pid, Ptype) :- Prod(Pid, Ptype), Sales(Cid, Pid, Sid, Date), Store(Sid, Casino, Saddr)$$

définit l'ensemble des produits vendus à Casino et leurs types. Nous avons :
$PCasino[S] = \{(1, Lait), (2, Fromage), (4, Pain)\}$.

2.2.2 Contextes d'extraction et vues

Nous introduisons maintenant la définition formelle de *contexte d'extraction* que nous utilisons pour spécifier la forme des motifs qui sont extraits. Les contextes d'extraction que nous considérons utilisent les notions de *vues* et de substitutions élémentaires. Rappelons qu'une substitution σ est dite élémentaire s'il existe une unique variable $x_0 \in Var$ telle que, pour tout terme $t \neq x_0$, $\sigma(t) = t$ et $\sigma(x_0) \in Const$.

Définition 2.1 Contexte d'extraction.
Un contexte d'extraction C *est un triplet* $C = \langle r, \overline{Q}, \Sigma \rangle$ *où :*

- r *est une vue, appelée la* vue de référence *de* C,
- $\overline{Q}$ *est une requête conjonctive de la forme* $(\exists Y)(r(K) \wedge v_1(X_1) \wedge \ldots \wedge v_N(X_N))$ *où pour tout* $i = 1, \ldots, N$, v_i *est soit une vue ou un prédicat de base,* $K = Var(r)$, $X_1, \ldots, X_N$ *sont des ensembles de variables et* $Y = (X_1 \cup \ldots \cup X_N) \setminus K$.
  $\overline{Q}$ *est appelée la* requête de base *de* C.
- $\Sigma = \{\sigma_1, \ldots, \sigma_P\}$ *est un ensemble de substitutions élémentaires* σ_i *telles que* $\sigma_i(x) = x$ *pour tout* $x \in K$. Σ *est appelé le* domaine d'intérêt *de* C.

Etant donné un ou plusieurs contextes d'extraction $C = \langle r, \overline{Q}, \Sigma \rangle$, nous nous intéressons par la suite à deux types de motifs. Les premiers sont des *requêtes conjonctives* dont les variables libres sont les variables de la vue de référence r. Les seconds sont des *implications logiques* où toutes les variables de r sont quantifiées universellement, alors que toutes les autres variables sont quantifiées existentiellement.

Définition 2.2 Espace de recherche
Etant donné un contexte d'extraction $C = \langle r, \overline{Q}, \Sigma \rangle$, *on définit l'ensemble des* requêtes conjonctives $\mathcal{Q}(C)$ *par :*

$$\mathcal{Q}(C) = \{\overline{Q}\beta \ / \ \beta \in \Sigma^*\}$$

où Σ^* *est l'ensemble de toutes les compositions de substitutions élémentaires de* Σ.

Etant donnés deux contextes d'extraction $C_1 = \langle r, \overline{Q}_1, \Sigma_1 \rangle$ *et* $C_2 = \langle r, \overline{Q}_2, \Sigma_2 \rangle$, *de même vue de référence et de même ensemble de variables libres* K, *on définit l'ensemble des* règles (ou implications) $\mathcal{I}(C_1, C_2)$ *par :*

$$\mathcal{I}(C_1, C_2) = \{(\forall K)(P_1 \Rightarrow P_2) \ / \ P_1 \in \mathcal{Q}(C_1) \text{ et } P_2 \in \mathcal{Q}(C_2)\}$$

Exemple 2.3 *Dans le cadre de l'exemple 2.1, considérons les contextes d'extraction* $C_3 = \langle AllCust, \overline{Q_3}, \Sigma_J \rangle$ *et* $C_4 = \langle AllCust, \overline{Q_4}, \Sigma_T \rangle$ *où* $\overline{Q_3}$, $\overline{Q_4}$, Σ_J *et* Σ_T *sont définis comme suit :*

$$\overline{Q}_3 : \quad (\exists\ Cjob, Caddr, Pid, Ptype)(AllCust(Cid) \wedge Cust(Cid, Cjob, Caddr) \wedge SLocal(Cid, Pid) \wedge Prod(Pid, Ptype))$$

$$\overline{Q}_4 : \quad (\exists\ Cjob, Caddr, Pid, Ptype)(AllCust(Cid) \wedge PCasino(Pid, Ptype) \wedge SMars(Cid, Pid))$$

Soit Θ *l'ensemble de toutes les substitutions élémentaires (de toutes les variables).*
$\Sigma_J = \{\sigma \in \Theta \mid \sigma(Cjob) \in \{Professeur, Avocat\}\}$

$\Sigma_T = \{\sigma \in \Theta \mid \sigma(Ptype) \in \{Lait, Fromage, Pain\}\}$
et où les vues sont définies par :

$$\begin{array}{rcl} AllCust(Cid) & :- & Cust(Cid, Cjob, Caddr) \\ SLocal(Cid, Pid) & :- & Cust(Cid, Cjob, Caddr), Sales(Cid, Pid, Sid, Date), \\ & & Store(Sid, Sname, Caddr) \\ PCasino(Pid, Ptype) & :- & Prod(Pid, Ptype), Sales(Cid, Pid, Sid, Date), \\ & & Store(Sid, Casino, Saddr) \\ SMars(Cid, Pid) & :- & Sales(Cid, Pid, Sid, Mars) \end{array}$$

Les contextes d'extraction C_3 et C_4 permettent de rechercher les liens existant entre les professions des clients ayant effectué des achats dans leur ville, et les types des produits vendus à Casino que ces clients ont achetés au mois de Mars. Des exemples de requêtes et de règle appartenant respectivement à $\mathcal{Q}(C_3)$, $\mathcal{Q}(C_4)$ et $\mathcal{I}(C_3, C_4)$ sont présentés ci-dessous :

$$\begin{array}{rcl} Q_1 & : & (\exists Caddr, Pid, Ptype)(AllCust(Cid) \wedge Cust(Cid, Professeur, Caddr)) \wedge \\ & & SLocal(Cid, Pid) \wedge Prod(Pid, Ptype)) \\ Q_2 & : & (\exists Pid)(AllCust(Cid) \wedge SMars(Cid, Pid) \wedge PCasino(Pid, Lait)) \\ R_{12} & : & (\forall X_1)(Q_1 \Rightarrow Q_2) \end{array}$$

La règle R_{12} stipule que tous les clients qui ont effectué des achats dans leur ville et qui sont professeurs, sont des clients qui ont acheté en mars du lait vendu à Casino.
On remarquera que l'introduction des vues permet d'exprimer simplement des requêtes complexes.

2.2.3 Mesures d'intérêt.

Toutes les mesures d'intérêt définies dans le cadre des règles d'association standards [58] peuvent être étendues à notre formalisme.

Etant donné un ensemble de données S et un contexte d'extraction $C = \langle r, \overline{Q}, \Sigma \rangle$, avec $K = Var(r)$, soit $Q : (\exists Y')(r(K) \wedge v_1(X_1)\beta \wedge \ldots \wedge v_N(X_N)\beta)$ une requête de $Q(C)$. La réponse de Q dans S, $Q(S)$ est ici l'ensemble des n-uplets $K\theta$ où θ est une substitution telle que $K\theta \in r(S)$ et $(X_i\beta)\theta \in v_i(S)$ pour tout $i = 1, ..., N$. On peut alors donner la définition du support.

Définition 2.3 Support d'une requête
Etant donné un contexte d'extraction $C = \langle r, \overline{Q}, \Sigma \rangle$, soit Q une requête de la forme $Q : (\exists Y')(r(K) \wedge v_1(X_1)\beta \wedge \ldots \wedge v_N(X_N)\beta)$. Pour tout ensemble de données S, le support de Q dans S par rapport à r est défini comme suit :

$$Sup(Q \ / \ r, S) = |Q(S)| \ / \ |r(S)|$$

Le support et la confiance d'une règle sont définis comme suit :

Définition 2.4 - Support et confiance d'une règle. *Etant donné deux contextes d'extraction $C_1 = \langle r, \overline{Q}_1, \Sigma_1 \rangle$ et $C_2 = \langle r, \overline{Q}_2, \Sigma_2 \rangle$, de même vue de référence r avec $K = Var(r)$, soit $R : (\forall K)(Q_1 \Rightarrow Q_2)$ une règle (ou implication) appartenant à $\mathcal{I}(C_1, C_2)$. Pour tout ensemble de données S :*

1. *Le* support de R dans S relativement à r, *noté $Sup(R/r, S)$, est défini par :*

$$Sup(R/r, S) = Sup(Q_1 \wedge Q_2/r, S)$$

2. *La* confiance de R dans S relativement à r, *notée $Conf(R/r, S)$, est définie par :*

$$Conf(R/r, S) = \frac{|(Q_1 \wedge Q_2)(S)|}{|Q_1(S)|} = \frac{Sup(Q_1 \wedge Q_2/r, S)}{Sup(Q_1/r, S)}$$

Exemple 2.4 *Considérons l'ensemble de données S défini dans l'exemple 2.2 et les contextes d'extraction spécifiés dans l'exemple 2.3. Les requêtes Q_1 et Q_2 du même exemple 2.3 ont pour supports :*

$$Sup(Q_1/AllCust, S) = 2/4 \text{ et } Sup(Q_2/AllCust, S) = 2/4$$

Pour déterminer les support et confiance de la règle R_{12} de l'exemple 2.3, il faut maintenant calculer le support de la requête $Q_1 \wedge Q_2$ définie par :

$$(\exists Caddr, Pid1, Ptype, Pid2)(AllCust(Cid) \wedge Cust(Cid, Professeur, Caddr) \wedge \\ Slocal(Cid, Pid1) \wedge Prod(Pid1, Ptype) \wedge SMars(Cid, Pid2) \wedge PCasino(Pid2, Lait))$$

Nous avons $Sup(R_{12}/AllCust, S) = Sup(Q_1 \wedge Q_2/AllCust, S) = 1/4$.
Par conséquent, la confiance de la règle R_{12} est :

$$Conf(R_{12}/AllCust, S) = \frac{Sup(Q_1 \wedge Q_2/AllCust, S)}{Sup(Q_1/AllCust, S)} = \frac{1}{2}$$

2.2.4 Propriétés sur les mesures d'intérêt

Appliquons dans notre contexte une propriété importante du support que nous avons énoncée dans la partie sur la découverte de motifs intéressants. Etant donné un ensemble de données S et un contexte d'extraction $C = \langle r, \overline{Q}, \Sigma \rangle$, pour tout $Q \in Q(C)$, on a : $0 \leq Sup(Q \ / \ r, S) \leq 1$ et $Sup(Q \ / \ r, S) = 1$ si et seulement si S est un modèle de l'implication $(\forall K)(r \Rightarrow Q)$ où $K = Var(r)$. Dans le cas de l'exemple 2.3, cela signifie que $Sup(Q_1 \ / \ AllCust, S) = 1$ si tous les professeurs font des achats dans leur ville .

La propriété suivante est essentielle par la suite. Elle indique la monotonie du support par rapport à l'implication, et étend ainsi la propriété de base du support dans le contexte des règles d'associations [2], connue sous le nom de "Apriori trick".

Proposition 2.1 *Soit deux contextes d'extraction $C_1 = \langle r_1, \overline{Q}_1, \Sigma_1 \rangle$ et $C_2 = \langle r_2, \overline{Q}_2, \Sigma_2 \rangle$ où $K_1 = Var(r_1)$, $K_2 = Var(r_2)$ et $K_1 \subseteq K_2$, et soit deux requêtes Q_1 et Q_2 de $\mathcal{Q}(C_1)$ et $\mathcal{Q}(C_2)$ respectivement.*
Pour tout ensemble de données S, si S est un modèle des implications $(\forall K_2)(r_1 \Rightarrow r_2)$ et $(\forall K_2)(Q_2 \Rightarrow Q_1)$, alors $Sup(Q_2/r_2, S) \leq Sup(Q_1/r_1, S)$.

Preuve :
$Sup(Q_2 \ / \ r_2, S) = |Q_2(S)| \ / \ |r_2(S)|$.
Comme S est un modèle de $(\forall K_2)(Q_2 \Rightarrow Q_1)$, on a $Q_2(S) \subseteq Q_1(S)$ et par conséquent $|Q_2(S)| \leq |Q_1(S)|$. Ainsi $Sup(Q_2 \ / \ r_2, S) \leq |Q_1(S)| \ / \ |r_2(S)|$.
De même, S est un modèle de $(\forall K_2)(r_1 \Rightarrow r_2)$ implique $|r_1(S)| \leq |r_2(S)|$.
Par conséquent, $Sup(Q_2/r_2, S) \leq Sup(Q_1/r_1, S)$. $\diamond$

Par la suite, la proposition précédente est uniquement utilisée lorsque $K_1 = K_2$ (de façon à ce que l'implication $(\forall K_2)(r_1 \Rightarrow r_2)$ corresponde à une règle saine), et lorsque les implications $(\forall K_2)(r_1 \Rightarrow r_2)$ et $(\forall K_2)(Q_2 \Rightarrow Q_1)$ sont des formules valides, i.e. lorqu'elles sont vraies indépendamment de S (voir paragraphe 2.3.3).

Exemple 2.5 *Dans le cadre de l'exemple 2.1, considérons les contextes d'extraction suivants :*

$$C_1 = \langle AllCust, \overline{Q_5}, \Sigma_J \cup \Sigma_V \cup \Sigma_T \rangle$$
$$C_2 = \langle AllCust, \overline{Q_6}, \Sigma_J \cup \Sigma_V \cup \Sigma_T \rangle$$

où

$$\overline{Q}_5 : \quad (\exists\ Cjob, Caddr, Pid, Ptype)(AllCust(Cid) \wedge Cust(Cid, Cjob, Caddr) \wedge Purchase(Cid, Pid) \wedge Prod(Pid, Ptype))$$

$$\overline{Q}_6 : \quad (\exists\ Cjob, Caddr, Pid, Ptype)(AllCust(Cid) \wedge Cust(Cid, Cjob, Caddr) \wedge LocalPurchase(Cid, Pid) \wedge Prod(Pid, Ptype))$$

Σ_J *et* Σ_T *sont définis dans l'exemple 2.3,*
$\Sigma_V = \{\sigma \in \Theta \mid \sigma(Caddr) \in \{Blois, Orleans, Tours\}\}$
et les vues $Purchase$ *et* $LocalPurchase$ *sont définies par :*

$$\begin{aligned} Purchase(Cid, Pid) &:- Sales(Cid, Pid, Sid, Date) \\ LocalPurchase(Cid, Pid) &:- Cust(Cid, Cjob, Caddr), Sales(Cid, Pid, Sid, Date), \\ &\quad Store(Sid, Sname, Caddr) \end{aligned}$$

Des exemples de requêtes Q_1 *et* Q_2 *appartenant respectivement à* $\mathcal{Q}(C_1)$ *et* $\mathcal{Q}(C_2)$ *sont donnés ci-dessous :*

$$\begin{aligned} Q_1 : \quad & (\exists Pid)(AllCust(Cid) \wedge Cust(Cid, Avocat, Blois) \\ & \qquad \wedge Purchase(Cid, Pid) \wedge Prod(Pid, Fromage)) \\ Q_2 : \quad & (\exists Pid)(AllCust(Cid) \wedge Cust(Cid, Avocat, Blois) \\ & \qquad \wedge LocalPurchase(Cid, Pid) \wedge Prod(Pid, Fromage)) \end{aligned}$$

On vérifie que l'implication $(\forall Cid)(Q_2 \Rightarrow Q_1)$ *est une formule valide (voir paragraphe 2.3.3 et exemple 2.6). D'après la proposition 2.1, nous devons avoir* $Sup(Q_2/AllCust, S) \leq Sup(Q_1/AllCust, S)$*, ce qui est bien vérifié puisque* $Sup(Q_2/AllCust, S) = 1/4$ *et* $Sup(Q_1\ /\ AllCust, S) = 2/4$*. En effet, Claire et Eric sont avocats, blésois et achètent du fromage, mais seul Eric a acheté du fromage à Blois.*

Considérons maintenant un exemple de changement de vue de référence. Soit le contexte d'extraction $C_3 = \langle r_3, \overline{Q'}, \Sigma_J \cup \Sigma_T \rangle$ *où la requête de base* $\overline{Q'}$*, la nouvelle vue de référence* r_3 *et la vue* $CBlois$ *sont définies par :*

$$\overline{Q'} : \quad (\exists\ Cjob, Caddr, Pid, Ptype)(r_3(Cid) \wedge CBlois(Cid, Cjob) \wedge LocalPurchase(Cid, Pid) \wedge Prod(Pid, Ptype))$$

$$\begin{aligned} r_3(Cid) &:- Cust(Cid, Cjob, Blois) \\ CBlois(Cid, Cjob) &:- Cust(Cid, Cjob, Blois) \end{aligned}$$

La requête Q_3 *suivante :*

$$\begin{aligned} Q_3 : \quad & (\exists Pid, Cjob)(r_3(Cid) \wedge CBlois(Cid, Cjob) \\ & \qquad \wedge LocalPurchase(Cid, Pid) \wedge Prod(Pid, Fromage)) \end{aligned}$$

appartient à $\mathcal{Q}(C_3)$*. On vérifie simplement que l'implication* $(\forall Cid)(Q_2 \Rightarrow Q_3)$ *est une formule valide, de même que l'implication* $(\forall\ Cid)(r_3 \Rightarrow AllCust)$*. D'après la proposition 2.1, nous devons donc avoir* $Sup(Q_2/AllCust, S) \leq Sup(Q_3/r_3, S)$*, ce qui est bien vérifié puisque* $Sup(Q_2/AllCust, S) = 1/4$ *et* $Sup(Q_3/r_3, S) = 1/2$*. En effet, si les deux clients Eric et Claire habitent Blois et achètent du fromage, seulement Eric achète du fromage à Blois.*

La proposition suivante établit le lien entre l'implication logique et la confiance d'une règle.

Proposition 2.2 *Etant donné deux contextes d'extraction* $C_1 = \langle r, \overline{Q_1}, \Sigma_1 \rangle$ *et* $C_2 = \langle r, \overline{Q_2}, \Sigma_2 \rangle$ *de même vue de référence* r*, soit une règle* $R : (\forall K)(Q_1 \Rightarrow Q_2)$ *de* $\mathcal{I}(C_1, C_2)$ *où* $K = Var(r)$*. Dans le cadre de l'hypothèse du monde clos, pour tout ensemble de données* S*, nous avons :*

1. $0 \leq Conf(R/r, S) \leq 1$.
2. $Conf(R/r, S) = 1$ *si et seulement si* S *est un modèle de l'implication* $(\forall K)(Q_1 \Rightarrow Q_2)$.

2.3 Approche itérative

Dans cette section, nous commençons par rappeler que la phase de découverte de requêtes fréquentes est la phase essentielle de toute procédure d'extraction de règles intéressantes. Nous nous focalisons ainsi sur le problème de la découverte de requêtes fréquentes à partir d'un contexte d'extraction, puis nous montrons comment l'inclusion de requêtes peut être utilisée pour la comparaison des requêtes de différents contextes d'extraction. Ensuite, nous présentons un algorithme itératif pour le calcul de l'ensemble des requêtes fréquentes d'une nouvelle extraction en utilisant les frontières des ensembles de requêtes déjà calculées.

2.3.1 Découverte de requêtes fréquentes

Selon le formalisme général de Mannila et Toivonen [63], étant donné un contexte d'extraction $C = \langle r, \overline{Q}, \Sigma \rangle$, le calcul de l'ensemble des requêtes fréquentes consiste à trouver la théorie $Th(\mathcal{Q}(C), r, S, q) = \{Q \in \mathcal{Q}(C) \mid q(Q, r, S) \text{ est vrai } \}$ où le prédicat de sélection q est défini pour toute requête Q dans $\mathcal{Q}(C)$ par : $q(Q, r, S)$ est vrai si le support de Q dans S par rapport à r est supérieur à un seuil défini par l'utilisateur.
Dans notre approche, cela est traduit comme suit :
Etant donné un seuil de support α, un contexte d'extraction $C = \langle r, \overline{Q}, \Sigma \rangle$, on note $freq_\alpha(\mathcal{C}, S)$ l'ensemble de toutes les requêtes fréquentes dans $\mathcal{Q}(\mathcal{C})$ dont le support dans S par rapport à r est supérieur ou égal à α,

$$freq_\alpha(C, S) = \{Q \in \mathcal{Q}(C) \mid Sup(Q \mid r, S) \geq \alpha\}.$$

Comme S est constant, on pourra aussi noter $freq_\alpha(C)$.

Etant donné un ensemble de données S, un contexte d'extraction C, $freq_\alpha(C)$ peut être calculé en utilisant l'algorithme générique de Mannila et Toivonen [63]. Dans notre contexte, la phase de génération des candidats (voir Figure III.1) est basée sur la monotonie du support par rapport à l'implication. Plus précisément, soit $K = Var(r)$ et Q une requête de $\mathcal{Q}(C)$. Alors, pour toute substitution élémentaire $\sigma \in \Sigma$, l'implication $(\forall K)(Q\sigma \Rightarrow Q)$ est valide. Par conséquent, d'après la proposition 2.1, on a $Sup(Q\sigma/r, S) \leq Sup(Q/r, S)$, ce qui veut dire que la requête $Q\sigma$ ne peut pas être intéressante si la requête Q n'est pas intéressante.

2.3.2 Découverte de règles intéressantes

Etant donné deux contextes $C_1 = \langle r, \overline{Q_1}, \Sigma_1 \rangle$ et $C_2 = \langle r, \overline{Q_2}, \Sigma_2 \rangle$ de même vue de référence, un seuil minimal de support α et un seuil minimal de confiance β, on note $int_{\alpha,\beta}(C_1, C_2)$ l'ensemble des règles intéressantes :

$$int_{\alpha,\beta}(C_1, C_2) = \{R \in \mathcal{I}(C_1, C_2) \mid Sup(R \ / \ r, S) \geq \alpha \ et \ Conf(R \ / \ r, S) \geq \beta\}.$$

Dans notre approche, pour calculer $int_{\alpha,\beta}(C_1, C_2)$, on commence par calculer successivement $freq_\alpha(C_1)$, $freq_\alpha(C_2)$ et $freq_\alpha(C_{12})$, où $C_{12} = \langle r, \overline{Q_1} \wedge \overline{Q_2}, \Sigma_1 \cup \Sigma_2 \rangle$. Le calcul de $int_{\alpha,\beta}(C_1, C_2)$ peut se faire sans accéder aux données. Ainsi, la phase importante et complexe en termes de temps de calcul est la phase de calcul des requêtes fréquentes.

2.3.3 Inclusion de requêtes conjonctives

Etant donné deux contextes $C_1 = \langle r_1, \overline{Q_1}, \Sigma_1 \rangle$ et $C_2 = \langle r_2, \overline{Q_2}, \Sigma_2 \rangle$ tels que $K = Var(r_1) = Var(r_2)$, soit deux requêtes $Q_1 \in \mathcal{Q}(C_1)$ et $Q_2 \in \mathcal{Q}(C_2)$. D'après la proposition 2.1, nous rappelons que $Sup(Q_2/r_2, S) \leq Sup(Q_1/r_1, S)$ si :

1. $(\forall K)(r_1 \Rightarrow r_2)$ est une formule valide, et
2. $(\forall K)(Q_2 \Rightarrow Q_1)$ est une formule valide.

Nous rappelons dans ce paragraphe que les conditions 1 et 2 ci-dessus sont équivalentes à des tests d'inclusion de requêtes conjonctives. Auparavant, nous rappelons quelques définitions et propriétés (voir [21] pour plus de détails).

Définition 2.5 - Inclusion de requêtes. *Etant donné deux requêtes conjonctives P et Q définies sur le même ensemble de prédicats de base, et de même ensemble de variables libres K, P* est incluse *dans Q, noté $P \sqsubseteq Q$, si pour tout ensemble de données S, $P(S) \subseteq Q(S)$.*

Il est important de noter que, en reprenant les notations de la définition ci-dessus, si $P \sqsubseteq Q$, alors l'implication $(\forall K)(P \Rightarrow Q)$ est logiquement valide et inversement. D'autre part, un résultat classique [21] permet de tester l'inclusion de requête grâce à la notion d'homomorphisme définie comme suit :

Définition 2.6 - Homomorphisme. *Soit deux requêtes conjonctives P et Q définies sur le même ensemble de prédicats de base et de même ensemble de variables libres K. Un* homomorphisme *$h : Q \Rightarrow P$ est une fonction totale de l'ensemble des termes de Q dans l'ensemble des termes de P, telle que si l'atome $p(x_1, \ldots, x_n)$ apparaît dans Q, alors $p(h(x_1), \ldots, h(x_n))$ apparaît dans P. De plus, h doit être l'identité sur les constantes.*

L'inclusion de requêtes conjonctives est alors caractérisée comme suit :

Proposition 2.3 *Etant donné deux requêtes conjonctives P et Q définies sur le même ensemble de prédicats de base, et de même ensemble de variables libres K, $P \sqsubseteq Q$ si et seulement si il existe un homomorphisme $h : Q \Rightarrow P$.*

Dans le cas général, la recherche d'un homomorphisme h entre deux requêtes P et Q est un problème NP-complet. Cependant, il est important de noter que ce test se fait indépendamment des données. Sa complexité ne dépend donc pas de la taille des ensembles de données sur lesquels on travaille.

Rappelons que toute vue v considérée dans notre contexte est définie par une seule règle Datalog de la forme $v(Y) :- p_1(X_1), \ldots, p_m(X_m)$. Nous notons par $body(v)$ la conjonction $p_1(X_1) \wedge \ldots \wedge p_m(X_m)$.
Etant donné un contexte d'extraction $C = \langle r, \overline{Q}, \Sigma \rangle$, soit $Q : (\exists Y)(r \wedge v_1\beta \wedge \ldots \wedge v_N\beta)$ une requête de $\mathcal{Q}(C)$. On appelle *expansion* de Q, et on note $E(Q)$, la requête obtenue en remplaçant r par $body(r)$ et chaque vue v_i apparaissant dans Q par $body(v_i)$. Plus précisément, $E(Q)$ est la requête $(\exists Y')(body(r) \wedge body(v_1)\beta \wedge \ldots \wedge body(v_N)\beta)$ où $Y' = Var(E(Q)) \setminus K$. Par extension, on note aussi $E(r)$ la requête $(\exists Y')(body(r))$ où $Y' = Var(E(r)) \setminus K$. Nous pouvons alors formuler la proposition suivante :

Proposition 2.4 *Soient deux contextes d'extraction $C_1 = \langle r_1, \overline{Q}_1, \Sigma_1 \rangle$ et $C_2 = \langle r_2, \overline{Q}_2, \Sigma_2 \rangle$ tels que $K = Var(r_1) = Var(r_2)$, et soient Q_1 et Q_2 deux requêtes de $\mathcal{Q}(C_1)$ et $\mathcal{Q}(C_2)$ respectivement.*

1. *L'implication $(\forall K)(r_1 \Rightarrow r_2)$ est valide si et seulement si $E(r_1) \sqsubseteq E(r_2)$.*
2. *L'implication $(\forall K)(Q_2 \Rightarrow Q_1)$ est valide si et seulement si $E(Q_2) \sqsubseteq E(Q_1)$.*

Exemple 2.6 *Dans le contexte de l'exemple 2.5, on peut montrer que l'implication* $(\forall Cid)(Q_2 \Rightarrow Q_1)$ *est valide en utilisant la proposition 2.4. En effet, les expansions* $E(Q_1)$ *et* $E(Q_2)$ *des requêtes* Q_1 *et* Q_2 *sont définies comme suit :*

$$\begin{aligned} E(Q_1): \quad & (\exists Z_1, Z_2, Pid, Z_3, Z_4)(Cust(Cid, Z_1, Z_2) \wedge Cust(Cid, Avocat, Blois) \wedge \\ & Sales(Cid, Pid, Z_3, Z_4) \wedge Prod(Pid, Fromage)) \\ E(Q_2): \quad & (\exists W_1, W_2, W_3, W_4, Pid, W_5, W_6, W_7)(Cust(Cid, W_1, W_2) \wedge \\ & Cust(Cid, Avocat, Blois) \wedge Cust(Cid, W_3, W_4) \wedge \\ & Sales(Cid, Pid, W_5, W_6) \wedge Store(W_5, W_7, W_4) \wedge Prod(Pid, Fromage)) \end{aligned}$$

Selon la proposition 2.4, l'implication $(\forall Cid)(Q_2 \Rightarrow Q_1)$ *est valide car la fonction* h_1 *définie par :*

$$h_1(Z_1) = W_1, h_1(Z_2) = W_2, h_1(Z_3) = W_5, h_1(Z_4) = W_6,$$

est un homomorphisme de $E(Q_1)$ *vers* $E(Q_2)$*, ce qui montre que* $E(Q_2) \sqsubseteq E(Q_1)$*.*

2.3.4 Comparaison de contextes d'extraction

Dans ce paragraphe, nous commençons par définir une relation sur les contextes d'extraction. Nous introduisons ensuite des notations permettant d'exprimer que l'intérêt ou le non-intérêt d'une requête peut être déterminé indépendamment des données, connaissant par ailleurs un ensemble déjà calculé de requêtes intéressantes ou non-intéressantes.

Nous comparons les contextes d'extraction selon une relation de préordre définie sur l'ensemble des contextes d'extraction.

Définition 2.7 Comparaison de contextes d'extraction.
Un contexte d'extraction $C_1 = \langle r_1, \overline{Q}_1, \Sigma_1 \rangle$ *est dit* **plus étroit** *que* $C_2 = \langle r_2, \overline{Q}_2, \Sigma_2 \rangle$*, noté :*

$$C_1 \leq C_2,$$

si $K = Var(r_1) = Var(r_2)$ *et* $(\forall K)(r_1 \Rightarrow r_2)$*.*

Cette relation de pré-ordre induit ainsi :

- Une relation d'équivalence $\approx$ définie par $C_1 \approx C_2$ si et seulement si $C_1 \leq C_2$ et $C_2 \leq C_1$, i.e. $K = Var(r_1) = Var(r_2)$, $(\forall K)(r_1 \Leftrightarrow r_2)$.
- Une relation d'ordre $\preceq$ sur les classes d'équivalence définies par $\overset{\circ}{C_1} \preceq \overset{\circ}{C_2}$ si et seulement si il existe $C_1 \in \overset{\circ}{C_1}$ et $C_2 \in \overset{\circ}{C_2}$ tels que $C_1 \leq C_2$.

Exemple 2.7 *Dans le cadre de l'exemple 2.5, regardons les relations entre les contextes d'extractions* C_1*,* C_2 *et* C_3*. Nous avons* $C_1 \approx C_2$ *car* C_1 *et* C_2 *partagent la même vue de référence* $AllCust$*. Par ailleurs,* $C_3 \leq C_2$ *car* $(\forall Cid)(r_3 \Rightarrow AllCust)$*.*

Notons $freq^-_\alpha(C, S)$ le complément de $freq_\alpha(C, S)$ dans $\mathcal{Q}(C)$, i.e. $freq^-_\alpha(C, S) = \mathcal{Q}(C) \setminus freq_\alpha(C, S)$.

Soient C_1 et C_2 deux contextes d'extraction telles que $C_1 \leq C_2$ et $\alpha_1 \leq \alpha_2$, soit X_1 un sous-ensemble de $freq^-_\alpha(C, S)$ et soit Q_2 une requête de $\mathcal{Q}(C_2)$. S'il existe une requête Q_1 dans X_1 telle que $(\forall K)(Q_2 \Rightarrow Q_1)$, alors, selon la proposition 2.1, on a $Sup(Q_2/r_2, S) \leq Sup(Q_1/r_1, S)$. De plus, on a $Sup(Q_1/r_1, S) < \alpha_1$ puisque $Q_1 \in freq^-_\alpha(C, S)$. Par conséquent, $Sup(Q_2/r_2, S) \leq Sup(Q_1/r_1, S) < \alpha_1 \leq \alpha_2$, ce qui montre que Q_2 est non fréquente, i.e. $Q_2 \in freq^-_\alpha(C, S)$.

Dans une telle situation, le non-intérêt de la requête Q_2 est démontré sans accéder aux données de S. Cette situation est notée $NotFreq(Q_2/X_1)$.

Supposons maintenant que $C_1 \geq C_2$ et $\alpha_1 \geq \alpha_2$. Soit X_1 un sous-ensemble de $freq_\alpha(C, S)$ et soit Q_2 une requête de $\mathcal{Q}(C_2)$. S'il existe une requête Q_1 dans X_1 telle que $(\forall K)(Q_1 \Rightarrow Q_2)$, alors on peut montrer comme ci-dessus que Q_2 est intéressante, i.e. $Q_2 \in freq_\alpha(C, S)$ sans accéder aux données de S, puisque $Sup(Q_2/r_2, S) \geq Sup(Q_1/r_1, S) \geq \alpha_1 \geq \alpha_2$. L'intérêt de la requête Q_2 sans accéder aux données est notée : $Freq(Q_2/X_1)$.

Dans leur formalisme général [63], Mannila et Toivonen ont montré que toute théorie $Th(\mathcal{L}, S, q)$ peut se représenter par ses *frontières positive ou négative* si le critère de qualité q est monotone par rapport à la relation de spécialisation sur $\mathcal{L}$. Dans ce paragraphe, nous montrons comment utiliser ces notions pour optimiser le calcul de $freq_\alpha(C_2, S)$ en utilisant un ensemble de requêtes fréquents $freq_\alpha(C_1, S)$ déjà calculé.

Soit un contexte d'extraction $C = \langle r, \overline{Q}, \Sigma \rangle$ où $K = Var(r)$, soit Q et Q' deux requêtes de $\mathcal{Q}(C)$. Notons la relation $(\forall K)(Q' \Rightarrow Q)$ par $Q' \succeq Q$, on a alors la définition suivante :

Définition 2.8 Frontières. *Etant donné un ensemble de données S et un contexte C, soit $freq_\alpha(C, S)$ l'ensemble des requêtes fréquentes. Les* frontières positive *et* négative *de $freq_\alpha(C, S)$, notées respectivement $Bd^+_\alpha(C, S)$ et $Bd^-_\alpha(C, S)$, sont définies par :*

$$Bd^+_\alpha(C, S) = \{Q \in freq_\alpha(C, S) \mid (\forall Q' \in \mathcal{Q}(C))(Q' \succ Q \Rightarrow Q' \in freq^-_\alpha(C, S))\}$$
$$Bd^-_\alpha(C, S) = \{Q \in freq^-_\alpha(C, S) \mid (\forall Q' \in \mathcal{Q}(C))(Q \succ Q' \Rightarrow Q' \in freq_\alpha(C, S))\}$$

La proposition suivante rappelle que, dans notre approche, la connaissance des frontières positive ou négative d'une théorie est suffisante pour la reconstruire complètement.

Proposition 2.5 *Etant donné un ensemble de données S et un contexte d'extraction C, soit $freq_\alpha(C, S)$ l'ensemble des requêtes fréquentes de $\mathcal{Q}(C)$ dans S. Si $Bd^+_\alpha(C, S)$ et $Bd^-_\alpha(C, S))$ désignent respectivement ses frontières positive et négative, alors :*
1. $freq_\alpha(C, S) = \{Q \in \mathcal{Q}(C) \mid (\exists Q' \in Bd^+_\alpha(C, S))(Q' \succeq Q)\}$
2. $freq^-_\alpha(C, S) = \{Q \in \mathcal{Q}(C) \mid (\exists Q' \in Bd^-_\alpha(C, S))(Q \succeq Q')\}$

Il est maintenant important de noter que la taille des frontières d'une théorie peut être beaucoup plus faible que celle de la théorie elle-même (voir expériences dans [63]). Ce point est essentiel pour la suite, car nous supposerons que nous stockons les frontières des ensembles de requêtes fréquentes déjà calculées.

Les propositions suivantes indiquent que pour évaluer l'intérêt d'une requête candidate, il suffit de la comparer avec les requêtes des frontières négative ou positive de l'ensemble de requêtes fréquentes précédent.

Proposition 2.6 *Etant donné un ensemble de données S et deux contextes d'extraction C_1 et C_2 de seuils respectifs α_1 et α_2 tels que $C_1 \leq C_2$ et $\alpha_1 \leq \alpha_2$, soit Q_2 une requête de $\mathcal{Q}(C_2)$ et $Bd^-_{\alpha_1}(C_1, S)$ la frontière négative de $freq_{\alpha_1}(C_1, S)$.*
Si $NotFreq(Q_2/freq^-_{\alpha_1}(C_1, S))$ est vrai, alors $NotFreq(Q_2/Bd^-_{\alpha_1}(C_1, S))$ est vrai.

Preuve :
Si $NotFreq(Q_2/freq^-_{\alpha_1}(C_1, S))$ est vrai, alors d'après la définition, $\exists Q_1 \in freq^-_{\alpha_1}(C_1, S)$ telle que $(\forall K)(Q_2 \Rightarrow Q_1)$.
D'après la proposition 2.5, pour toute requête $Q \in freq^-_\alpha(C_1, S)$, $\exists Q' \in Bd^-(freq^-_\alpha(C_1, S))$ telle que $(Q \succeq Q')$.
Cela est équivalent à : $\exists Q' \in Bd^-(freq^-_\alpha(C_1, S))$ telle que $(\forall K)(Q \Rightarrow Q')$.
Puisque $Q_1 \in freq^-_{\alpha_1}(C_1, S)$, $\exists Q' \in Bd^-(freq^-_\alpha(C_1, S))$ telle que $(\forall K)(Q_1 \Rightarrow Q')$.
On en déduit d'après la définition de $NotFreq(Q_2/freq^-_{\alpha_1}(C_1, S))$ que :

$\exists Q' \in Bd^-(freq^-_\alpha(C_1, S))$ telle que $(\forall K)(Q_2 \Rightarrow Q')$
$\Longrightarrow NotFreq(Q_2 \ / \ Bd^-(freq^-_\alpha(C_1, S))$ est vrai. $\diamond$

Proposition 2.7 *Etant donné un ensemble de données S et deux contextes d'extraction C_1 et C_2 de seuils respectifs α_1 et α_2 tels que $C_1 \geq C_2$ et $\alpha_1 \geq \alpha_2$, soit Q_2 une requête de $\mathcal{Q}(C_2)$ et $Bd^+_{\alpha_1}(C_1(S))$ la frontière positive de $freq_{\alpha_1}(C_1, S)$.*
Si $Freq(Q_2/freq_{\alpha_1}(C_1, S))$ est vrai, alors $Freq(Q_2/Bd^+_{\alpha_1}(C_1, S))$ est vrai.

Preuve :
Si $Freq(Q_2/freq_{\alpha_1}(C_1, S))$ est vrai, alors d'après la définition, $\exists Q_1 \in freq_{\alpha_1}(C_1, S)$ telle que $(\forall K)(Q_1 \Rightarrow Q_2)$.
D'après la proposition 2.5, pour toute requête $Q \in freq_{\alpha_1}(C_1, S)$, $\exists Q' \in Bd^+(freq_{\alpha_1}(C_1, S))$ telle que $Q' \succeq Q$.
Cela est équivalent à : $\exists Q' \in Bd^+(freq_{\alpha_1}(C_1, S))$ telle que $Q' \Rightarrow Q$.
Puisque $Q_1 \in freq_{\alpha_1}(C_1, S)$, $\exists Q' \in Bd^+(freq_{\alpha_1}(C_1, S))$ telle que $Q' \Rightarrow Q_1$.
On en déduit d'après la définition de $Freq(Q_2/freq_{\alpha_1}(C_1, S))$ que :
$\exists Q' \in Bd^+(freq_{\alpha_1}(C_1, S))$ telle que $(\forall K)(Q' \Rightarrow Q_2)$
$\Longrightarrow Freq(Q_2/Bd^+_{\alpha_1}(C_1, S))$ est vrai. $\diamond$

L'algorithme itératif que nous proposons est basé sur les propositions 2.6 et 2.7.

2.3.5 Algorithme itératif

L'algorithme itératif que nous proposons (voir Figure III.1) est une extension de l'algorithme générique de Mannila et Toivonen. Les phases d'évaluation et de génération se caractérisent dans notre contexte comme suit :

- L'évaluation des supports des requêtes candidates d'un contexte C_i (ligne 4) peut se faire en une seule passe sur la jointure de la vue de référence r avec toutes les autres vues ou prédicats de bases de données de la requête de base.
- Pour chaque ligne de la table jointe $(r \bowtie v_1 \bowtie \ldots \bowtie v_n)$ où v_i est une vue ou un prédicat de base de données de la requête de base, il est possible d'utiliser les mêmes techniques de hachage que celles proposées dans *Apriori* [2] pour tester si le support d'une requête candidate doit être incrémenté ou non de 1. Toutefois, dans notre cadre, si K désigne l'ensemble des variables de la vue de référence, il ne faut pas incrémenter le support deux fois pour deux lignes distinctes de même valeur de K. Le parcours de la jointure selon l'ordre croissant des valeurs de K permet de détecter simplement les doublons pour chaque requête candidate.
- La génération des candidats (lignes 6 et 7) peut être réalisée selon les mêmes principes que ceux utilisés dans *Apriori* [2].

Notons que notre nouvelle phase d'élagage (ligne 8 à 12) peut nécessiter de comparer des requêtes candidates Q aux requêtes de Bd^+_i et Bd^-_i. Pour optimiser ces comparaisons, à savoir ces tests d'inclusion, on pourrait utiliser les propriétés suivantes :

- S'il existe un homomorphisme de P dans Q, alors il existe un homomorphisme de chaque atome de P dans chaque atome de Q,
- S'il existe un homomorphisme de P dans Q et si P est une spécialisation de P', alors il existe un homomorphisme de P' dans Q.

Entrée : Un contexte d'extraction C_{N+1} et un seuil α_{N+1} avec $C_{N+1} = \langle r_{N+1}, \overline{Q}_{N+1}, \Sigma_{N+1} \rangle$
Sortie : $freq_\alpha(C_{N+1})$
Utilisation : un ensemble de frontières $Bd_i^- = Bd_{\alpha_i}^-(C_i, S)$ et $Bd_i^+ = Bd_{\alpha_i}^+(C_i, S)$ où $i = 1, \ldots, N$.

1. $F_i = \emptyset$ pour $i = 1 \ldots |\Sigma_{N+1}|$, $E_1 = \{\overline{Q}_{N+1}\}$ et $i = 1$
2. **While** E_i non vide **Do**
3. // **Phase d'évaluation des candidats**
4. $F_i = F_i \cup \{Q \in E_i \mid Sup(Q/r_{N+1}, S) \geq \alpha_{N+1}\}$
5. // **Phase de génération des candidats**
6. $E_{i+1} = \{Q' = Q\sigma \mid Q \in F_i, \sigma \in \Sigma_{N+1} \text{ and } Q' \neq Q\sigma\}$
7. $E_{i+1} = \{Q' \in E_{i+1} \mid (\forall Q \in \mathcal{Q}(C_{N+1}))((\exists \sigma \in \Sigma_{N+1})(Q' = Q\sigma) \Rightarrow Q \in F_i)\}$
8. // **Nouvelle phase d'élagage**
9. **For all** $j = 1 \ldots N$ **Do**
10. **If** $C_j \leq C_{N+1}$ **and** $\alpha_j \leq \alpha_{N+1}$ **Then** Elaguer de E_{i+1} toutes les requêtes Q telles $NotFreq(Q/Bd_i^-)$ soit vrai
11. **If** $C_j \geq C_{N+1}$ **and** $\alpha_j \geq \alpha_{N+1}$ **Then** Elaguer de E_{i+1} et Insérer dans F_{i+1} toutes les requêtes Q telles que $Freq(Q/Bd_i^+)$ soit vrai
12. **End**
13. $i = i + 1$
14. **End**
15. **Retourner** $\bigcup_i F_i$

FIG. III.1 – Calcul itératif des ensembles de requêtes fréquentes

2.4 Conclusion

Dans cette partie, nous avons proposé une approche pour l'extraction de requêtes fréquentes basée sur un formalisme logique proche de celui de [25, 27], mais étendu à l'utilisation de vues. Nous avons d'une part, montré que les vues permettent d'exprimer simplement des requêtes complexes, et d'autre part, comment le calcul de l'ensemble des requêtes fréquentes définies par un contexte d'extraction peut être accéléré en utilisant les ensembles de requêtes fréquentes déjà calculés et stockés.

Le principal problème, avec cette approche, est que, pour toute requête candidate, on doit tester son inclusion dans toutes les requêtes de la frontière négative de tous les contextes d'extraction comparables déjà calculés. Ceci va être évidemment très coûteux, d'autant plus que la recherche d'un homomorphisme pour tester l'inclusion est NP-complet.

C'est ainsi que, pour ne plus avoir ce problème de complexité (lié à la recherche d'homomorphisme), nous allons, dans la partie suivante, introduire un langage permettant de combiner des contextes d'extraction. Dans ce cas, quand on sait qu'un contexte C_{12} est obtenu (par exemple) par jointure entre deux contextes C_1 et C_2, on sait (sans recherche d'homomorphisme) que toute requête candidate Q_{12} (de $\mathcal{Q}(C_{12})$) est incluse dans une requête Q_1 (de $\mathcal{Q}(C_1)$) et une requête Q_2 (de $\mathcal{Q}(C_2)$). Comme Q_{12} est incluse dans Q_1 et Q_2, si on sait que Q_1 ou Q_2 n'est pas intéressante, alors Q_{12} n'est pas intéressante.

3 Composition de contextes d'extraction pour une extraction efficace de règles d'associations

Les travaux exposés dans cette partie ont été présentés dans [29, 30].

3.1 Introduction

Dans la partie précédente, nous avons présenté une approche logique d'extraction itérative de requêtes. Nous avons vu le nombre important de tests d'inclusion (liés à la recherche d'homomorphisme) qu'il faut effectuer pour trouver l'intérêt ou le non-intérêt de toute requête candidate, ce qui représente un coût important.
Dans cette partie, nous allons présenter une autre approche pour l'extraction itérative de requêtes et de règles d'associations appelées *règles d'association multi-dimensionnelles*, i.e. des règles extraites à partir d'une ou de plusieurs tables.
Le contexte d'extraction que nous proposons permet de spécifier la forme des règles à extraire. L'extraction itérative est ici obtenue par l'introduction d'un langage de combinaison de contextes d'extraction en utilisant les opérations de l'algèbre relationnelle. Plus précisément, nous montrons comment optimiser le calcul de l'ensemble des requêtes fréquentes définies par un contexte d'extraction lorsque ce dernier est défini par composition de contextes d'ensembles de requêtes fréquentes déjà calculées. Nous allons montrer dans cette partie que l'introduction de ce langage permet d'éviter les tests d'inclusion nécessaires dans l'approche décrite à la partie précédente. Par conséquent, les problèmes d'efficacité liés à ces tests sont éliminés ici.

3.1.1 Règles d'association et contextes d'extraction

Différents formalismes peuvent être utilisés pour représenter les règles d'association multi-dimensionnelles et spécifier leurs formes. Alors que la plupart de ces formalismes ([27, 52, 89]) sont basés sur la logique de premier ordre et sur Datalog, nous allons utiliser l'algèbre relationnelle pour définir les règles d'association et les ensembles de requêtes définis par des contextes d'extraction.

Dans notre approche, un contexte d'extraction, ou simplement un contexte, spécifie la forme des règles à extraire. Plus précisément, un contexte spécifie les objets qui sont le sujet de l'extraction, les attributs sur ces objets et les conditions sur ces attributs. Nous allons clarifier ces différents concepts à travers l'exemple courant que nous avons utilisé dans la partie précédente et que nous utiliserons ici aussi.

Exemple 3.1 *Rappelons qu'il s'agit d'une base de données DBSALES contenant les différentes tables :*

- *Cust(Cid,Cjob,Caddr), où Cid Cjob et Caddr sont les identifiants, les professions et les adresses des clients respectivement,*
- *Prod(Pid,Ptype), où Pid et Ptype sont respectivement les identifiants et les types de produits,*
- *Store(Sid,Sname,Saddr), où Sid, Sname et Saddr sont respectivement les identifiants, les noms et adresses des clients,*
- *Sales(Cid,Pid,Sid,Date), où un tuple $< c, p, s, d >$ dans la table Sales signifie que le client c a acheté le produit p dans le magasin s à la date d.*

Supposons que l'on s'intéresse aux clients dont les identifiants sont stockés dans la table Cust, on considère alors l'expression relationnelle suivante : $AllCust = \pi_{Cid}(Cust)$.
Supposons maintenant que l'on s'intéresse aux attributs professions, adresses et types de produit que ces clients ont achetés. Ceci peut être spécifié par une expression relationnelle qui est la

jointure des tables Cust, Sales et Prod, notée $B_1 = Cust \bowtie Sales \bowtie Prod$.
De plus, si on s'intéresse seulement aux règles d'association concernant les enseignants ou les avocats, et qui achètent du thé ou du lait, alors on peut considérer l'ensemble des conditions de sélection Σ_1 *défini par* $\Sigma_1 = \{Cjob = Enseignant, Cjob = Avocat, Ptype = Thé, Ptype = Lait\}$.

Le triplet $\mathcal{C}_1 = \langle AllCust, B_1, \Sigma_1 \rangle$ *est un exemple de contexte d'extraction (ou tout simplement de contexte) . De plus, AllCust,* B_1 *et* Σ_1 *sont appelés respectivement la référence, la base et le domaine de* C_1. *Etant donné un contexte* $\mathcal{C}_1 = \langle AllCust, B_1, \Sigma_1 \rangle$, *la règle suivante est un exemple de règle d'association :*

$$AllCust \bowtie \sigma_{Cjob=Enseignant}(B_1) \Rightarrow AllCust \bowtie \sigma_{Ptype=Thé}(B_1)$$

Cette règle signifie intuitivement que si un client est un enseignant, alors il achète du thé. De façon plus formelle, puisque Cid est l'attribut de la référence AllCust, on considère que cette règle est satisfaite dans l'instance I de DBSales si $(\pi_{Cid}(AllCust \bowtie \sigma_{Cjob=Enseignant}(B_1)))(I) \subseteq (\pi_{Cid}(AllCust \bowtie \sigma_{Ptype=Thé}(B_1)))(I)$, *i.e. si chaque identifiant de client dans la réponse de* $\pi_{Cid}(AllCust \bowtie \sigma_{Cjob=Enseignant}(B_1))$ *dans I, se trouve aussi dans la réponse de* $\pi_{Cid}(AllCust \bowtie \sigma_{Ptype=Thé}(B_1))$ *dans I.*

Plus généralement, un contexte est un triplet $\mathcal{C} = \langle R, B, \Sigma \rangle$, où R et B sont deux expressions relationnelles et Σ est un ensemble de conditions de sélections. Etant donné un contexte $\mathcal{C} = \langle R, B, \Sigma \rangle$, on considère les deux types de motifs suivants :

1. Une *requête de* C de la forme $R \bowtie \sigma_S(B)$ où S est une conjonction de conditions de sélections dans Σ.
2. Une *règle d'association de* C est une expression de la forme $q_1 \Rightarrow q_2$ où q_1 and q_2 sont des requêtes de $\mathcal{C}$. De plus, on dit que la règle $q_1 \Rightarrow q_2$ est satisfaite dans une instance I si $(\pi_K(q_1))(I) \subseteq (\pi_K(q_2))(I)$, où K est l'ensemble des attributs de la table de référence R.

Comme avec les autres approches, étant donné un contexte C, un seuil minimum de support et un seuil minimum de confiance, le problème est de trouver l'ensemble de toutes les règles d'association intéressantes de $\mathcal{C}$. Ce problème est résolu en deux étapes :

1. Trouver toutes les requêtes fréquentes de $\mathcal{C}$.
2. Générer toutes les règles intéressantes de $\mathcal{C}$ à partir des requêtes fréquentes de $\mathcal{C}$.

La performance de l'extraction est surtout déterminée par la première étape. Lors de l'implantation de notre approche, nous pouvons utiliser n'importe lequel des algorithmes par niveau existants ([2, 17, 74]) pour calculer les requêtes fréquentes d'un contexte. Par la suite, nous allons montrer comment le calcul des requêtes fréquentes d'un contexte peut être optimisé en utilisant ces algorithmes, si on suppose que les résultats des extractions précédentes sont stockés.

3.1.2 Composition de contextes d'extraction

L'idée principale de l'approche que nous présentons dans cette partie est la suivante : Supposons qu'un certain nombre d'extractions ont déjà été effectuées et que leurs requêtes fréquentes sont stockées. Alors, toute nouvelle extraction dont le contexte est défini par composition des contextes des anciennes extractions, peut être calculée de façon plus efficace que si elle avait été définie sans référence aux anciennes extractions. Pour cela, nous définissons des opérations sur les contextes similaires aux opérations de l'algèbre relationnelle, et nous étudions leurs propriétés. L'exemple suivant montre comment les contextes sont composés.

Exemple 3.2 *Dans la base de données DBSales de l'exemple 3.1, supposons qu'on veut trouver des relations entre les professions des clients vivant à Paris et les types de produits qu'ils achètent. En utilisant le contexte $\mathcal{C}_1$ défini précédemment, un contexte noté $\mathcal{C}_1^* = \sigma^b_{Caddr=Paris}(\mathcal{C}_1)$, peut être défini par application de la condition de sélection $(Caddr = Paris)$ sur la base de $\mathcal{C}_1$. Ainsi, $\mathcal{C}_1^* = \langle AllCust, \sigma_{Caddr=Paris}(B_1), \Sigma_1\rangle$.*

Etant donné le nouveau contexte $\mathcal{C}_1^*$, soit $q_1^* = AllCust \bowtie \sigma_S(\sigma_{Caddr=Paris}(B_1))$ une requête de $\mathcal{C}_1^*$. On constate que la réponse de la requête q_1^* est toujours incluse dans celle de $q_1 = AllCust \bowtie \sigma_S(B_1)$ de $\mathcal{C}_1$. Plus généralement, on peut se rendre compte que toute requête q_1^* de $\mathcal{C}_1^*$ est incluse (dans le sens de l'inclusion de requête [91]) dans une requête q_1 de $\mathcal{C}_1$.
Donc, le support de toute requête q_1^* de $\mathcal{C}_1^*$ est plus petit ou égal au support d'une requête q_1 de $\mathcal{C}_1$. Ainsi, si une requête q_1^* de $\mathcal{C}_1^*$ est fréquente, alors elle est incluse dans une requête fréquente q_1 de $\mathcal{C}_1$. Inversement, si une requête q_1^* de $\mathcal{C}_1^*$ n'est pas incluse dans une requête fréquente q_1 de $\mathcal{C}_1$, alors elle n'est pas fréquente.
Par conséquent, si on suppose que les requêtes fréquentes de $\mathcal{C}_1$ sont stockées, cette propriété peut être utilisée pour élaguer certaines requêtes candidates de $\mathcal{C}_1^*$ lors du calcul de ses requêtes fréquentes. De plus, cette nouvelle phase d'élagage peut être effectuée sans accès à la base de données, d'où l'optimisation du calcul des requêtes fréquentes de $\mathcal{C}_1^*$.

Supposons maintenant que l'on s'intéresse aux règles concernant les professions des clients et les noms des magasins où ils font leurs achats. Si on s'intéresse seulement aux règles impliquant les enseignants ou les avocats et concernant les magasins Carrefour ou Ikea, alors on peut définir un contexte $\mathcal{C}_2 = \langle AllCust, B_2, \Sigma_2\rangle$, avec la même référence que $\mathcal{C}_1$, où $B_2 = Cust \bowtie Sales \bowtie Store$ et $\Sigma_2 = \{Cjob = Enseignant, Cjob = Avocat, Sname = Carrefour, Sname = Ikea\}$.
Supposons que les requêtes fréquentes de C_2 sont déjà calculées et stockées.
Alors, en se basant sur $\mathcal{C}_1$ et $\mathcal{C}_2$, un nouveau contexte $\mathcal{C}_{12}$ peut être défini par la jointure des bases et l'union des domaines de $\mathcal{C}_1$ et $\mathcal{C}_2$. En d'autres termes, $\mathcal{C}_{12}$ est défini comme suit :

$$\mathcal{C}_{12} = \langle AllCust, B_1 \bowtie B_2, \Sigma_1 \cup \Sigma_2\rangle$$

Ce nouveau contexte permet de considérer les relations entre les professions des clients, les types de produits qu'ils achètent et les noms des magasins où les achats ont été effectués. Comme nous le verrons plus tard, le calcul des requêtes fréquentes de $\mathcal{C}_{12}$ peut être optimisé, en se basant sur les requêtes fréquentes de $\mathcal{C}_1$ et $\mathcal{C}_2$.

Par la suite, nous montrerons comment l'exemple ci-dessus peut être généralisé et comment le calcul des requêtes fréquentes de contextes composés peut être optimisé.

3.1.3 Travaux existants

Nous avons présenté dans le deuxième chapitre de ce mémoire dans le partie "Extraction itérative de motifs" différents langages de requêtes basés sur SQL pour la définition de requêtes d'extraction ([44, 49, 69]), mais aucun de ces langages ne permet, à notre connaissance, de combiner des requêtes d'extraction en se basant sur les propriétés de l'algèbre relationnelle.
D'autre part, nous avons présenté différentes techniques ([5, 69, 72, 51]) qui stockent et réutilisent les résultats de requêtes d'extraction pour optimiser de nouvelles requêtes d'extraction. Cependant, ces approches comparent seulement des requêtes d'extraction définies sur une seule référence et une seule base (en utilisant notre terminologie), et elles ne proposent pas d'opérations de compositions de contextes comme nous.
Dans l'approche que nous présentons, la base de données source contient plusieurs tables. Par conséquent, nous pouvons spécifier et comparer des contextes d'extraction avec différentes bases ou différentes références. Rappelons que ces comparaisons permettent d'optimiser le calcul de

la réponse à une nouvelle requête d'extraction. Finalement, dans notre technique de stockage, seules les frontières positives des résultats des requête d'extraction seront stockées.

La suite de cette partie est organisée comme suit : A la section 3.2, nous donnons la définition formelle de contexte d'extraction introduit précédemment, les définitions et propriétés du support et de la confiance. Les opérations pour la composition de contextes d'extraction sont étudiées à la section 3.3, et à la section 3.4, nous montrons comment utiliser les propriétés de la section 3.3 pour l'optimisation de l'extraction de motifs. La section 3.5 conclut et ouvre des perspectives.

3.2 Contextes d'extraction et ensemble de requêtes fréquentes

3.2.1 Définitions et notations

Le formalisme utilisé ici est basé sur le modèle relationnel ([91]). Rappelons qu'un schéma de base de données relationnelles est un ensemble de relations, dont chacune est associée à un ensemble d'attributs. On appelle schéma de la relation R l'ensemble des attributs associés à R et on le note $sch(R)$. Pour chaque attribut A, les valeurs possibles de A appartiennent à un ensemble spécifique de valeurs, appelé le domaine de A et noté $dom(A)$.

Par la suite, nous supposons que l'on dispose d'un schéma fixe de bases de données, $DB = \{R_1, R_2, ..., R_n\}$ et on appelle instance de DB tout ensemble de relations $I = \{r_1, r_2, ..., r_n\}$, où r_i est une relation sur $sch(R_i)$, pour tout $i = 1, 2, ..., n$. Etant donné une expression relationnelle, ou une requête q sur DB, on appelle schéma de q l'ensemble des attributs qui définissent q, et on le note $sch(q)$. De plus, on note $q(I)$ la réponse de q dans I.

Comme dans [21], si q_1 et q_2 sont deux requêtes telles que $sch(q_1) = sch(q_2)$, on dit que q_1 est *incluse* dans q_2, ou que q_1 est *plus spécifique* que q_2, et on note $q_1 \sqsubseteq q_2$, si pour toute instance I, on a $q_1(I) \subseteq q_2(I)$. Les requêtes q_1 et q_2 sont dites *équivalentes*, $q_1 \equiv q_2$, si $q_1 \sqsubseteq q_2$ et $q_2 \sqsubseteq q_1$.

3.2.2 Contextes d'extraction

Dans notre approche, la forme des règles à extraire est spécifiée par un *contexte d'extraction.*

Définition 3.1 Contexte d'extraction *Un contexte d'extraction $\mathcal{C}$, ou simplement un contexte, est un triplet $\mathcal{C} = \langle R, B, \Sigma \rangle$ où :*

- *R est une expression relationnelle, appelée la référence de $\mathcal{C}$,*
- *B est une expression relationnelle telle que $sch(R) \subseteq sch(B)$, appelée la base de $\mathcal{C}$.*
- *Σ est un ensemble de conditions de sélections atomiques, appelé le domaine de $\mathcal{C}$. Ces conditions de sélections sont de la forme $(A = a)$ où A est un attribut de B qui ne figure pas parmi les attributs de R, et a est une constante dans $dom(A)$.*

Rappelons que la notion de référence est trés importante, puisqu'elle spécifie les valeurs qui sont l'objet du comptage lors de la recherche des motifs fréquents (voir définition 3.3).

Nous donnons maintenant les définitions des différents types de motifs auxquels on s'intéresse.

Définition 3.2 Espace de recherche. *Etant donné un contexte $\mathcal{C} = \langle R, B, \Sigma \rangle$*

1. *Σ^* représente l'ensemble de toutes les conditions de sélections de la forme $(A_1 = a_1) \wedge \ldots \wedge (A_n = a_n)$, où $(A_i = a_i)$ avec $(i = 1 \ldots n)$ est une condition de sélection atomique dans Σ.*
2. *$\mathcal{Q}(\mathcal{C})$ représente l'ensemble de toutes les requêtes de la forme $R \bowtie \sigma_S(B)$ où $S \in \Sigma^*$. Les requêtes dans $\mathcal{Q}(\mathcal{C})$ sont appelées requêtes candidates de $\mathcal{C}$, ou simplement requêtes de $\mathcal{C}$.*
3. *$\mathcal{R}(\mathcal{C})$ représente l'ensemble de toutes les règles de la forme $q_1 \Rightarrow q_2$ où $q_1 = R \bowtie \sigma_{S_1}(B)$ et $q_2 = R \bowtie \sigma_{S_1 \wedge S_2}(B)$ sont des requêtes de $\mathcal{C}$. Les règles de $\mathcal{R}(\mathcal{C})$ sont appelées règles candidates de $\mathcal{C}$, ou simplement règles de $\mathcal{C}$. De plus, on dit qu'une règle $q_1 \Rightarrow q_2$ est satisfaite dans une instance I si $(\pi_K(q_1))(I) \subseteq (\pi_K(q_2))(I)$, où $K = sch(R)$.*

Comme $S_1 \wedge S_2$ est plus spécifique que S_1, on a toujours $\pi_K(q_2) \sqsubseteq \pi_K(q_1)$. Par conséquent, la règle $q_1 \Rightarrow q_2$ est satisfaite dans I si et seulement si $(\pi_K(q_1))(I) = (\pi_K(q_2))(I)$. D'autre part, on pourrait considérer les règles plus générales de la forme $q_1 \Rightarrow q_2$ où $q_1 = R \bowtie \sigma_{S_1}(B)$ et $q_2 = R \bowtie \sigma_{S_2}(B)$ sont des requêtes de $\mathcal{C}$. Cependant, comme nous le verrons dans la section 3.2.3, la confiance de telles règles ne serait pas le rapport des supports de $q_1 \cap q_2$ et de q_1.

3.2.3 Support et confiance

Toutes les mesures standard d'évaluation des règles d'association comme indiqué dans [58], peuvent être définies dans notre approche. Nous nous contenterons cependant de définir le support et la confiance d'une règle.

Définition 3.3 Support d'une requête. *Soit $\mathcal{C} = \langle R, B, \Sigma \rangle$ un contexte et q une requête telle que $sch(R) \subseteq sch(q)$. Pour toute instance I, le support de q relativement à $\mathcal{C}$ et I, noté $Sup(q \mid \mathcal{C}, I)$, est défini comme suit :*

$$Sup(q \mid \mathcal{C}, I) = \frac{|(\pi_K(q))(I)|}{|R(I)|}, \text{ où } K = sch(R).$$

Etant donné un seuil de support α, on note $freq_\alpha(\mathcal{C}, I)$ l'ensemble de toutes les requêtes dans $\mathcal{Q}(\mathcal{C})$ dont le support relativement à $\mathcal{C}$ et I est supérieur ou égal à α, i.e.

$$freq_\alpha(\mathcal{C}, I) = \{q \in \mathcal{Q}(\mathcal{C}) \mid Sup(q \mid \mathcal{C}, I) \geq \alpha\}.$$

Les requêtes dans $freq_\alpha(\mathcal{C}, I)$ sont appelées *α-frequentes* dans I, ou simplement fréquentes si α et I sont déterminés.
La proposition suivante établit la monotonie du support par rapport à l'inclusion de requêtes, ce qui généralise la propriété de base de la monotonie que l'on utilise pour les règles d'association standards [2].

Proposition 3.1 *Soit $\mathcal{C}_1 = \langle R_1, B_1, \Sigma_1 \rangle$ et $\mathcal{C}_2 = \langle R_2, B_2, \Sigma_2 \rangle$ deux contextes tels que $K = sch(R_1) = sch(R_2)$, soit q_1 et q_2 deux requêtes de $\mathcal{Q}(\mathcal{C}_1)$ et $\mathcal{Q}(\mathcal{C}_2)$ respectivement.*
Si $R_1 \sqsubseteq R_2$ et $\pi_K(q_2) \sqsubseteq \pi_K(q_1)$, alors pour toute instance I, on a :

$$Sup(q_2 \mid \mathcal{C}_2, I) \leq Sup(q_1 \mid \mathcal{C}_1, I).$$

Preuve :
Par définition $Sup(q_2 \mid \mathcal{C}_2, I) = \frac{|(\pi_K(q_2))(I)|}{|R_2(I)|}$.
Puisque $\pi_K(q_2) \sqsubseteq \pi_K(q_1)$, alors $(\pi_K(q_2))(I) \subseteq (\pi_K(q_1))(I)$, ce qui implique que $|(\pi_K(q_2))(I)| \leq |(\pi_K(q_1))(I)|$. On a alors :
$Sup(q_2 \mid \mathcal{C}_2, I) \leq \frac{|(\pi_K(q_1 2))(I)|}{|R_2(I)|}$.
De même $R_1 \sqsubseteq R_2$ implique que $|R_1(I)| \leq |R_2(I)|$, i.e. $1/|R_2(I)| \leq 1/|R_1(I)|$. Par conséquent : $Sup(q_2 \mid \mathcal{C}_2, I) \leq \frac{|(\pi_K(q_1))(I)|}{|R_1(I)|}$, i.e. $Sup(q_2 \mid \mathcal{C}_2, I) \leq Sup(q_1 \mid \mathcal{C}_1, I)$. $\diamond$

En se basant sur cette proposition et sur les propriétés de l'algèbre relationnelle, on peut facilement démontrer ce qui suit :

Corollaire 3.1 *Soit $\mathcal{C}_1 = \langle R_1, B_1, \Sigma_1 \rangle$ et $\mathcal{C}_2 = \langle R_2, B_2, \Sigma_2 \rangle$ deux contextes tels que $K = sch(R_1) = sch(R_2)$, soit q_1 et q_2 deux requêtes de $\mathcal{Q}(\mathcal{C}_1)$ et $\mathcal{Q}(\mathcal{C}_2)$ respectivement, et soit S une condition de sélection dans Σ_1^*. Alors, pour toute instance I, on a :*
1. $Sup(\sigma_S(q_1) \mid \mathcal{C}_1, I) \leq Sup(q_1 \mid \mathcal{C}_1, I)$,

2. $Sup(q_1 \bowtie q_2 \mid \mathcal{C}_i, I) \leq Sup(q_i \mid \mathcal{C}_i, I)$ *pour* $i = 1, 2$.
De plus, si $sch(q_1) = sch(q_2)$ *alors on a aussi :*
3. $Sup(q_1 \cap q_2 \mid \mathcal{C}_i, I) \leq Sup(q_i \mid \mathcal{C}_i, I)$ *pour* $i = 1, 2$,
4. $Sup(q_2 - q_1 \mid \mathcal{C}_2, I) \leq Sup(q_2 \mid \mathcal{C}_2, I)$,
5. $Sup(q_2 \cup q_1 \mid \mathcal{C}_i, I) \geq Sup(q_i \mid \mathcal{C}_i, I)$ *pour* $i = 1, 2$.

Etant donné un contexte $\mathcal{C} = \langle R, B, \Sigma \rangle$, la confiance d'une règle d'association est définie comme suit :

Définition 3.4 Confiance d'une règle. *Soit* $\mathcal{C} = \langle R, B, \Sigma \rangle$ *un contexte et* q_1 *et* q_2 *deux requêtes de* $\mathcal{Q}(\mathcal{C})$. *Pour toute instance* I, *la confiance de la règle* $\mathcal{R} : \ q_1 \Rightarrow q_2$ *relativement à* $\mathcal{C}$ *et* I, *notée* $Conf(\mathcal{R} \mid \mathcal{C}, I)$, *est le rapport suivant :*

$$Conf(\mathcal{R} \mid \mathcal{C}, I) = \frac{|(\pi_K(q_1))(I) \cap (\pi_K(q_2))(I)|}{|(\pi_K(q_1))(I)|}, \ \textit{où} \ K = sch(R).$$

Etant donné un seuil de support α et un seuil de confiance β, on note $int_{\alpha,\beta}(\mathcal{C}, I)$ l'ensemble de toutes les règles $q_1 \Rightarrow q_2$ dans $\mathcal{R}(\mathcal{C})$ telles que q_1 et q_2 soient des requêtes fréquentes dans $freq_\alpha(\mathcal{C}, I)$, et $Conf(q_1 \Rightarrow q_2 \mid \mathcal{C}, I) \geq \beta$, i.e.

$$int_{\alpha,\beta}(\mathcal{C}, I) = \{q_1 \Rightarrow q_2 \in \mathcal{R}(\mathcal{C}) \mid q_1, q_2 \in freq_\alpha(\mathcal{C}, I) \ et \ Conf(q_1 \Rightarrow q_2 \mid \mathcal{C}, I) \geq \beta\}.$$

Les règles dans $int_{\alpha,\beta}(\mathcal{C}, I)$ sont appelées (α, β)-*intéressantes* dans I, ou simplement intéressantes lorsque α, β et I sont clairement définis.
La proposition suivante énonce les propriétés de base de la confiance.

Proposition 3.2 *Soit* $\mathcal{C} = \langle R, B, \Sigma \rangle$ *un contexte et* $\mathcal{R} : \ q_1 \Rightarrow q_2$ *une règle dans* $\mathcal{R}(\mathcal{C})$. *Alors, pour toute instance* I *:*

1. $Conf(\mathcal{R} \mid \mathcal{C}, I) = 1$ *si et seulement si* $\mathcal{R}$ *est satisfaite dans* I.
2. $Conf(\mathcal{R} \mid \mathcal{C}, I) = Sup(q_2 \mid \mathcal{C}, I) \ / \ Sup(q_1 \mid \mathcal{C}, I)$.

Notons que la première partie de la proposition 3.2 est valable pour toute règle, alors que la seconde partie est vraie seulement si q_2 est plus spécifique que q_1 (ce qui est vrai pour toute règle de notre contexte). Soulignons que cette seconde partie est importante du point de vue du calcul : elle montre que la confiance d'une règle peut être calculée sans accéder à la base de données (en supposant que les supports de toutes les requêtes fréquentes sont disponibles). Ainsi, on voit que la phase la plus importante de l'extraction de règles d'association est le calcul des requêtes fréquentes. Dans la section suivante, nous allons voir comment la composition de contextes permet d'optimiser le calcul des requêtes fréquentes.

3.3 Composition de contextes

Les opérations définies dans cette section sont basées sur celles de l'algèbre relationnelle [91] et permettent la définition de nouvelles requêtes d'extraction à partir d'anciennes.

3.3.1 Manipulation et comparaison d'ensembles de requêtes

Nous allons d'abord étendre les opérateurs usuels de l'algèbre relationnelle à des ensembles de requêtes de même schéma, et nous définissons un nouvel opérateur que nous appelons la *réduction*.

Définition 3.5 - Opérations sur les ensembles de requêtes. *Soit $\mathcal{Q}$ un ensemble de requêtes de même schéma Sch.*

1. *Soit X un sous-ensemble de Sch, la* projection *de $\mathcal{Q}$ sur X, notée $\pi_X(\mathcal{Q})$, est définie par :* $\pi_X(\mathcal{Q}) = \{\pi_X(q) \mid q \in \mathcal{Q}\}$.
2. *Etant donnée une condition de sélection S sur les attributs de Sch, la* sélection *de $\mathcal{Q}$ sur S, notée $\sigma_S(\mathcal{Q})$, est définie par :* $\sigma_S(\mathcal{Q}) = \{\sigma_S(q) \mid q \in \mathcal{Q}\}$.
3. *Etant donnée une condition de sélection S sur les attributs de Sch, la* réduction *de $\mathcal{Q}$ sur S, notée $\tau_S(\mathcal{Q})$, est définie par :* $\tau_S(\mathcal{Q}) = \{q \in \mathcal{Q} \mid \sigma_S(q) \equiv q\}$. *Intuitivement $\tau_S(\mathcal{Q})$ représente l'ensemble des requêtes de $\mathcal{Q}$ qui restent inchangées après la sélection sur S.*

Soit $\mathcal{Q}_1$ un ensemble de requêtes de schéma Sch_1 et $\mathcal{Q}_2$ un ensemble de requêtes de schéma Sch_2.

4. *La jointure de $\mathcal{Q}_1$ et $\mathcal{Q}_2$, notée $\mathcal{Q}_1 \bowtie \mathcal{Q}_2$, est définie par :* $\mathcal{Q}_1 \bowtie \mathcal{Q}_2 = \{q_1 \bowtie q_2 \mid q_1 \in \mathcal{Q}_1 \text{ et } q_2 \in \mathcal{Q}_2\}$.
5. *Si $Sch_1 = Sch_2$, alors soit θ un opérateur ensembliste (i.e. union, intersection ou différence). On donne alors la définition suivante :* $\mathcal{Q}_1 \ \theta \ \mathcal{Q}_2 = \{q_1 \ \theta \ q_2 \mid q_1 \in \mathcal{Q}_1 \text{ et } q_2 \in \mathcal{Q}_2\}$.

Exemple 3.3 *Nous illustrons la différence entre la sélection et la réduction en utilisant l'exemple 3.1. Considérons la condition de sélection $S = (Caddr = Paris)$ et l'ensemble de requêtes $\mathcal{Q} = \{q_1, q_2\}$ où $q_1 = \sigma_{(Cjob=Avocat \wedge Caddr=Paris)}(Cust)$ et $q_2 = \sigma_{Cjob=Enseignant}(Cust)$.*
Puisque la sélection sur $\mathcal{Q}$ se fait sur chaque requête, on a : $\sigma_S(\mathcal{Q}) = \{\sigma_S(q_1), \sigma_S(q_2)\}$, *et en remplaçant S, q_1 et q_2 par leurs définitions respectives, on obtient :*

$$\sigma_S(\mathcal{Q}) = \{\sigma_{(Cjob=Avocat \wedge Caddr=Paris)}(Cust), \sigma_{(Cjob=Enseignant \wedge Caddr=Paris)}(Cust)\}.$$

Par contre, la réduction de $\mathcal{Q}$ se traduit en gardant q_1 et en enlevant q_2 de $\mathcal{Q}$, puisque $\sigma_S(q_1) \equiv q_1$ et $\sigma_S(q_2) \not\equiv q_2$. Ainsi on a : $\tau_S(\mathcal{Q}) = \{q_1\}$, *i.e.*

$$\tau_S(\mathcal{Q}) = \{\sigma_{(Cjob=Avocat \wedge Caddr=Paris)}(Cust)\}.$$

Nous définissons maintenant deux méthodes de comparaison d'ensembles de requêtes. La première utilise l'inclusion de requêtes standard [91], alors que la seconde utilise l'équivalence.

Définition 3.6 - Comparaison d'ensembles de requêtes. *Soit $\mathcal{Q}_1$ et $\mathcal{Q}_2$ deux ensembles de requêtes de même schéma.*

- *$\mathcal{Q}_1$ est* inclus *dans $\mathcal{Q}_2$, noté $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$, si pour toute requête q_1 dans $\mathcal{Q}_1$, il existe une requête q_2 dans $\mathcal{Q}_2$ telle que $q_1 \sqsubseteq q_2$.*
- *$\mathcal{Q}_1$ is* fortement inclus *dans $\mathcal{Q}_2$, noté $\mathcal{Q}_1 \widetilde{\subseteq} \mathcal{Q}_2$, si pour toute requête q_1 dans $\mathcal{Q}_1$, il existe une requête q_2 dans $\mathcal{Q}_2$ telle que $q_1 \equiv q_2$.*
  De plus, $\mathcal{Q}_1$ est fortement équivalent *à $\mathcal{Q}_2$, noté $\mathcal{Q}_1 \sim \mathcal{Q}_2$, si $\mathcal{Q}_1 \widetilde{\subseteq} \mathcal{Q}_2$ et $\mathcal{Q}_2 \widetilde{\subseteq} \mathcal{Q}_1$.*

Ainsi, on peut dire que si $\mathcal{Q}_1$ et $\mathcal{Q}_2$ sont deux ensembles de requêtes de même schéma tels que $\mathcal{Q}_1 \widetilde{\subseteq} \mathcal{Q}_2$, alors on a $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$, tandis que le contraire n'est pas vrai. De plus, on peut démontrer les propositions suivantes :

Proposition 3.3 *Soit $\mathcal{C}_1 = \langle R_1, B_1, \Sigma_1 \rangle$ et $\mathcal{C}_2 = \langle R_2, B_2, \Sigma_2 \rangle$ deux contextes tels que $K = sch(R_1) = sch(R_2)$ et $sch(R_1 \bowtie B_1) = sch(R_2 \bowtie B_2)$. Alors pour toute instance I, on a :*

1. *Si $\mathcal{Q}(\mathcal{C}_1) \sqsubseteq \mathcal{Q}(\mathcal{C}_2)$ et $R_2 \sqsubseteq R_1$, alors pour toute requête $q_1 \in \mathcal{Q}(\mathcal{C}_1)$, il existe une requête $q_2 \in \mathcal{Q}(\mathcal{C}_2)$ telle que $Sup(q_1 \mid \mathcal{C}, I) \leq Sup(q_2 \mid \mathcal{C}, I)$.*
2. *Si $\mathcal{Q}(\mathcal{C}_1) \widetilde{\subseteq} \mathcal{Q}(\mathcal{C}_2)$, alors pour toute requête $q_1 \in \mathcal{Q}(\mathcal{C}_1)$, il existe une requête $q_2 \in \mathcal{Q}(\mathcal{C}_2)$ telle que $Sup(q_1 \mid \mathcal{C}_1, I)/Sup(q_2 \mid \mathcal{C}_2, I) = |R_2(I)|/|R_1(I)|$.*

Preuve :
Par définition, si $\mathcal{Q}(\mathcal{C}_1) \sqsubseteq \mathcal{Q}(\mathcal{C}_2)$, si pour toute requête q_1 dans $\mathcal{Q}(\mathcal{C}_1)$, il existe q_2 dans $\mathcal{Q}(\mathcal{C}_2)$ telle que $q_1 \sqsubseteq q_2$.
$\Longrightarrow \pi_K(q_1) \sqsubseteq \pi_K(q_2)$.
De plus, on a : $R_2 \sqsubseteq R_1$.
Ainsi, d'après la proposition 3.1, on a :
$Sup(q_1 \mid \mathcal{C}, I) \leq Sup(q_2 \mid \mathcal{C}, I)$, ce qui démontre l'item 1 de la proposition.
Par définition, si $\mathcal{Q}(\mathcal{C}_1) \widetilde{\subseteq} \mathcal{Q}(\mathcal{C}_2)$, alors pour toute requête $q_1 \in \mathcal{Q}(\mathcal{C}_1)$, il existe une requête $q_2 \in \mathcal{Q}(\mathcal{C}_2)$ telle que $q_1 \equiv q_2$.
Si $q_1 \equiv q_2$, alors pour toute instance I, $q_1(I) = q_2(I)$, ce qui implique que $\pi_K(q_1)(I) = \pi_K(q_2)(I)$.
$Sup(q_1 \mid \mathcal{C}_1, I) = \frac{|(\pi_K(q_1))(I)|}{|R_1(I)|}$.
$Sup(q_2 \mid \mathcal{C}_2, I) = \frac{|(\pi_K(q_2))(I)|}{|R_2(I)|}$.
Par conséquent $\frac{Sup(q_1 \mid \mathcal{C}_1, I)}{Sup(q_2 \mid \mathcal{C}_2, I)} = \frac{|(\pi_K(q_1))(I)|}{|R_1(I)|} * \frac{|R_2(I)|}{|(\pi_K(q_2))(I)|}$.
Puisque $|(\pi_K(q_1))(I)| = |(\pi_K(q_2))(I)|$, alors
$Sup(q_1 \mid \mathcal{C}_1, I)/Sup(q_2 \mid \mathcal{C}_2, I) = |R_2(I)|/|R_1(I)|$. $\diamond$
Notons que lorsque $R_1 = R_2 = R$, le cas 2 revient à $Sup(q_1 \mid \mathcal{C}_1, I) = Sup(q_2 \mid \mathcal{C}_2, I)$. Cette proposition sera utilisée à la section 3.4 pour savoir si une requête est fréquente sans accéder à la base de données.

3.3.2 Projection sur la base d'un contexte

Regardons maintenant comment l'ensemble des requêtes fréquentes d'un contexte change lorsque l'on supprime par projection des attributs de sa base.

Dans la définition et dans la proposition suivante, étant donné un domaine Σ d'un contexte, on note $att(\Sigma)$ l'ensemble de tous les attributs qui apparaissent dans Σ.

Définition 3.7 - Projection d'un Contexte. *Soit $\mathcal{C} = \langle R, B, \Sigma \rangle$ un contexte, et soit X un ensemble d'attributs. Si $att(\Sigma) \subseteq X \subseteq sch(B)$, alors la* b-projection de $\mathcal{C}$ sur X, *notée $\pi^b_X(\mathcal{C})$, est définie par :*

$$\pi^b_X(\mathcal{C}) = \langle R, \pi_X(B), \Sigma \rangle.$$

Proposition 3.4 *Soit $\mathcal{C} = \langle R, B, \Sigma \rangle$ un contexte, et soit X un ensemble d'attributs. Si $sch(R) \subseteq X$ et $att(\Sigma) \subseteq X \subseteq sch(B)$, alors pour tout seuil de support α et toute instance I, on a :*

$$freq_\alpha(\pi^b_X(\mathcal{C}), I) \sim \pi_X(freq_\alpha(\mathcal{C}, I)).$$

Preuve :
Soit $\mathcal{C}^*$ le contexte d'extraction défini par $\mathcal{C}^* = \langle R, \pi_X(B), \Sigma \rangle$. Soit $q = R \bowtie \sigma_S(B) = \sigma_S(R \bowtie B)$ une requête de $\mathcal{Q}(\mathcal{C})$. Puisque $att(S) \subseteq att(\Sigma) \subseteq X$ et $Sch(R) \subseteq X$, on a :

$$\begin{aligned}
\pi_X(q) &\equiv \pi_X(\sigma_S(R \bowtie B)) \\
&\equiv \sigma_S(\pi_X(R \bowtie B)) \\
&\equiv \sigma_S(\pi_{X \cap sch(R)}(R) \bowtie \pi_{X \cap sch(B)}(B)) \\
&\equiv \sigma_S(\pi_{sch(R)}(R) \bowtie \pi_X(B)) \\
&\equiv \sigma_S(R \bowtie \pi_X(B))
\end{aligned}$$

Puisque $\sigma_S(R \bowtie \pi_X(B)) \in \mathcal{Q}(\mathcal{C}^*)$, on peut dire que toute requête de $\pi_X(\mathcal{Q}(\mathcal{C}))$ est équivalente à une requête dans $\mathcal{Q}(\mathcal{C}^*)$ avec le même support.
Inversement, soit $q^* = \sigma_S(R \bowtie \pi_X(B)$ une requête de $\mathcal{Q}(\mathcal{C}^*)$. On a : $q^* \equiv \pi_X(\sigma_S(R \bowtie B))$ où $\sigma_S(R \bowtie B) \in \mathcal{Q}(\mathcal{C})$. Ainsi, toute requête de $\mathcal{Q}(\mathcal{C}^*)$ est aussi équivalente à une requête de

$\pi_X(\mathcal{Q}(\mathcal{C}))$. Par conséquent, $\pi_X(\mathcal{Q}(\mathcal{C}))$ et $\mathcal{Q}(\mathcal{C}^*)$ sont des ensembles de requêtes équivalents et $freq_\alpha(\pi^b_X(\mathcal{C}), I) \sim \pi_X(freq_\alpha(\mathcal{C}, I))$. $\diamond$
Intuitivement, étant donné un contexte $\mathcal{C} = \langle R, B, \Sigma\rangle$, la proposition 3.4 ci-dessus implique que tous attributs qui n'apparaissent pas dans R ou dans Σ peuvent ne pas être pris en compte lors de l'extraction.

Exemple 3.4 *Considérons de nouveau la base de données DBSales de l'exemple 3.1, et soit* $\mathcal{C}_1 = \langle AllCust, Cust \bowtie Sales \bowtie Prod, \Sigma_1\rangle$ *où* $AllCust = \pi_{Cid}(Cust)$ *et* Σ_1 *est l'ensemble de toutes les conditions de sélection de la forme* $(Cjob = job)$ *ou* $(Ptype = type)$ *où job et type sont des constantes appartenant à* $dom(Cjob)$ *et* $dom(Ptype)$ *respectivement. Dans les exemples suivants, on note un tel domaine par :* $\Sigma_1 = \{Cjob = *, Ptype = *\}$.
Alors, considérons le nouveau contexte $\mathcal{C}_2 = \pi^b_{Cid,Cjob,Ptype}(\mathcal{C}_1)$. *On a :*

$$\mathcal{C}_2 = \langle AllCust, \pi_{Cid,Cjob,Ptype}(Cust \bowtie Sales \bowtie Prod), \Sigma_1\rangle.$$

De plus, en utilisant la proposition 3.4, pour tout seuil de support α *et toute instance* I *de DBSales, on a :*

$$freq_\alpha(\mathcal{C}_2, I) \ \sim \ \pi_{Cid,Cjob,Ptype}(freq_\alpha(\mathcal{C}_1, I)).$$

Ainsi, en appliquant la proposition 3.3, l'ensemble $freq_\alpha(\mathcal{C}_2, I)$ *peut être obtenu par projection des requêtes dans* $freq_\alpha(\mathcal{C}_1, I)$.

3.3.3 Sélection sur la base d'un contexte

Dans cette sous-section, nous étudions comment l'ensemble des requêtes fréquentes d'un contexte change lorsque l'on applique une sélection sur sa base.

Dans la définition et dans la proposition suivante, étant donné une condition de sélection S, on note $att(S)$ l'ensemble de tous les attributs qui apparaissent dans S.

Définition 3.8 - Sélection sur un Contexte. *Soit* $\mathcal{C} = \langle R, B, \Sigma\rangle$ *un contexte, et soit* S *une condition de sélection. Si* $att(S) \subseteq sch(B)$, *alors la* b-sélection de $\mathcal{C}$ sur S, *notée* $\sigma^b_S(\mathcal{C})$, *est définie par :* $\sigma^b_S(\mathcal{C}) = \langle R, \sigma_S(B), \Sigma\rangle$.

Proposition 3.5 *Soit* $\mathcal{C} = \langle R, B, \Sigma\rangle$ *un contexte. Si* S *est une condition de sélection dans* Σ^*, *alors pour tout seuil de support* α *et toute instance* I, *on a :*

$$freq_\alpha(\sigma^b_S(\mathcal{C}), I) \sim \tau_S(freq_\alpha(\mathcal{C}, I)).$$

Preuve :
Soit $\mathcal{C}^*$ le contexte défini par $\mathcal{C}^* = \langle R, \sigma_S(B), \Sigma\rangle$. Soit $q = \sigma_{S_1}(R \bowtie B)$ une requête de $\tau_S(\mathcal{Q}(\mathcal{C}))$. Par définition, $\sigma_S(\sigma_{S_1}(R \bowtie B)) \equiv \sigma_{S_1}(R \bowtie B)$. Ainsi q appartient à $\tau_S(\mathcal{Q}(\mathcal{C}))$ si $S_1 = S \wedge S_2$ où $S_2 \in \Sigma^*$. On a alors :

$$\begin{array}{rcl} q & \equiv & \sigma_{S \wedge S_2}(R \bowtie B) \\ & \equiv & \sigma_{S_2}(R \bowtie \sigma_S(B)) \end{array}$$

où $\sigma_{S_2}(R \bowtie \sigma_S(B)) \in \mathcal{Q}(\mathcal{C}^*)$. Ainsi, toute requête de $\tau_S(\mathcal{Q}(\mathcal{C}))$ est équivalente à une requête dans $\mathcal{Q}(\mathcal{C}^*)$ avec le même support, ce qui montre que $\tau_S(freq_\alpha(\mathcal{C}, I)) \widetilde{\subseteq} freq_\alpha(\sigma^b_S(\mathcal{C}), I)$.
Inversement, soit $q^* = \sigma_{S_2}(R \bowtie \sigma_S(B))$ une requête dans $\mathcal{Q}(\mathcal{C}^*)$. On a : $q^* \equiv \sigma_{S_2 \wedge S}(R \bowtie B)$ où $\sigma_{S_2 \wedge S}(R \bowtie B) \in \mathcal{Q}(\mathcal{C})$. De plus, $\sigma_S(\sigma_{S_2 \wedge S}(R \bowtie B)) \equiv \sigma_{S_2 \wedge S}(R \bowtie B)$. Ainsi, $\sigma_{S_2 \wedge S}(R \bowtie B)$ est une requête dans $\tau_S(\mathcal{Q}(\mathcal{C}))$, ce qui veut dire que toute requête dans $\mathcal{Q}(\mathcal{C}^*)$ est équivalente à une requête dans $\tau_S(\mathcal{Q}(\mathcal{C})$ avec le même support. Ainsi, $freq_\alpha(\sigma^b_S(\mathcal{C}), I) \widetilde{\subseteq} \tau_S(freq_\alpha(\mathcal{C}, I))$ et

par conséquent $freq_\alpha(\sigma^b_S(\mathcal{C}), I) \sim \tau_S(freq_\alpha(\mathcal{C}, I))$. $\diamond$

Par la suite, nous dirons qu'une conjonction de conditions de sélection atomiques $S = (A_1 = a_1) \wedge \ldots \wedge (A_n = a_n)$ est *indépendante* d'un domaine Σ si pour tout $i = 1 \ldots n$, la condition de sélection atomique $(A_i = a_i)$ ne se trouve pas dans Σ.

Proposition 3.6 *Soit $\mathcal{C} = \langle R, B, \Sigma\rangle$ un contexte. Si S est une condition de sélection indépendente de Σ telle que $att(S) \subseteq sch(B)$, alors pour tout seuil de support α et toute instance I, on a :*

$$freq_\alpha(\sigma^b_S(\mathcal{C}), I) \;\widetilde{\subseteq}\; \sigma_S(freq_\alpha(\mathcal{C}, I)).$$

Preuve :
Soit $q^* = \sigma_{S_2}(R \bowtie \sigma_S(B))$ une requête dans $freq_\alpha(\sigma^b_S(\mathcal{C}), I)$. Puisque l'opération σ est commutative, on a $q^* \equiv \sigma_{S_2}(\sigma_S(R \bowtie B)) \equiv \sigma_S(\sigma_{S_2}(R \bowtie B))$ où $q = \sigma_{S_2}(R \bowtie B)$ est une requête de $\mathcal{Q}(\mathcal{C})$. Ainsi, toute requête dans $freq_\alpha(\sigma^b_S(\mathcal{C}), I)$ est équivalente à une requête dans $\sigma_S(\mathcal{Q}(\mathcal{C}))$. De plus, $q^* = \sigma_S(q)$. Ainsi, selon le corollaire 3.1, on a :

$$Sup(q \mid \mathcal{C}, I) \geq Sup(q^* \mid \mathcal{C}^*, I)$$

où $\mathcal{C}^* = \langle R, \sigma_S(B), \Sigma\rangle$.De plus, $q^* \in freq_\alpha(\sigma^b_S(\mathcal{C}), I)$. Donc :

$$Sup(q \;/\; \mathcal{C}, I) \geq Sup(q^* \;/\; C^*, I) \geq \alpha.$$

Ainsi q est dans $freq_\alpha(\mathcal{C}, I)$, ce qui complète la preuve. $\diamond$

Le corollaire suivant est une conséquence immédiate des propositions 3.5 et 3.6.

Corollaire 3.2 *Soit $\mathcal{C} = \langle R, B, \Sigma\rangle$ un contexte, et $S = S_1 \wedge S_2$ une condition de sélection telle que $att(S) \subseteq sch(B)$, $S_1 \in \Sigma^*$, et S_2 est indépendente de Σ. Pour tout seuil de support α et toute instance I, on a :*

$$freq_\alpha(\sigma^b_S(\mathcal{C}), I) \;\widetilde{\subseteq}\; \tau_{S_1}(\sigma_{S_2}(freq_\alpha(\mathcal{C}, I))).$$

En utilisant la commutativité de l'opération de sélection, on peut voir que $\tau_{S_1}(\sigma_{S_2}(freq_\alpha(\mathcal{C}, I))) \sim \sigma_{S_2}(\tau_{S_1}(freq_\alpha(\mathcal{C}, I)))$. Ainsi le corollaire 3.2 implique que : $freq_\alpha(\sigma^b_S(\mathcal{C}), I) \;\widetilde{\subseteq}\; \sigma_{S_2}(\tau_{S_1}(freq_\alpha(\mathcal{C}, I)))$.

Exemple 3.5 *Considérons de nouveau la base de données DBSales de l'exemple 3.1 et la requête AllCust définie par : $AllCust = \pi_{Cid}(Cust)$.*

Supposons que l'on s'intéresse d'abord aux relations entre les professions des clients et les types de produits qu'ils achètent. On considère alors le contexte $\mathcal{C}_1 = \langle AllCust, Cust \bowtie Sales \bowtie Prod, \Sigma\rangle$ où $\Sigma = \{Cjob = *, Ptype = *\}$.

Supposons maintenant que l'on s'intéresse seulement aux règles concernant les avocats. Si nous considérons le contexte $\mathcal{C}_2 = \sigma^b_{Cjob=Avocat}(\mathcal{C}_1)$, alors, selon la proposition 3.5, pour tout seuil de support α et toute instance I de $DBSales$, on a :

$$freq_\alpha(\mathcal{C}_2, I) \sim \tau_{Cjob=Avocat}(freq_\alpha(\mathcal{C}_1, I))$$

Ainsi, selon la proposition 3.3, cela revient à dire que les requêtes fréquentes de $\mathcal{C}_1$ qui concernent les avocats sont exactement les requêtes fréquentes de $\mathcal{C}_2$.

Finalement, supposons que l'on s'intéresse aux règles concernant seulement les clients vivant à Paris. Le contexte correspondant $\mathcal{C}_3$ peut être défini par $\mathcal{C}_3 = \sigma^b_{Caddr=Paris}(\mathcal{C}_1)$, et selon la proposition 3.6, pour toute instance I de $DBSales$, on a : $freq_\alpha(\mathcal{C}_3, I) \subseteq \sigma_{Caddr=Paris}(freq_\alpha(\mathcal{C}_1, I))$. Dans ce cas, en appliquant la proposition 3.3, si une requête q_1 de $\mathcal{C}_1$ n'est pas fréquente, la requête correspondante $q_3 = \sigma_{Caddr=Paris}(Q_1)$ de $\mathcal{C}_3$ n'est pas fréquente non plus, et cette information est connue sans accéder à la base de données.

3.3.4 Jointure de contextes

Dans cette sous-section, nous considérons la jointure de contextes.

Définition 3.9 - Jointure de contextes. *Soit* $\mathcal{C}_1 = \langle R, B_1, \Sigma_1 \rangle$ *et* $\mathcal{C}_2 = \langle R, B_2, \Sigma_2 \rangle$ *deux contextes de même reference* R. *La* jointure *de* $\mathcal{C}_1$ *et* $\mathcal{C}_2$, *notée* $\mathcal{C}_1 \bowtie_b \mathcal{C}_2$, *est définie par :* $\mathcal{C}_1 \bowtie_b \mathcal{C}_2 = \langle R, B_1 \bowtie B_2, \Sigma_1 \cup \Sigma_2 \rangle$.

Proposition 3.7 *Soit* $\mathcal{C}_1 = \langle R, B_1, \Sigma_1 \rangle$ *et* $\mathcal{C}_2 = \langle R, B_2, \Sigma_2 \rangle$ *deux contextes de même reference* R. *Pour tout seuil de support* α *et toute instance* I, *on a :*

$$freq_\alpha(\mathcal{C}_1 \bowtie_b \mathcal{C}_2, I) \mathrel{\widetilde{\subseteq}} freq_\alpha(\mathcal{C}_1, I) \bowtie freq_\alpha(\mathcal{C}_2, I).$$

Preuve :
Soit $q_{12} = \sigma_{S_{12}}(R \bowtie B_1 \bowtie B_2)$ une requête de $freq_\alpha(\mathcal{C}_1 \bowtie_b \mathcal{C}_2, I)$. Si S_1 est la conjonction de conditions de sélection dans S_{12} qui implique seulement les attributs dans $sch(R) \cup sch(B_1)$, et S_2 est la conjonction de conditions de sélection dans S_{12} qui implique seulement les attributs dans $sch(R) \cup sch(B_2)$, on a : $q_{12} \equiv \sigma_{S_1}(R \bowtie B_1) \bowtie \sigma_{S_2}(R \bowtie B_2)$ où $q_1 = \sigma_{S_1}(R \bowtie B_1)$ est une requête de $\mathcal{Q}(\mathcal{C}_1)$ et $q_2 = \sigma_{S_2}(R \bowtie B_2)$ est une requête de $\mathcal{Q}(\mathcal{C}_2)$. De plus, selon le corollaire 3.1, on a : $Sup(q_i \mid \mathcal{C}_i, I) \geq Sup(q_{12} \mid \mathcal{C}_i, I) \geq \alpha$ pour $i = 1, 2$. Ainsi, q_1 et q_2 sont des requêtes dans $freq_\alpha(\mathcal{C}_1, I)$ et $freq_\alpha(\mathcal{C}_2, I)$, ce qui complète la preuve. $\diamond$

Exemple 3.6 *Considérons de nouveau la base de données* $DBSales$ *de l'exemple 3.1 et la requête* $AllCust = \pi_{Cid}(Cust)$.

Supposons dans un premier temps que l'on s'interesse aux relations entre les professions des clients et les types de produits qu'ils achètent. Alors, comme nous l'avons déjà mentionné, nous définissons le contexte $\mathcal{C}_1 = \langle AllCust, Cust \bowtie Sales \bowtie Prod, \Sigma_1 \rangle$ *où* $\Sigma_1 = \{Cjob = *, Ptype = *\}$.

Ensuite supposons que l'on s'intéresse aux relations entre les noms des magasins où les clients effectuent leurs achats et les types de produits qu'ils achètent. Le contexte correspondant $\mathcal{C}_2$ *est défini par* $\mathcal{C}_2 = \langle AllCust, Store \bowtie Sales \bowtie Prod, \Sigma_2 \rangle$ *où* $\Sigma_2 = \{Sname = *, Ptype = *\}$.

Maintenant, si on considère les relations entre les professions des clients, les noms de magasins où ils effectuent leurs achats et les types de produits qu'ils achètent, alors le contexte correspondant $\mathcal{C}_{12}$ *peut être défini par :* $\mathcal{C}_{12} = \mathcal{C}_1 \bowtie_b \mathcal{C}_2$.

Selon la proposition 3.7, pour tout seuil de support α *et toute instance* I *de* $DBSales$, *on a :* $freq_\alpha(\mathcal{C}_{12}, I) \mathrel{\widetilde{\subseteq}} freq_\alpha(\mathcal{C}_1, I) \bowtie freq_\alpha(\mathcal{C}_2, I)$. *Ainsi, à partir de la définition 3.6, pour toute requête* q_{12} *dans* $\mathcal{Q}(\mathcal{C}_{12})$, *il existe deux requêtes* q_1 *et* q_2 *de* $\mathcal{C}_1$ *et* $\mathcal{C}_2$ *respectivement, telles que* $q_{12} \equiv q_1 \bowtie q_2$. *De plus, selon le corollaire 3.1,* $Sup(q_1 \bowtie q_2 \mid \mathcal{C}_{12}, I) \leq Sup(q_i \mid \mathcal{C}_i, I)$, *pour* $i = 1, 2$. *Par conséquent, si* q_{12} *est fréquente, alors* q_1 *et* q_2 *le sont aussi.*

3.3.5 Union, intersection et différence de contextes

Dans cette sous-section, nous construisons de nouveaux contextes à partir d'anciens en appliquant les opérations ensemblistes sur les bases des anciens contextes.

Définition 3.10 Union, Intersection et différence de contextes. *Soit* $\mathcal{C}_1 = \langle R, B_1, \Sigma \rangle$ *et* $\mathcal{C}_2 = \langle R, B_2, \Sigma \rangle$ *deux contextes de même référence* R *et de même domaine* Σ, *tels* $sch(B_1) = sch(B_2)$.

1. *L'union de* $\mathcal{C}_1$ *et* $\mathcal{C}_2$, *notée* $\mathcal{C}_1 \cup_b \mathcal{C}_2$ *est définie par* $\mathcal{C}_1 \cup_b \mathcal{C}_2 = \langle R, B_1 \cup B_2, \Sigma \rangle$.
2. *L'intersection de* $\mathcal{C}_1$ *et* $\mathcal{C}_2$, *notée* $\mathcal{C}_1 \cap_b \mathcal{C}_2$ *est définie par* $\mathcal{C}_1 \cap_b \mathcal{C}_2 = \langle R, B_1 \cap B_2, \Sigma \rangle$.

3. La différence de $\mathcal{C}_2$ et $\mathcal{C}_1$, notée $\mathcal{C}_2 \setminus_b \mathcal{C}_1$, est définie par $\mathcal{C}_2 \setminus_b \mathcal{C}_1 = \langle R, B_2 - B_1, \Sigma \rangle$.

A partir de cette définition et du corollaire 3.1, on peut énoncer la proposition suivante :

Proposition 3.8 *Soit $\mathcal{C}_1 = \langle R, B_1, \Sigma \rangle$ et $\mathcal{C}_2 = \langle R, B_2, \Sigma \rangle$ deux contextes de même référence R et de même domaine Σ, tels que $sch(B_1) = sch(B_2)$. Pour tout seuil de support α et toute instance I, on a :*

1. *$freq_\alpha(\mathcal{C}_i, I) \sqsubseteq freq_\alpha(\mathcal{C}_1 \cup_b \mathcal{C}_2, I)$, pour $i = 1, 2$.*
2. *$freq_\alpha(\mathcal{C}_2 \cap_b \mathcal{C}_1, I) \sqsubseteq freq_\alpha(\mathcal{C}_i, I)$, pour $i = 1, 2$.*
3. *$freq_\alpha(\mathcal{C}_2 \setminus_b \mathcal{C}_1, I) \sqsubseteq freq_\alpha(\mathcal{C}_2, I)$.*

Preuve :
Soit q une requête de $freq_\alpha(\mathcal{C}_1 \cup_b \mathcal{C}_2, I)$. Alors q est de la forme $R \bowtie \sigma_S(B_1 \cup B_2)$, où S est une condition de sélection.
$R \bowtie \sigma_S(B_1 \cup B_2) = R \bowtie (\sigma_S(B_1) \cup \sigma_S(B_2)) = (R \bowtie \sigma_S(B_1)) \cup (R \bowtie \sigma_S(B_2)) \sqsupseteq (R \bowtie \sigma_S(B_i))$ pour $i = 1, 2$.
Ainsi, pour toute requête $q_i = R \bowtie \sigma_S(B_i) \in freq_\alpha(\mathcal{C}_i, I)$, avec $i = 1, 2$, il existe une requête $q \in freq_\alpha(\mathcal{C}_1 \cup_b \mathcal{C}_2, I)$ telle que $q_i \sqsubseteq q$.
Par conséquent, $freq_\alpha(\mathcal{C}_i, I) \sqsubseteq freq_\alpha(\mathcal{C}_1 \cup_b \mathcal{C}_2, I)$, pour $i = 1, 2$, ce qui démontre l'item 1.
Soit q une requête de $freq_\alpha(\mathcal{C}_2 \cap_b \mathcal{C}_1, I)$. Alors q est de la forme $R \bowtie \sigma_S(B_1 \cap B_2)$, où S est une condition de sélection.
$R \bowtie \sigma_S(B_1 \cap B_2) = R \bowtie (\sigma_S(B_1) \cap \sigma_S(B_2)) = (R \bowtie \sigma_S(B_1)) \cap (R \bowtie \sigma_S(B_2)) \sqsubseteq (R \bowtie \sigma_S(B_i))$ pour $i = 1, 2$.
On démontre ainsi de la même façon que $freq_\alpha(\mathcal{C}_2 \cap_b \mathcal{C}_1, I) \sqsubseteq freq_\alpha(\mathcal{C}_i, I)$, pour $i = 1, 2$.
Un raisonnement similaire peut être fait pour démontrer l'item 3.

Exemple 3.7 *Considérons de nouveau la base de données $DBSales$ de l'exemple 3.1 et la requête $AllCust = \pi_{Cid}(Cust)$.*

Supposons d'abord que l'on s'intéresse aux relations entre les clients qui font leurs achats dans les magasins "Carrefour" et les types de produits qu'ils achètent. Le contexte correspondant $\mathcal{C}_1$ est défini par $\mathcal{C}_1 = \langle AllCust, CCarrefour \bowtie Sales \bowtie Prod, \Sigma \rangle$ où $CCarrefour = \pi_{Cid,Cjob}(Cust \bowtie Sales \bowtie \sigma_{Sname=Carrefour}(Store))$ et $\Sigma = \{Cjob = *, Ptype = *\}$.

De même, si l'on s'intéresse aux relations entre les professions des clients qui font leurs achats dans les magasins "Fnac" et les types de produits qu'ils achètent, le contexte correspondant $\mathcal{C}_2$ est défini par $\mathcal{C}_2 = \langle AllCust, CFnac \bowtie Sales \bowtie Prod, \Sigma \rangle$ où
$CFnac = \pi_{Cid,Cjob}(\sigma_{Sname=Fnac}(Store) \bowtie Sales \bowtie Cust)$ et $\Sigma = \{Cjob = *, Ptype = *\}$. Alors les contextes $\mathcal{C}_1 \cup_b \mathcal{C}_2$, $\mathcal{C}_1 \cap_b \mathcal{C}_2$ et $\mathcal{C}_1 \setminus_b \mathcal{C}_2$ référent respectivement aux relations entre les professions des clients qui font leurs achats :

- *dans les magasins "Carrefour" ou "Fnac",*
- *dans les magasins "Carrefour" et "Fnac",*
- *dans les magasins "Carrefour" mais pas à la "Fnac",*

et les types de produits qu'ils achètent. De plus, selon la proposition 3.8, on a :

1. *$freq_\alpha(\mathcal{C}_i, I) \sqsubseteq freq_\alpha(\mathcal{C}_1 \cup_b \mathcal{C}_2, I)$, pour $i = 1, 2$.*
2. *$freq_\alpha(\mathcal{C}_2 \cap_b \mathcal{C}_1, I) \sqsubseteq freq_\alpha(\mathcal{C}_i, I)$, pour $i = 1, 2$.*
3. *$freq_\alpha(\mathcal{C}_2 \setminus_b \mathcal{C}_1, I) \sqsubseteq freq_\alpha(\mathcal{C}_2, I)$.*

Ainsi, la proposition 3.3 implique que les calculs de $freq_\alpha(\mathcal{C}_1 \cup_b \mathcal{C}_2, I)$, $freq_\alpha(\mathcal{C}_1 \cap_b \mathcal{C}_2, I)$ et $freq_\alpha(\mathcal{C}_1 \setminus_b \mathcal{C}_2, I)$ peuvent être optimisés en utilisant les résultats de $freq_\alpha(\mathcal{C}_1, I)$ et $freq_\alpha(\mathcal{C}_2, I)$

3.3.6 Restriction d'un contexte

Dans cette section, nous considérons une opération qui change la référence d'un contexte. Plus précisément, étant donné un contexte $\mathcal{C} = \langle R, B, \Sigma \rangle$, cette opération restreint la référence aux objets qui satisfont un critère de sélection spécifié dans une requête de $\mathcal{C}$. La nouvelle référence est alors plus spécifique que R.

Définition 3.11 *Soit $\mathcal{C} = \langle R, B, \Sigma \rangle$ un contexte avec $K = sch(R)$, et soit S une condition de sélection dans Σ^*. La restriction de $\mathcal{C}$ sur S est définie par :*

$$\rho_S(\mathcal{C}) = \langle \pi_K(R \bowtie \sigma_S(B)), \sigma_S(B), \Sigma \rangle$$

Proposition 3.9 *Soit $\mathcal{C} = \langle R, B, \Sigma \rangle$ un contexte, et soit S une condition de sélection dans Σ^*. Alors, pour tout seuil de support α et toute instance I, on a :*

$$\tau_S(freq_\alpha(\mathcal{C}), I) \mathrel{\widetilde{\subseteq}} freq_\alpha(\rho_S(\mathcal{C}), I).$$

Preuve :
Il s'agit de montrer que pour toute requête $q_1 \in \tau_S(freq_\alpha(\mathcal{C}), I)$, il existe une requête $q_2 \in freq_\alpha(\rho_S(\mathcal{C}), I)$ telle que $q_1 \equiv q_2$.
Soit $q = R \bowtie \sigma_{S_1}(B)$ une requête de $\tau_S(freq_\alpha(\mathcal{C}), I)$. Par définition :

$$\sigma_S(R \bowtie \sigma_{S_1}(B)) \equiv R \bowtie \sigma_{S_1}(B).$$

Une telle équivalence implique que $S_1 = S \wedge S_2$, où $S_2 \in \Sigma^*$.
Il faut alors montrer qu'il existe une requête q_2 telle que $q_2 \equiv q$ où :

$$q_2 = \pi_K(R \bowtie \sigma_S(B)) \bowtie \sigma_{S_2}(\sigma_S(B)) \in freq_\alpha(\rho_S(\mathcal{C}), I).$$

Pour cela, nous allons montrer que pour toute instance I :

1. $(\pi_K(R \bowtie \sigma_S(B)) \bowtie \sigma_{S \wedge S_2}(B))[I] \subseteq q[I]$ et
2. $q[I] \subseteq (\pi_K(R \bowtie \sigma_S(B)) \bowtie \sigma_{S \wedge S_2}(B))[I]$.

1. $(\pi_K(R \bowtie \sigma_S(B)) \bowtie \sigma_{S \wedge S_2}(B))[I] \subseteq q[I]$.
   Soit t un tuple tel que $t \in (\pi_K(R \bowtie \sigma_S(B)) \bowtie \sigma_{S \wedge S_2}(B))[I]$. Rappelons que $K = sch(R)$ et $sch(R) \subseteq sch(B)$. La jointure entre R et B se fait donc sur K. On a donc : $t.K \in R[I]$, où $t.K$ représente les valeurs de t sur les attributs de K. De même $t \in \sigma_{S \wedge S_2}(B)[I]$.
   Par conséquent, $t \in (R \bowtie \sigma_{S \wedge S_2}(B))$, i.e. $t \in q[I]$.

2. $q[I] \subseteq (\pi_K(R \bowtie \sigma_S(B)) \bowtie \sigma_{S \wedge S_2}(B))[I]$.
   Si $t \in q[I]$, alors $t.K \in R[I]$ et $t \in \sigma_{S \wedge S_2}(B)[I]$. Donc $t \in \sigma_S(B)[I]$ (qui contient $\sigma_{S \wedge S_2}(B)[I]$) et $t.K \in (\pi_K(R \bowtie \sigma_S(B)))[I]$.
   Par conséquent, $t \in (\pi_K(R \bowtie \sigma_S(B)) \bowtie \sigma_{S \wedge S_2}(B))[I]$. $\diamond$

Exemple 3.8 *Considérons de nouveau la base de données DBSales de l'exemple 3.1 et supposons maintenant que l'on s'intéresse aux relations entre les professions des clients, leurs villes de résidence et les types de produits qu'ils achètent. On définit alors le contexte $\mathcal{C}_1 = \langle AllCust, Cust \bowtie Sales \bowtie Prod, \Sigma \rangle$ où $\Sigma = \{Cjob = *, Caddr = *, Ptype = *\}$.*

Supposons que l'on se focalise sur les clients vivant à Paris. Le contexte correspondant $\mathcal{C}_2$ peut être défini par $\mathcal{C}_2 = \rho_{Caddr=Paris}(\mathcal{C}_1)$. Notons que la référence de $\mathcal{C}_2$ est définie par la requête $\pi_{Cid}(AllCust \bowtie \sigma_{Caddr=Paris}(Cust \bowtie Sales \bowtie Prod))$, ce qui signifie que le support des requêtes

est calculé par rapport au nombre de clients vivant à Paris (et pas par rapport au nombre total de clients). De plus, selon la proposition 3.9, pour tout seuil de support α et toute instance I de DBSales, on a : $\tau_{Caddr=Paris}(freq_\alpha(\mathcal{C}_1, I)) \widetilde{\subseteq} freq_\alpha(\mathcal{C}_2, I)$.

Par conséquent, si q_1 est une requête fréquente dans $\tau_{Caddr=Paris}(freq_\alpha(\mathcal{C}_1, I))$, alors il existe une requête dans $freq_\alpha(\mathcal{C}_2, I)$ qui est équivalente à q_1, et on aboutit à cette conclusion sans aucun accès à la base de données.

3.4 Extraction itérative de requêtes fréquentes

Nous avons vu jusque-là des opérations pour la combinaison de contextes dans le but de définir de nouveaux contextes. Nous avons aussi donné les propriétés qui montrent comment un ensemble de requêtes fréquentes d'un contexte change lorsque l'on applique ces opérations sur la référence ou la base. Dans cette section, nous passons en revue tous les cas où ces propriétés permettent d'optimiser le calcul de requêtes fréquentes de contextes combinés.

Bien entendu, pour utiliser les requêtes fréquentes des contextes, nous devons les stocker. Nous proposons deux méthodes de stockage des requêtes fréquentes : soit stocker toutes les requêtes, ou stocker seulement leurs frontières positives [64]. Rappelons que dans le second cas, on peut calculer l'ensemble des requêtes fréquentes à partir de la frontière positive en une seule passe sur les données [64].

3.4.1 Optimisations possibles

Soit $\mathcal{C}_{new}$ un nouveau contexte défini par combinaison d'autres contextes. Etant donné un seuil de support α et une instance I de DB, nous considérons tous les cas où le calcul de $freq_\alpha(\mathcal{C}_{new}, I)$ peut être optimisé en utilisant les ensembles de requêtes fréquentes dans I qui ont déjà été calculés. Nous distinguons trois cas :

1. $freq_\alpha(\mathcal{C}_{new}, I)$ est fortement équivalent à un ensemble de requêtes fréquentes déjà calculées.
2. $freq_\alpha(\mathcal{C}_{new}, I)$ est inclus ou fortement inclus dans un ensemble de requêtes fréquentes déjà calculées.
3. $freq_\alpha(\mathcal{C}_{new}, I)$ contient, fortement ou non, un ensemble de requêtes fréquentes déjà calculées.

Ces différents cas sont l'objet des paragrahes suivants.

Ensembles de requêtes fréquentes fortement équivalents

L'ensemble $freq_\alpha(\mathcal{C}_{new}, I)$ est fortement équivalent à un ensemble de requêtes fréquentes d'un contexte $\mathcal{C}_{old}$ lorsque $\mathcal{C}_{new}$ est défini à partir de $\mathcal{C}_{old}$ de l'une des façons suivantes :

- par projection de $\mathcal{C}_{old}$ (voir proposition 3.4),
- par sélection de $\mathcal{C}_{old}$, avec une condition de sélection impliquant seulement des conditions dans le domaine de $\mathcal{C}_{old}$ (voir proposition 3.5).

Dans les deux cas, si toutes les requêtes dans $freq_\alpha(\mathcal{C}_{old}, I)$ et leurs supports sont stockés, aucun calcul ou accès à la base de données ne sera nécessaire pour trouver $freq_\alpha(\mathcal{C}_{new}, I)$. Il faut juste appliquer l'opération correspondante aux requêtes de $freq_\alpha(\mathcal{C}_{old}, I)$ pour trouver l'ensemble $freq_\alpha(\mathcal{C}_{new}, I)$.

Cependant, si seule la frontière positive de $freq_\alpha(\mathcal{C}_{old}, I)$ est stockée, alors il est nécessaire de reconstruire $freq_\alpha(\mathcal{C}_{old}, I)$, et pour cela, il faut faire une passe sur les données.

Sous-ensemble d'ensembles de requêtes fréquentes

L'ensemble $freq_\alpha(\mathcal{C}_{new}, I)$ est inclus, fortement ou pas, dans l'ensemble de requêtes fréquentes d'un contexte $\mathcal{C}_{old}$ lorsque $\mathcal{C}_{new}$ est défini de l'une des façons suivantes :

- par sélection de $\mathcal{C}_{old}$ avec une condition de sélection impliquant des conditions qui ne se trouvent pas dans le domaine de $\mathcal{C}_{old}$ (voir proposition 3.6 et corollaire 3.2),
- par jointure ou intersection de deux contextes $\mathcal{C}_{old_1}$ and $\mathcal{C}_{old_2}$ (voir proposition 3.7 et proposition 3.8).

Dans les deux cas, on connaît seulement une limite supérieure des supports des requêtes candidates de $\mathcal{C}_{new}$. Ces supports doivent donc être calculés. Il est cependant important de souligner que le calcul déjà effectué des frontières positives des ensembles de requêtes fréquentes permet de faire un élagage supplémentaire (par rapport aux algorithmes de niveau standard) lors de la génération des requêtes candidates de niveau $k+1$ à partir des requêtes fréquentes de niveau k.

Pour illustrer cela, considérons le cas d'une b-sélection avec une condition de sélection S indépendante du domaine de $\mathcal{C}_{old}$. Si une requête candidate q_{new} appartient à $freq_\alpha(\mathcal{C}_{new}, I)$, alors selon la proposition 3.6, il existe une requête q_{old} dans $freq_\alpha(\mathcal{C}_{old}, I)$ telle que $q_{new} \equiv \sigma_S(q_{old})$. De plus, puisque q_{old} est dans $freq_\alpha(\mathcal{C}_{old}, I)$, il existe une requête q^+_{old} dans la frontière positive de $freq_\alpha(\mathcal{C}_{old}, I)$ telle que $q^+_{old} \sqsubseteq q_{old}$. Avec la monotonie de l'opération de sélection, on a $\sigma_S(q^+_{old}) \sqsubseteq \sigma_S(q_{old})$. Ainsi on obtient $\sigma_S(q^+_{old}) \sqsubseteq q_{new}$.

Par conséquent, s'il n'existe pas de requête q^+_{old} comme indiqué ci-dessus, alors q_{new} peut être supprimée de l'ensemble des requêtes candidates de $\mathcal{C}_{new}$. Nous montrons dans la partie sur l'implémentation comment tester efficacement l'existence de q^+_{old}.

De façon similaire , dans le cas de la jointure, si une requête candidate q_{new} appartient à $freq_\alpha(\mathcal{C}_{new}, I)$, alors selon la proposition 3.7, il existe deux requêtes fréquentes q_{old_1} et q_{old_2} telles que $q_{new} \equiv q_{old_1} \bowtie q_{old_2}$. Ainsi, il existe $q^+_{old_1}$ et $q^+_{old_2}$ dans les frontières positives de $freq_\alpha(\mathcal{C}_1, I)$ et $freq_\alpha(\mathcal{C}_2, I)$ telles que $q^+_{old_1} \bowtie q^+_{old_2} \sqsubseteq q_{new}$. Nous sommes par conséquent dans le même cas que ci-dessus, et nous montrons dans la partie sur l'implémentation comment tester efficacement l'existence de $q^+_{old_1}$ et de $q^+_{old_2}$.

Sur-ensemble d'un ensemble de requêtes fréquentes

L'ensemble $freq_\alpha(\mathcal{C}_{new}, I)$ contient, fortement ou non, un ensemble de requêtes fréquentes déjà calculées lorsque $\mathcal{C}_{new}$ est définie par :

- restriction de $\mathcal{C}_{old}$ (voir proposition 3.9),
- union de $\mathcal{C}_{old_1}$ et $\mathcal{C}_{old_2}$ (voir proposition 3.8).

Dans les deux cas, un sous-ensemble $\mathcal{F}$ de $freq_\alpha(\mathcal{C}_{new}, I)$ est connu sans calcul supplémentaire. En effet, selon la proposition 3.3, pour chaque requête q_{old} dans $freq_\alpha(\mathcal{C}_{old}, I)$ (ou dans $freq_\alpha(\mathcal{C}_{old_1} \cup_b \mathcal{C}_{old_2})), I)$, il existe une requête q_{new} dans $freq_\alpha(\mathcal{C}_{new}, I)$ dont le support est supérieur ou égal à celui de q_{old}. Ainsi, $\mathcal{F}$ est précisément la composition de $freq_\alpha(\mathcal{C}_{old}, I)$ (ou de $freq_\alpha(\mathcal{C}_{old_1}, I)$ et $freq_\alpha(\mathcal{C}_{old_2}, I))$ par l'opérateur correspondant.

De plus, dans le cas de la restriction, les supports des requêtes dans $\mathcal{F}$ peuvent être facilement obtenus, puisque seul le dénominateur du support des requêtes correspondantes dans $freq_\alpha(\mathcal{C}_{old}, I)$ change. Cependant, dans le cas de l'union, les supports des requêtes dans $\mathcal{F}$ doivent être calculés, et cela peut être fait en une seule passe sur les données.

Finalement, pour les requêtes qui ne sont pas dans $\mathcal{F}$, il n'y a pas d'optimisation possible. Le calcul de telles requêtes necéssitera l'utilisation de l'approche standard [2].

3.5 Conclusion

Nous avons présenté une nouvelle approche pour l'extraction itérative de règles d'association par la composition de contextes d'extraction en utilisant les opérations de l'algèbre relationnelle. Cette approche permet d'une part d'introduire un langage de manipulation des contextes, ce qui donne à l'utilisateur une plus grande marge de manœuvre dans la spécification de sa requête d'extraction, et d'autre part de réduire le temps de calcul des requêtes fréquentes des contextes composés en utilisant les requêtes fréquentes stockées des contextes composants.
Nous avons vu qu'il suffisait de stocker et d'utiliser la frontière positive des ensembles de requêtes fréquentes pour optimiser le calcul des requêtes fréquentes lorsque de nouveaux contextes sont définis par composition d'anciens. Cette optimisation étant basée sur la comparaison des requêtes candidates avec toutes les requêtes de la frontière positive, nous proposerons dans la partie sur l'implémentation une méthode efficace permettant d'effectuer cette comparaison, en utilisant une représentation inverse de la frontière positive. Dans cette même partie sur l'implémentation, nous montrerons par des tests expérimentaux les gains de notre approche.

Cependant, un des principaux problèmes qui se posent est la redondance dans le stockage des requêtes fréquentes des contextes composants, étant donné que ces derniers ne sont pas indépendants. Pour résoudre ce problème, nous introduisons la notion de représentation condensée d'un ensemble de réponses à des requêtes d'extraction que nous verrons dans la partie correspondante.

4 Implémentation

Cette partie est consacrée à l'implémentation de l'approche algébrique que nous avons vue dans le chapitre précédent sur la composition de contextes d'extraction pour une extraction efficace des règles d'associations.

4.1 Introduction

L'implémentation que nous allons présenter entre dans le cadre des algorithmes d'extraction par niveau. Rappelons que l'extraction par niveau constitue une des approches principales de la découverte de motifs fréquents. Elle consiste à parcourir le treillis des motifs niveau par niveau [63]. Durant chaque itération correspondant à un niveau, l'ensemble des motifs candidats est crée en joignant les candidats fréquents découverts lors de l'itération précédente. Ensuite, il y a la phase de coupure qui consiste à vérifier pour tous les candidats que les motifs qui leur sont plus généraux sont fréquents. Les supports de tous les motifs candidats sont calculés et les motifs non fréquents sont élagués.

Notre approche est une extraction itérative de motifs fréquents qui utilise les résultats stockés des extractions antérieures pour optimiser le calcul d'une nouvelle extraction. Cette nouvelle extraction, est, comme nous l'avons déjà vu, une composition des extractions antérieures (composition des contextes d'extraction). C'est ainsi que nous ajoutons à l'algorithme d'extraction par niveau une nouvelle phase d'optimisation.

Nous parlerons dans un premier temps de l'architecture de l'implémentation, ensuite des spécificités de notre approche et finalement des tests effectués et résultats expérimentaux.

4.2 Architecture de l'implémentation

Nous allons présenter deux schémas qui illustrent l'implantation du système et sa mise en œuvre. Le premier (voir figure III.2) décrit l'architecture du système. Suite la connexion à la base de données via ODBC (par le menu Fichier "Ouvrir"), l'utilisateur par l'intermédiaire de l'interface, spécifie le contexte d'extraction (requête de référence, requête de base, les attributs de référence (qui sont les attributs de la requête de référence), les attributs du domaine d'intérêt et le seuil minimal du support. Il exécute ensuite sa requête d'extraction par le moteur d'extraction (menu "Extraction Executer") et obtient en sortie la réponse à la requête d'extraction, réponse qui est stockée dans la base de motifs. Etant donné que le contexte de la nouvelle requête d'extraction peut être une composition de contextes déjà calculés et stockés, l'interface utilisateur dispose d'une fenêtre où apparaissent les informations sur les différents contextes d'extraction stockés (le numéro de la requête d'extraction, la requête de référence, la requête de base, les attributs de référence, les attributs du domaine d'intérêt et le seuil minimal de support). Soulignons que deux opérateurs de manipulations de contextes ont été implantés : la sélection et la jointure. C'est ainsi que l'interface utilisateur dispose de deux champs pour les numéros de contextes à joindre lorsque l'opération de composition de contextes est la jointure (menu "Extraction Joindre"). Lorsque l'opération est la sélection (menu "Extraction Selection"), l'utilisateur doit indiquer dans le champ "selection" la condition de sélection. Il est également possible pour l'utilisateur de faire la combinaison de la sélection et de la jointure (menu "Extraction Combiner").

Etant donné un contexte $\mathcal{C} = \langle R, B, \Sigma \rangle$, nous construisons le treillis des requêtes de la forme $R \bowtie \sigma_S(B)$, où S est une conjonction de conditions de sélection atomiques ($S \in \Sigma^*$). Au niveau 1 du treillis, nous avons des requêtes avec une condition de sélection atomique, au niveau 2, nous avons des requêtes où S est une conjonction de deux conditions de sélections atomiques, et ainsi de suite. Pour cela, l'algorithme passe par différentes phases. Le deuxième schéma (voir figure

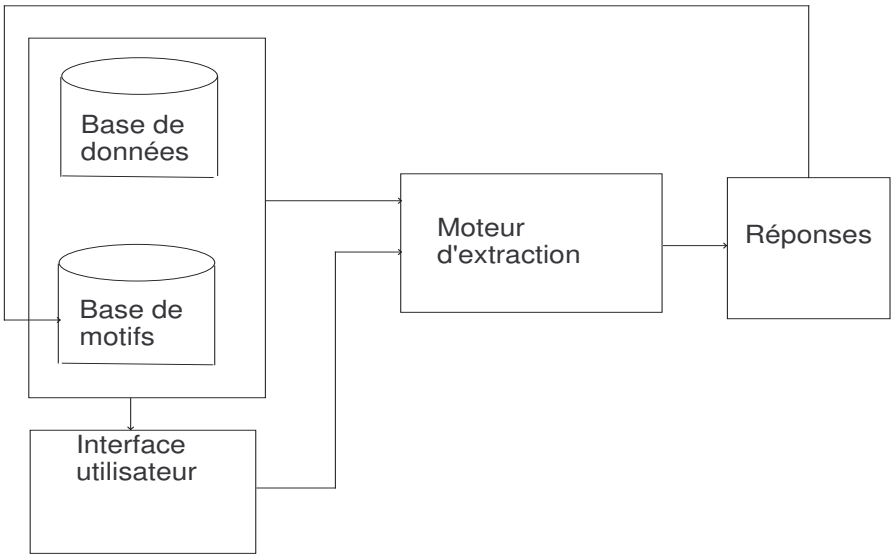


FIG. III.2 – Architecture du système

III.3) montre les différentes phases de cet algorithme d'extraction par niveau que nous utilisons, et surtout précise à quel endroit intervient la phase d'optimisation ou d'élagage que nous avons introduite (qui est mise en relief dans le schéma).
La phase d'*initialisation* initialise le treillis en créant le niveau 1. Celui-ci correspond aux requêtes candidates avec une condition de sélection atomique. Il y a deux types d'initialisation qui se distinguent par le fait que l'utilisateur exécute ou non une requête d'extraction itérative. Lorsque l'utilisateur n'utilise pas les résultats déjà stockés (menu "Extraction Executer"), les requêtes construites sont de la forme $R \bowtie \sigma_{A=a}(B)$, où A est un attribut du domaine d'intérêt Σ et $a \in dom(A)$. Lorsque l'utilisateur effectue une extraction itérative par composition de contextes (menu "Extraction Selection" ou "Extraction Joindre" ou "Extraction Combiner"), le premier niveau est créé à partir des frontières positives des réponses aux requêtes d'extraction correspondantes. Nous parlerons de façon plus détaillée de ce deuxième cas dans la sous-section 4.3.2.
La phase *Evaluation + Fréquent* permet d'évaluer le niveau i du treillis et de déterminer les candidats fréquents. La phase d'évaluation se caractérise dans notre contexte comme suit : l'évaluation des supports des requêtes candidates d'un niveau se fait en une seule passe sur la jointure de la requête de référence et de la requête de base. Pour chaque ligne de cette table, il est possible d'utiliser les mêmes techniques de hachage que celles utilisées dans *Apriori* [2] pour tester si le support d'une requête candidate doit être incrémenté ou non de 1. Toutefois, dans notre cadre, si K désigne l'ensemble des variables de la requête de référence, il ne faut pas incrémenter le support deux fois pour deux lignes distinctes de même valeur de K. Le parcours de la jointure selon l'ordre croissant des valeurs de K permet de détecter simplement les doublons pour chaque requête candidate.
La phase de *construction*, qui correspond à la phase de génération dans *Apriori* construit le niveau i à partir du niveau $i-1$. Une requête candidate de niveau i est construite à partir de deux requêtes $R \bowtie \sigma_{S_1}(B)$ et $R \bowtie \sigma_{S_2}(B)$ de niveau i-1 si ces S1 et S2 ont i-2 premières conditions de sélections atomiques communes. A ce titre, il faut souligner les travaux sur le treillis relationnel de [20] où les auteurs définissent l'espace de recherche à parcourir lorsque les données en entrée

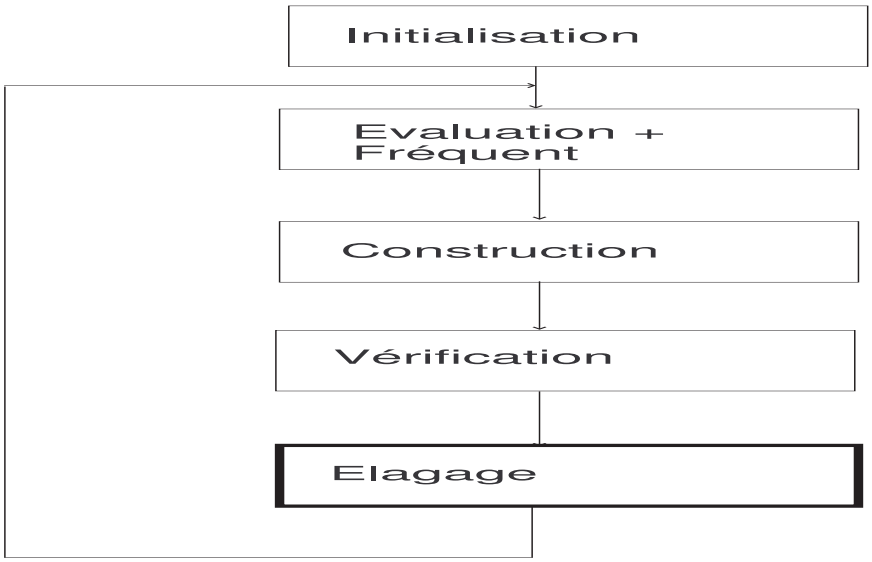


FIG. III.3 – Les différentes phases de l'algorithme

ne sont pas des données binaires, mais des données sous forme relationnelle. Signalons toutefois que notre implémentation n'est pas basée sur ces travaux.
La phase de *vérification* est comme celle définie dans *Apriori*. On vérifie que toutes les requêtes qui incluent la requête candidate sont fréquentes. S'il existe une de ces requêtes qui n'est pas fréquente, alors la requête candidate est supprimée de l'ensemble des requêtes candidates.
La phase d'*élagage* qui constitue notre contribution sera l'objet de la section suivante.

4.3 Spécificités de notre approche

Comme nous l'avons déjà vu, nous proposons dans notre approche le stockage de la frontière positive des réponses aux requêtes d'extraction. Nous allons donc voir pratiquement comment se fait ce stockage. D'autre part, étant donné que nous utilisons la frontière positive pour l'optimisation du calcul de nouvelles requêtes d'extraction, il faut une utilisation efficace de celle-ci.
Nous avons vu que, pour un contexte donné $\mathcal{C} = \langle R, B, \Sigma \rangle$, les requêtes de $\mathcal{Q}(\mathcal{C})$ à considérer sont toutes de la forme $R \bowtie \sigma_S(B)$, où S est une conjonction de conditions de sélection atomiques. Par conséquent, pour un contexte donné $\mathcal{C}$, chaque requête $R \bowtie \sigma_S(B)$ de $\mathcal{Q}(\mathcal{C})$ peut être assimilée à l'ensemble des conditions de sélection atomiques (ou élémentaires) apparaissant dans S. Afin de ne pas compliquer les notations, cet ensemble sera également noté S dans ce qui suit. En utilisant cette convention d'écriture, il est facile de voir que si on considère deux requêtes $q_1 = R \bowtie \sigma_{S_1}(B)$ et $R \bowtie \sigma_{S_2}(B)$ de $\mathcal{Q}(\mathcal{C})$, assimilées respectivement aux ensembles S_1 et S_2, alors on a :

$$Si\ S_1 \subseteq S_2,\ alors\ q_2 \sqsubseteq q_1.$$

On pourra ainsi considérer le treillis des conditions de sélection, où au niveau 1, les candidats sont des conditions de sélection composées d'une condition atomique, au niveau 2 de deux conditions atomiques, etc...

4.3.1 Stockage de la frontière positive

Dans cette sous-section, nous montrons comment les ensembles de requêtes fréquentes des frontières positives peuvent être stockés dans une base de données relationnelles que nous appelons $DMining$. Cette base de données contient les quatre tables suivantes :

- $Freq(CId, RQuery, BQuery, DId, Minsup)$, où :
  - CId stocke l'identifiant du contexte,
  - $RQuery$ et $BQuery$ stockent les expressions de l'algèbre relationnelle qui définissent respectivement la référence et la base du contexte,
  - DId stocke l'identifiant du domaine,
  - $Minsup$ stocke le seuil de support,
  - $Border$ est un booléen qui indique si seule la frontière est stockée ou pas.
- $Dom(DId, Name)$, où DId est l'identifiant défini ci-dessus et $Name$ est le nom de l'attribut qui apparaît dans le domaine. Nous supposons ici implicitement que si un attribut A est dans le domaine d'un contexte, alors les conditions de sélection correspondantes sont de la forme $A = a$, pour $a \in dom(A)$.
- $Query(CId, QId, Sup, Border)$, où CId est l'identifiant du contexte, QId est l'identifiant de requête, Sup stocke le support de la requête, et $Border$ indique si la requête appartient ou non à la frontière positive.
- $Cond(CId, QId, Name, Const)$, où CId et QId sont indiqués ci-dessus, $Name$ est un nom d'attribut et $Const$ est une constante du domaine de cet attribut.

Exemple 4.1 *La figure III.4 montre comment les frontières positives de l'ensemble des requêtes fréquentes de deux contextes sont stockées. Dans notre exemple, la requête d'identifiant* 3 *est une requête fréquente du contexte d'identifiant* 1. *Cette requête est définie par l'expression suivante :*

$$q_3 = AllCust \bowtie \sigma_{Cjob=Lawyer \wedge Ptype=Bread}(Cust \bowtie Sales \bowtie Prod).$$

4.3.2 Utilisation efficace des requêtes fréquentes stockées

Nous allons voir dans cette sous-section, lorsque la nouvelle requête d'extraction est définie par composition de contextes, comment le premier niveau du treillis des requêtes est construit à partir des frontières positives, et de plus comment l'élagage des requêtes candidates est effectué.

Création du premier niveau du treillis résultant de la composition de contextes.
Supposons qu'un certain nombre de requêtes d'extraction ont été effectuées et que leurs frontières positives sont stockées.

B-sélection

Considérons dans un premier temps le cas de la b-sélection. Soit $\mathcal{C} = \langle R, B, \Sigma \rangle$ un contexte et S une condition de sélection. Comme nous l'avons déjà vu la *b-sélection de $\mathcal{C}$ sur S,* notée $\sigma^b_S(\mathcal{C})$, est définie par : $\sigma^b_S(\mathcal{C}) = \langle R, \sigma_S(B), \Sigma \rangle = \mathcal{C}_2$.
Soit $Q_2 \in \mathcal{Q}(\mathcal{C}_2)$ et S_2 une condition de sélection dans Σ^*.

$$Q_2 = \sigma_{S_2}(R \bowtie \sigma_S(B)) \equiv \sigma_{S_2 \wedge S}(R \bowtie B).$$

Nous distinguons deux cas, selon que S est ou non dans Σ^*.
1er cas : Si $S \in \Sigma^*$, $Q_2 \in \mathcal{Q}(\mathcal{C})$.

Freq					
Cid	RQuery	BQuery	Did	Minsup	Border
1	AllCust	Join(Cust,Sales,Prod)	1	0.1	true
2	AllCust	Join(Cust,Sales,Prod)	2	0.1	true

Query			
Cid	Qid	Sup	Border
1	1	0.3	true
1	2	0.2	true
1	3	0.4	true
1	4	0.2	true
2	5	0.3	true
2	6	0.2	true
2	7	0.3	true

Dom	
DId	Name
1	Cjob
1	Caddr
1	Ptype
2	Cjob
2	Caddr
2	Sname

Cond			
CId	QId	Name	Const
1	1	Cjob	Professeur
1	1	Caddr	Paris
1	2	Cjob	Avocat
1	2	Caddr	Blois
1	2	Ptype	Lait
1	3	Cjob	Avocat
1	3	Ptype	Pain
1	4	Cjob	Professeur
1	4	Ptype	Pain
2	5	Sname	Auchan
2	5	Caddr	Blois
2	5	Cjob	Avocat
2	6	Sname	Casino
2	6	Caddr	Blois
2	6	Cjob	Avocat
2	7	Caddr	Blois
2	7	Sname	Leclerc

FIG. III.4 – Stockage des requêtes fréquentes de deux contextes d'identifiants 1 et 2

Nous savons d'après les résultats théoriques obtenus que pour tout seuil de support α et pour toute instance I, on a :

$$freq_\alpha(\sigma_S^b(\mathcal{C}), I) \sim \tau_S(freq_\alpha(\mathcal{C}, I)).$$

Comme seule la frontière positive a été stockée, pour construire le premier niveau du treillis, il suffit alors de récupérer l'ensemble $\tau_S(Bd_\alpha^+(\mathcal{C}, I))$ qui correspond à la réduction sur S de l'ensemble des éléments de la frontière positive de $freq_\alpha(\mathcal{C}, I)$.
Pour le treillis des conditions de sélection, cela correspond à récupérer l'ensemble $Cond_S$ des conditions de sélection qui contiennent la condition de sélection S et récupérer les différentes conditions de sélection atomiques (en dehors de celles dans S) de $Cond_S$. Si on appelle $Atom_S$ cet ensemble de conditions atomiques, le premier niveau du treillis sera constitué des conditions de sélection $(A = a) \in Atom_S$ correspondant aux requêtes de la forme $R \bowtie \sigma_{A=a}(\sigma_S(B))$.

Exemple 4.2 *Reprenons la base de données DBSALES décrite dans le chapitre sur la composition de contextes d'extraction pour une extraction efficace de règles d'associations. Considérons que la requête d'extraction correspondant au contexte suivant est déjà calculée :* $\mathcal{C}_1 = \langle AllCust, B_1, \Sigma_1 \rangle$*, où* $AllCust = \pi_{Cid}(Cust)$*,* $B_1 = Cust \bowtie Sales \bowtie Prod$ *et* $\Sigma_1 = \{Cjob = Professeur, Cjob = Avocat, Ptype = Thé, Ptype = Lait, Caddr = Blois, Caddr = Paris\}$*, et que sa frontière positive est constituée des requêtes :* $Q_i^+ = AllCust \bowtie \sigma_{S_i}(B_1)$ *où* $i = 1, ..., 4$ *avec :*

1 $S_1 = \{Caddr = Blois\}$
2 $S_2 = \{Ptype = Lait\}$
3 $S_3 = \{Cjob = Avocat, Ptype = Pain\}$
4 $S_4 = \{Caddr = Orléans, Ptype = Pain\}$

Considérons maintenant la b-sélection sur $Cjob = Avocat$ *de* $\mathcal{C}_1$*.* $Cjob = Avocat \in \Sigma^*$*. Pour construire le premier niveau du treillis correspondant, on récupère toutes les conditions de sélection qui contiennent* $Cjob = avocat$*, et on prend leurs sous-ensembles constituées d'une condition de sélection atomique. Dans le cas présent, on récupère seulement* $S_3 = Cjob = avocat, Ptype = Pain$*. Le premier niveau du treillis sera donc constitué par le candidat* $Ptype = Pain$*. La requête correspondante est* $AllCust \bowtie \sigma_{Ptype=Pain}(B_1)$*, qui correspond ici à l'ensemble des avocats qui ont acheté du pain.*

2ème cas : $S \notin \Sigma^*$, nous savons d'après les résultats théoriques obtenus que pour tout seuil de support α et pour toute instance I, on a :

$$freq_\alpha(\sigma_S^b(\mathcal{C}), I) \widetilde{\subseteq} \sigma_S(freq_\alpha(\mathcal{C}, I)).$$

Etant donné que nous avons seulement la frontière positive, nous récupérons l'ensemble des requêtes $\sigma_S(Bd_\alpha^+(\mathcal{C}, I))$ pour construire le premier niveau du treillis. Cela correspond pour le treillis des conditions de sélection, à récupérer l'ensemble $Cond$ de toutes les conditions de sélections des requêtes de $Bd_\alpha^+(\mathcal{C}, I)$. Si on appelle $Atom$ l'ensemble des conditions de sélection atomiques de $Cond$, le premier niveau du treillis sera constitué par les des conditions de sélection $(A = a) \in Atom$ correspondant aux requêtes de la forme $R \bowtie \sigma_{A=a}(\sigma_S(B))$.

Jointure de deux contextes

Considérons maintenant la jointure de deux contextes. Soit $\mathcal{C}_1 = \langle R, B_1, \Sigma_1 \rangle$ et $\mathcal{C}_2 = \langle R, B_2, \Sigma_2 \rangle$ deux contextes de même référence R. Nous savons d'après les résultats théoriques obtenus que pour tout seuil de support α et pour toute instance I, on a :

$$freq_\alpha(\mathcal{C}_1 \bowtie_b \mathcal{C}_2, I) \widetilde{\subseteq} freq_\alpha(\mathcal{C}_1, I) \bowtie freq_\alpha(\mathcal{C}_2, I).$$

Soit $Bd^+_\alpha(\mathcal{C}_1)$ et $Bd^+_\alpha(\mathcal{C}_2)$ les frontières positives de $freq_\alpha(\mathcal{C}_1)$ et $freq_\alpha(\mathcal{C}_2)$ respectivement. Considérons les treillis des conditions de sélection pour voir quelles sont les conditions de sélections atomiques qui seront conservées pour construire le premier niveau du treillis correspondant à la jointure des contextes de $\mathcal{C}_1$ et $\mathcal{C}_2$. Soit $bd1$ l'ensemble des conditions de sélection correspondant aux requêtes de $Bd^+_\alpha(\mathcal{C}_1)$ et $bd2$ l'ensemble des conditions de sélection correspondant aux requêtes de $Bd^+_\alpha(\mathcal{C}_2)$. Soit $A = a$ une condition de sélection atomique, où A est un attribut du domaine d'intérêt de Σ_1 ou de Σ_2. Différents cas se présentent :

- si $A \in att(\Sigma_1) \cap att(\Sigma_2)$, alors $A = a$ appartient au premier niveau si et seulement si $A = a \in bd1 \cap bd_2$;
- si $A \in att(\Sigma_1) \backslash att(\Sigma_2)$, alors $A = a$ appartient au premier niveau ;
- si $A \in att(\Sigma_2) \backslash att(\Sigma_1)$, alors $A = a$ appartient au premier niveau.

Exemple 4.3 *Considérons qu'on dispose de deux contextes $\mathcal{C}_1$ et $\mathcal{C}_2$ pour lesquels $bd1$ et $bd2$ sont donnés comme suit :*

$$bd1 = \{\ Ptype = Pain, Cjob = Prof\ \}$$
$$bd2 = \{\ Ptype = Lait, Caddr = Lyon, Ptype = Pain, Caddr = Paris\ \}$$

Supposons que $\Sigma_1 = \{Ptype = *, Cjob = *\}$ et $\Sigma_2 = \{Ptype = *, Caddr = *\}$.
Le premier niveau de la jointure est constitué par les candidats suivants : $Ptype = Pain$ (puisque la condition de sélection atomique appartient à $bd1$ et $bd2$), $Cjob = Prof$, $Caddr = Lyon$, $Caddr = Paris$. Par contre, le candidat $Ptype = Lait$ ne fait pas partie du premier niveau, parce la condition de sélection atomique est présente seulement dans $bd2$, alors que $Ptype \in att(\Sigma_1) \cap att(\Sigma_2)$.

Elagage des candidats

Dans le chapitre sur la composition de contextes d'extraction pour une extraction efficace des règles d'association, nous avons vu que la frontière positive des ensembles de requêtes peut être utilisée pour optimiser le calcul des ensembles de requêtes fréquentes lorsque de nouveaux contextes sont définis à partir d'anciens en utilisant la $b - sélection$ ou la $jointure$. Cependant, il a été noté que cette optimisation est basée sur la comparaison d'une requête candidate avec toutes les requêtes de la frontière positive.

Nous proposons une méthode efficace pour effectuer ce test, en utilisant une *représentation inverse* de la frontière positive. Avant de donner plus de détails sur la représentation inverse, rappelons que dans le treillis des conditions de sélection, une condition de sélection S est un ensemble de conditions atomiques. De façon plus précise, la représentation inverse d'une frontière donnée consiste à considérer séparément, pour chaque condition de sélection atomique s, l'ensemble des identifiants des requêtes dont les conditions de sélection contiennent s. Ainsi, étant donnée une condition de sélection S', l'ensemble des requêtes dont les conditions de sélection contiennent S' est simplement l'intersection des ensembles associés aux conditions de sélection atomiques de S'. Autrement dit, les requêtes dont l'identifiant est dans l'intersection ont une condition de sélection contenant les conditions atomiques. Nous donnons ci-dessous une illustration de notre méthode.

Exemple 4.4 *Comme nous l'avons déjà vu dans l'exemple 4.1, la figure III.4 montre comment les frontières positives de l'ensemble des requêtes fréquentes de deux contextes $\mathcal{C}_1$ et $\mathcal{C}_2$ sont stockées. Nous donnons ci-dessous la représentation inverse de la frontière positive de l'ensemble*

des requêtes fréquentes de $\mathcal{C}_1$:

$Cjob = Professeur$	*1,4*
$Caddr = Paris$	*1*
$Cjob = Avocat$	*2,3*
$Caddr = Blois$	*2*
$Ptype = Lait$	*2*
$Ptype = Pain$	*3,4*

La première ligne de la représentation inverse indique que la condition de sélection $Cjob = Professeur$ est présente dans les requêtes d'identifiant 1 et 4.
Supposons maintenant qu'en plus des attributs Cid, $Cjob$ et $Caddr$, la table $Cust$ contienne aussi l'attribut $Csal$, qui peut prendre la valeur "Haut", où $Csal = Haut$ veut dire que le Client Cid a un salaire élevé.
Supposons que la frontière positive, dont la représentation inverse est donnée ci-dessus, est obtenue après une sélection sur le contexte par $Csal = Haut$.
Deux cas se distinguent : $Csal \in \Sigma^*$ et $Csal \notin \Sigma^*$.
1er cas : $Csal \in \Sigma^*$
Dans ce cas, considérons une requête candidate $q_1 = R \bowtie \sigma_{Cjob=Professeur \wedge Caddr=Paris}(\sigma_{Csal=Haut}(B))$. Alors $q_1 = R \bowtie \sigma_{Cjob=Professeur \wedge Caddr=Paris \wedge Csal=Haut}(B)$ est une requête qui a déjà été considérée. Si elle est sous la frontière positive, elle est fréquente, sinon elle ne l'est pas. Pour savoir si elle est sous la frontière positive, on fait l'intersection des ensembles d'identifiants associés à $Cjob = Professeur$ et $Caddr = Paris$. L'intersection est égale à $\{1,4\} \cap \{1\} = 1$. Elle est non vide, donc la requête est fréquente. Si l'intersection est vide, la requête n'est pas fréquente.
2ème cas : $Csal \notin \Sigma^*$
Dans ce cas la requête candidate est $q_1 = R \bowtie \sigma_{Cjob=Professeur \wedge Caddr=Paris}(\sigma_{Csal=Haut}(B))$. Dans ce cas, si l'intersection des ensembles d'identifiants de $Cjob = Professeur$ et $Caddr = Paris$ est non vide, on peut dire que la requête $q_2 = R \bowtie \sigma_{Cjob=Professeur \wedge Caddr=Paris}(B)$ contient une requête de la frontière positive déjà calculée. Donc q_2 est fréquente, mais on ne peut rien dire de plus.
Si l'intersection calculée est vide, on peut conclure que q_2 est non fréquente, et donc que q_1 est non fréquente puisque q_1 est incluse dans q_2. Par exemple, considérons la requête $q_3 = R \bowtie \sigma_{Cjob=Professeur \wedge Ptype=Lait}(\sigma_{Csal=Haut}(B))$. L'intersection des ensembles d'identifiants de $Cjob = Professeur$ et $Ptype = Lait$ est égale à $\{1,4\} \cap \{2\} = \emptyset$. Par conséquent, on peut dire que q_3 n'est pas fréquente, et par suite cette requête est supprimée de l'ensemble des requêtes candidates.

Soit $\mathcal{C}_1 = \langle R, B_1, \Sigma_1 \rangle$ et $\mathcal{C}_2 = \langle R, B_2, \Sigma_2 \rangle$ deux contextes, et soit $\mathcal{C}_{new}$ le contexte défini par $\mathcal{C}_{new} = \mathcal{C}_1 \bowtie_b \mathcal{C}_2$. Etant donné un seuil de support α et une instance I, soit Bd_1^+ et Bd_2^+ les frontières positives de $freq_\alpha(\mathcal{C}_1, I)$ et $freq_\alpha(\mathcal{C}_2, I)$, respectivement.

Pour $i = 1, 2$, soit $QId(i, A = a)$ l'ensemble des identifiants de toutes les requêtes dans Bd_i^+ dont la condition de sélection contient $A = a$. Comme nous l'avons vu dans le chapitre sur la composition de contextes d'extraction pour une extraction efficace des règles d'association, une requête candidate $q_{new} = R \bowtie \sigma_S(B_1 \bowtie B_2)$ de $\mathcal{C}_{new}$ est fréquente si et seulement si il existe q_1^+ dans Bd_1^+ et q_2^+ dans Bd_2^+ telles que $q_1^+ \bowtie q_2^+ \sqsubseteq q_{new}$.

D'autre part, la condition de sélection $S = (A_1 = a_1) \wedge \ldots \wedge (A_n = a_n)$ de q_{new} peut être réécrite sous la forme $S = S_1 \wedge S_2$ où $S_1 = (A_{11} = a_{11}) \wedge \ldots \wedge (A_{1p} = a_{1p})$ est dans Σ_1^* et $S_2 = (A_{21} = a_{21}) \wedge \ldots \wedge (A_{2q} = a_{2q})$ est dans Σ_2^*.

Alors, on peut montrer qu'il existe q_1^+ dans Bd_1^+ et q_2^+ dans Bd_2^+ telles que $q_1^+ \bowtie q_2^+ \sqsubseteq q_{new}$

si et seulement si :

$$QId(1, A_{11} = a_{11}) \cap \ldots \cap QId(1, A_{1p} = a_{1p}) \neq \emptyset \quad \text{et}$$
$$QId(2, A_{21} = a_{21}) \cap \ldots \cap QId(2, A_{2q} = a_{2q}) \neq \emptyset.$$

Comme le montre l'exemple suivant, ces deux conditions peuvent être testées efficacement en utilisant la représentation inverse introduite précédemment.

Exemple 4.5 *Supposons qu'on utilise la base de données de l'exemple courant du chapitre sur la composition de contextes d'extraction pour une extraction efficace des règles d'association. Considérons les frontières positives Bd_1^+ et Bd_2^+ représentées à la figure III.4. Soit $\mathcal{C}_{new} = \mathcal{C}_1 \bowtie_b \mathcal{C}_2$, et soit $q_{new} = AllCust \bowtie \sigma_S(Cust \bowtie Sales \bowtie Prod \bowtie Store)$ avec $S = (Cjob = Avocat) \wedge (Caddr = Paris)$.*

Avec la notation introduite, on a $S = S_1 = S_2$, et puisque :

$$QId(1, Cjob = Avocat) = \{2, 3\}$$
$$QId(1, Caddr = Paris) = \{1\}$$
$$QId(1, Cjob = Avocat) \cap QId(1, Caddr = Paris) = \{2, 3\} \cap \{1\} = \emptyset,$$

on peut dire que q_{new} n'est pas *fréquente. Ainsi, cette requête est supprimée de l'ensemble des candidats de niveau* 2.

Considérons maintenant $q_{new} = AllCust \bowtie \sigma_S(Cust \bowtie Sales \bowtie Prod \bowtie Store)$ avec $S = (Cjob = Avocat) \wedge (Ptype = Lait) \wedge (Sname = Ikea)$ comme une requête candidate de $\mathcal{C}_{new}$. On a alors : $S_1 = (Cjob = Avocat) \wedge (Ptype = Lait)$ et $S_2 = (Cjob = Avocat) \wedge (Sname = Ikea)$, et ainsi :

$$QId(1, Ptype = Lait) = \{2\}$$
$$QId(2, Cjob = Avocat) = \{5, 6\}$$
$$QId(2, Sname = Ikea) = \{6\}$$
$$QId(1, Cjob = Avocat) \cap QId(1, Ptype = Lait) = \{2, 3\} \cap \{2\} \neq \emptyset \quad et$$
$$QId(2, Cjob = Avocat) \cap QId(2, Sname = Ikea) = \{5, 6\} \cap \{6\} \neq \emptyset.$$

Par conséquent, q_{new} peut être fréquente et doit être gardée parmi les requêtes candidates de niveau 3.

4.4 Tests effectués

Dans cette section, nous présentons les résultats expérimentaux de nos tests qui ont été effectués en utilisant des données synthétiques, générées comme dans [2]. Cependant, le générateur utilisé dans [2] a été modifié de sorte à prendre en compte le stockage des données pas sur une seule table, mais sur plusieurs tables d'une base de données. Nous parlerons ainsi dans un premier temps du générateur de données, ensuite des résultats expérimentaux et finalement des perspectives.

4.4.1 Génération des données

Comme indiqué plus haut, les données sont générées comme dans [2], mais notre générateur se distingue par le fait que les données sont stockées sur plusieurs tables.
Nous donnons ci-dessous un résumé de la génération des données.
Il s'agit, étant donné un contexte $\mathcal{C}$ et un seuil minimal de support, de générer un ensemble de motifs fréquents appartenant à $Q(\mathcal{C})$. Cependant, la génération des motifs se fait de manière

aléatoire. Reprenons le treillis des conditions de sélection pour donner une illustration de la génération. Un candidat étant un ensemble de conditions de sélections atomiques $s_1, ..., s_k$, il s'agit ainsi d'engendrer des données permettant d'obtenir différentes conditions de sélection atomiques qui constitueront les requêtes candidates et donc les requêtes fréquentes. Pour ce faire, notre système, dans un premier temps, affecte aléatoirement la taille moyenne N des candidats (i.e. le nombre de conditions de sélection atomiques). Ensuite, on tire au hasard N attributs du domaine, et pour chaque attribut A, on tire au hasard une valeur dans $dom(A)$ pour former les différentes conditions de sélection atomiques. On insère alors dans les différentes tables de la base de données les tuples tels que le motif apparaisse suffisamment de fois pour être fréquent.

4.4.2 Résultats expérimentaux

Dans cette partie, nous allons discuter des résultats expérimentaux en nous basant sur les expériences que nous avons effectuées respectivement pour la sélection et la jointure. Les tests ont été effectués sur une base de données, appelée $DBTest$ et contenant trois tables de schémas $Cust(x_1, x_2, ..., x_{10})$, $Sales(x_1, y_1, z_1, d_1)$ et $Prod(y_1, y_2, ..., y_{10})$. Différentes instances de cette base de données ont été considérées en changeant différents paramètres du générateur. D'abord, nous avons fait varier la cardinalité des domaines des motifs de 10, 40 et 80 pour un seuil de support minimum égal à 0.05 et une taille moyenne des motifs de 4. Nous obtenons ainsi les instances I10-1-5, I40-1-5 et I80-1-5 pour lesquelles les cardinalités des tables $Cust$, $Sales$ et $Prod$ sont respectivement de 2000, 6000 et 54000. Ceci constitue un premier groupe pour lequel nous ferons des comparaisons concernant les gains en temps de calcul.
Pour le deuxième groupe, nous avons fait varier la taille moyenne des motifs, et nous obtenons les instances TM3-1-5 et TM5-1-5. Le troisième groupe (NM50-1-20, NM100-1-10 et NM150-1-6), est constitué des instances pour lesquelles le nombre de motifs varie, de même que le seuil minimal de support. Enfin, nous avons une instance I-10-3 que nous comparerons avec I-10-5 pour voir l'influence du changement du seuil de support pour une même instance.
La figure III.5 montre le tableau des différentes instances du premier et du deuxième groupe. Le tableau des gains en temps de calcul obtenus avec notre méthode itérative sur ces différentes instances est présenté à la figure III.6. Le tableau de la figure III.7, présente la variation des paramètres liés à l'élagage pour cette même expérience. Nous utiliserons ce tableau pour expliquer le tableau de variation des gains de la figure III.6. Il s'agit ainsi de trois tableaux qui sont liés à une même expérience.
Nous procédons de la même façon avec les instances du quatrième groupe (voir figure III.8). Nous avons un tableau des gains en temps de calcul (voir figure III.9) et un tableau des paramètres de l'élagage (voir figure III.10).
Il est important de souligner que tous les gains obtenus correspondent au cas où les frontières déjà calculées nécessaires à l'élagage sont dans un cache mémoire.

Sélection

Rappelons que pour la sélection, nous pouvons avoir deux cas. Soit $\mathcal{C} = \langle R, B, \Sigma \rangle$ un contexte et S une condition de sélection. Si $S \in \Sigma^*$, alors on sait que $freq_\alpha(\sigma_S^b(\mathcal{C}))$ est un sous-ensemble de $freq_\alpha(\mathcal{C})$. L'ensemble des motifs fréquents de $freq_\alpha(\sigma_S^b(\mathcal{C}))$ est généré à partir de la frontière positive et l'évaluation des supports des motifs se fait en une seule passe sur la base de données. C'est ce que nous appelons évaluation en une seule passe. Par contre, si $S \notin \Sigma^*$, on ne sait pas si les motifs sont fréquents. Il faut les générer et faire le calcul des supports niveau par niveau. C'est l'évaluation en N passes, où N est le niveau maximal du treillis.
Les tests pour la sélection ont été faits comme suit : On considère d'abord le contexte $\mathcal{C}$ suivant :

	I10-1-5	I40-1-5	I80-1-5	TM3-1-5	TM5-1-5
Taille moyenne des motifs	4	4	4	3	5
Cardinalité des domaines des attributs	10	40	80	40	40
Nombre de motifs pour la génération	100	100	100	100	100
Cardinalité de Cust	2000	2000	2000	2000	2000
Cardinalité de Prod	6000	6000	6000	6000	6000
Cardinalité de Sales	54000	54000	54000	54000	54000
Seuil minimal de support	5%	5%	5%	5%	5%

FIG. III.5 – Instances du premier groupe (I10-1-5, I40-1-5 et I80-1-5) et du deuxième (TM3-1-5 et TM5-1-5) : variation des cardinalités des domaines des attributs (I10-1-5, I40-1-5 et I80-1-5) et de la taille moyenne (TM3-1-5 et TM5-1-5).

Temps d'évaluation non incrémental (s)	249	182	577	191	420
Temps total non incrémental	282	189	599	197	436
Temps d'évaluation incrémental en N passes (s)	124	55	58	52	127
Temps d'évaluation incrémental en 1 passe (s)	86	32	41	22	100
Temps total d'élagage (s)	4	1	7	1	1
Temps total incrémental en N passes (s)	224	75	92	59	141
Temps total incrémental en 1 passe (s)	193	53	81	30	113
Gain en temps en N passes	45,0%	69,3%	88,0%	72,6%	69,5%
Gain en temps en 1 passe	56,0%	81,0%	89,8%	87,3%	75,9%

FIG. III.6 – Tests pour la sélection : Gains en temps de calcul pour le premier et le deuxième groupe.

- Requête de référence : $AllSales = \pi_{x_1}(Sales)$;
- Requête de base : $Cust \bowtie Sales \bowtie Prod$;
- $\Sigma = \{x2 = *, x3 = *, x4 = *, y_1 = *, y_2 = *, y_3 = *\}$, où $x = *$ veut dire que x peut prendre toutes les valeurs qui sont dans son domaine.

Ceci constitue une première requête d'extraction que nous calculons et stockons. On calcule ensuite l'ensemble des requêtes de $\sigma_S^b(C)$, où S est la condition de sélection vide. Ce calcul incrémental qui utilise les résultats de la première extraction fera l'objet d'une comparaison avec le premier calcul. Soulignons qu'avec la condition de sélection vide, l'évaluation peut se faire en une passe. Cependant nous avons également calculé les gains en temps en effectuant une évaluation en N passes pour tester ce que seraient les gains si $S \notin \Sigma^*$.

Considérons le premier groupe (I10-1-5, I40-1-5, I80-1-5). Ici, on note des gains en temps de calcul importants aussi bien pour l'évaluation en une passe que pour l'évaluation en N passes (voir figure III.6). Cela s'explique par une réduction substantielle des temps d'évaluation du cas non incrémental au cas incrémental. Cette réduction s'explique par un élagage très important. On peut noter en effet que les pourcentages de candidats élagués tournent autour de 90% (voir figure III.7). De même, on peut noter que les gains augmentent d'une instance à l'autre proportionnellement à la cardinalité des domaines des attributs. Cela s'explique par la diminution du nombre de fréquents. On remarque par exemple (voir figure III.7) que le nombre de fréquents passe de 16525 pour l'instance I-10-5 à 4415 pour l'instance I-40-5.
Pour le deuxième groupe (TM-3, TM-5), on peut faire sensiblement le même raisonnement. Les

Temps total d'élagage (s)	4	1	7	1	1
Temps d'élagage par candidat (μs)	32,29	13,33	24,98	13,72	11,79
Nb de candidats avant élagage	123874	75022	280244	72878	84851
Nb de candidats après élagage	16425	4015	4567	1517	6933
Pourcentage de candidats restants	13,3%	5,4%	1,6%	2,1%	8,2%
Pourcentage de candidats élagués	86,7%	94,6%	98,4%	97,9%	91,8%
Taille frontière positive	7568	662	559	690	589
Nombre de candidats fréquents	16525	4415	5351	1917	7333

FIG. III.7 – Tests pour la sélection : Elagage des candidats pour le premier et le deuxième groupe.

	I10-10-5	I10-1-3
Taille moyenne des motifs	4	4
Cardinalité des domaines des attributs	10	10
Nombre de motifs pour la génération	100	100
Cardinalité de Cust	2000	2000
Cardinalité de Prod	6000	6000
Cardinalité de Sales	54000	54000
Seuil minimal de support	5%	3%

FIG. III.8 – Instances du quatrième groupe (I10-1-5, I10-1-3) : variation du seuil de support pour une même instance (I10-1-5, I10-1-3).

Temps d'évaluation non incrémental (s)	249	338
Temps total non incrémental	282	609
Temps d'évaluation incrémental en N passes (s)	124	195
Temps d'évaluation incrémental en 1 passe (s)	86	180
Temps total d'élagage (s)	4	17
Temps total incrémental en N passes (s)	224	616
Temps total incrémental en 1 passe (s)	193	601
Gain en temps en N passes	45,0%	17,7%
Gain en temps en 1 passe	56,0%	20,2%

FIG. III.9 – Tests pour la sélection : Gains en temps de calcul pour le quatrième groupe.

Temps total d'élagage (s)	4	17
Temps d'élagage par candidat (μs)	32,29	68,7
Nb de candidats avant élagage	123874	249009
Nb de candidats après élagage	16425	55451
Pourcentage de candidats restants	13,3%	22,3%
Pourcentage de candidats élagués	86.7%	77,7%
Taille frontière positive	7568	22230
Nombre de candidats fréquents	16525	55451

FIG. III.10 – Tests pour la sélection : Elagage des candidats pour le quatrième groupe.

Temps total non incrémental	282	189	599	197	436
Temps total incrémental (jointure) (s)	259	133	400	138	298
Temps d'élagage (s)	3	2	29	1	1
Gain en temps	8,2%	29,6%	33,2%	29,9%	31,7%

FIG. III.11 – Tests pour la jointure : Gains en temps de calcul pour le premier et le deuxième groupe.

Temps d'élagage (s)	3	2	29	1	1
Temps d'élagage par candidat (μs)	24,2	26,52	103,19	13,65	12,77
Nb de candidats avant élagage	123974	75422	281028	73278	78318
Nb de candidats après élagage	107662	43943	158459	41662	46859
Pourcentage de candidats restants	86,8%	58,3%	56,4%	56,9%	59,8%
Pourcentage de candidats élagués	13,2%	41,7%	43,6%	43,1%	40,2%

FIG. III.12 – Tests pour la jointure : Elagage des candidats pour le premier et le deuxième groupe.

gains importants en temps de calcul s'expliquent par le pourcentage important de candidats élagués qui est ici respectivement de 97.9% et 91.8%. Ici, on remarque que les gains diminuent lorsque la taille moyenne des motifs augmente. En effet, plus le motif est de taille importante plus il y a de motifs fréquents, puisque tous ses sous-motifs sont fréquents. On remarque ici que le nombre de fréquents passe de 1917 pour TM3-1-5 à 7333 pour TM5-1-5.
Un raisonnement similaire peut être fait pour le groupe 3 (NM50-1-20, NM100-1-10 et NM150-1-6).
Finalement, si on compare les instances I-10-5 et I-10-3, on note une réduction sensible du gain en temps de calcul. Cela est normal puisqu'il y a une augmentation importante du nombre de fréquents, puisqu'il y a moins de cas d'élagage.

Jointure

Pour la jointure, on considère les contextes $\mathcal{C}_1$, $\mathcal{C}_2$ et $\mathcal{C}_{12} = \mathcal{C}_1 \bowtie \mathcal{C}_2$, définis comme suit :
$\mathcal{C}_1$:
- Requête de référence : $AllSales = \pi_{x_1}(Sales)$;
- Requête de base : $Cust \bowtie Sales$;
- $\Sigma = \{y_1 = *, x_2 = *, x_3 = *, x_4 = *\}$

$\mathcal{C}_2$:
- Requête de référence : $AllSales = \pi_{x_1}(Sales)$;
- Requête de base : $Sales \bowtie Prod$;
- $\Sigma = \{y_1 = *, y_2 = *, y_3 = *\}$

Pour la jointure, les tests sont effectués de la même façon que pour la sélection. Le tableau des différentes instances du premier et du deuxième groupe (voir figure III.5) et le tableau des instances du quatrième groupe (voir figure III.8) sont les mêmes que ceux pour la sélection . Le tableau des gains en temps pour les instances du premier et du deuxième groupe est présenté à la figure III.11, et celui des gains en temps pour les instances du quatrième groupe, à la figure III.13. Le tableau des paramètres de l'élagage pour le premier et le deuxième groupe et celui pour le quatrième groupe sont présentés respectivement à la figure III.12 et à la figure III.14.

Dans le cas de la jointure, nous observons des gains de temps par rapport au cas non in-

Temps total non incrémental	282	609
Temps total incrémental (jointure) (s)	259	598
Temps total d'élagage (s)	3	4
Gain en temps	8,2%	1,8%

FIG. III.13 – Tests pour la jointure : Gains en temps de calcul pour le quatrième groupe.

Temps total d'élagage (s)	3	4
Temps d'élagage par candidat (μs)	24,2	20,65
Nb de candidats avant élagage	123974	193658
Nb de candidats après élagage	107662	175213
Pourcentage de candidats restants	86,8%	90,5%
Pourcentage de candidats élagués	13,2%	9,5%

FIG. III.14 – Tests pour la jointure : Elagage des candidats pour le quatrième groupe.

crémental, et on peut raisonner de manière générale de la même façon que pour le cas de la sélection. Par exemple, pour l'instance I10, la diminution du seuil minimal de support de 5% à 3% entraîne une réduction du gain en temps de calcul de 8.2% à 1.8%, et l'on constate que le pourcentage de candidats élagués diminue aussi. Cependant, les gains obtenus dans le cas de la jointure sont moins importants que dans le cas de la sélection. On trouve par exemple des gains de 8.2%, 29.6% et 33.2% respectivement pour les instances I10-1-5, I40-1-5 et I80-1-5 alors qu'ils étaient de 56.0%, 81.0% et 89.8%. Cela s'explique par le fait qu'il y a moins d'élagage dans le cas de la jointure que dans celui de la sélection. Prenons comme exemple le candidat $AllSales \bowtie \sigma_S(Cust \bowtie Sales \bowtie Prod)$ avec $S = (y_1 = a, x_2 = b, y_3 = d)$. Dans la cas de la sélection, on peut directement savoir à partir de la frontière positive si ce candidat est intéressant ou pas et l'élaguer. Dans le cas de la jointure, ce candidat n'a encore jamais été rencontré. On peut juste vérifier à partir des frontières positives de $freq_\alpha(\mathcal{C}_1)$ et de $freq_\alpha(\mathcal{C}_2)$ que ses sous-motifs correspondants sont fréquents. Il se peut ainsi qu'il ne soit pas fréquent, mais il ne sera pas élagué.
Nous montrons à la figure III.15 l'évolution des gains en fonction de la cardinalité du domaine pour la sélection et pour la jointure.
La figure III.16 montre l'évolution des gains en temps de calcul en fonction de la taille moyenne des motifs.
La figure III.17 montre la variation des gains en temps de calcul en fonction du seuil minimal de support pour une même instance de la base de données.

4.4.3 Complexité de la phase d'élagage

La phase d'élagage est une phase que nous introduisons dans l'algorithme d'extraction par niveau. Il s'agit ainsi d'une phase supplémentaire. Il est alors naturel de se demander si le temps requis par cette phase ne va pas augmenter le temps total de calcul.

Nous allons donc étudier la complexité de l'élagage pour savoir dans quelles circonstances cette phase d'élagage est plus intéressante que dans d'autres.
Comme nous l'avons déjà vu, l'élagage nécessite la comparaison de chaque candidat généré avec toutes les requêtes de la frontière positive, et pour cela, nous utilisons une représentation inverse de cette dernière. Ainsi les comparaisons vont se faire par de simples intersections des listes des

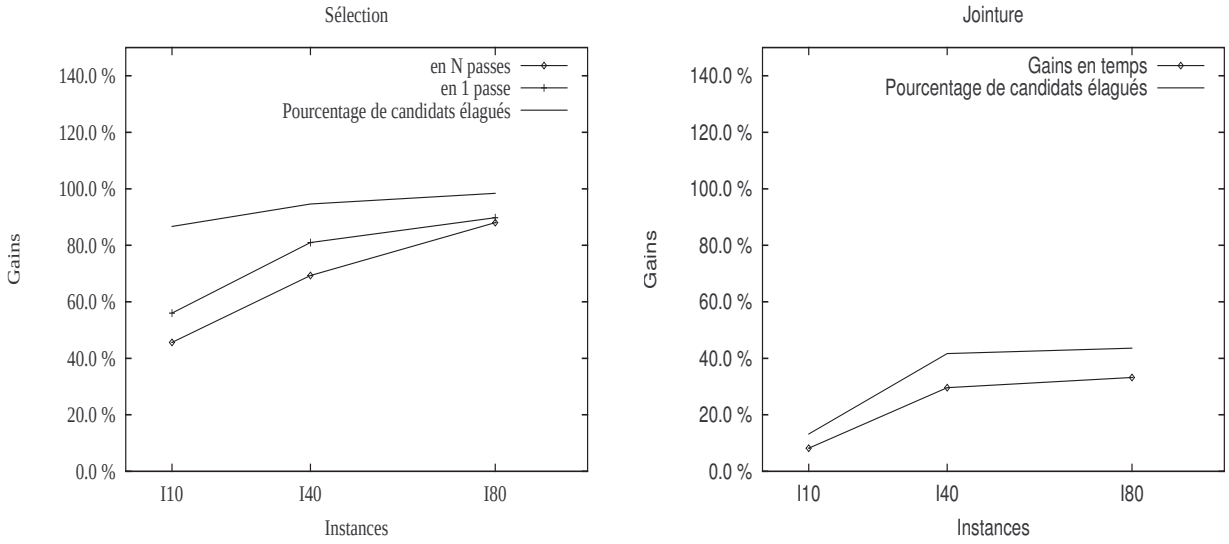


FIG. III.15 – Gains en temps de calcul en fonction de la cardinalité des domaines

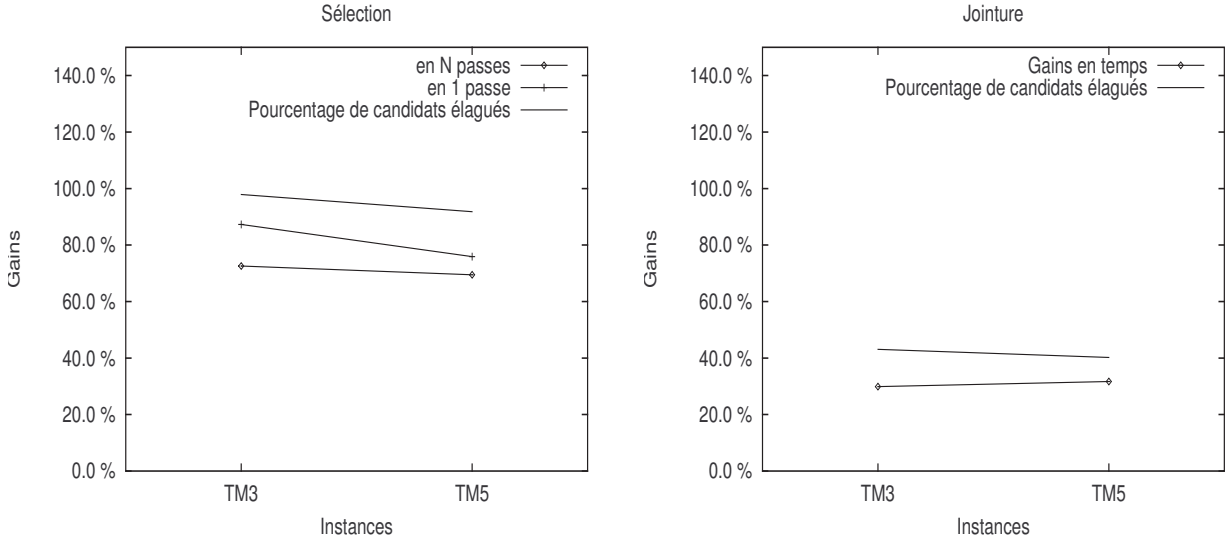


FIG. III.16 – Gains en temps de calcul en fonction de la taille moyenne des motifs

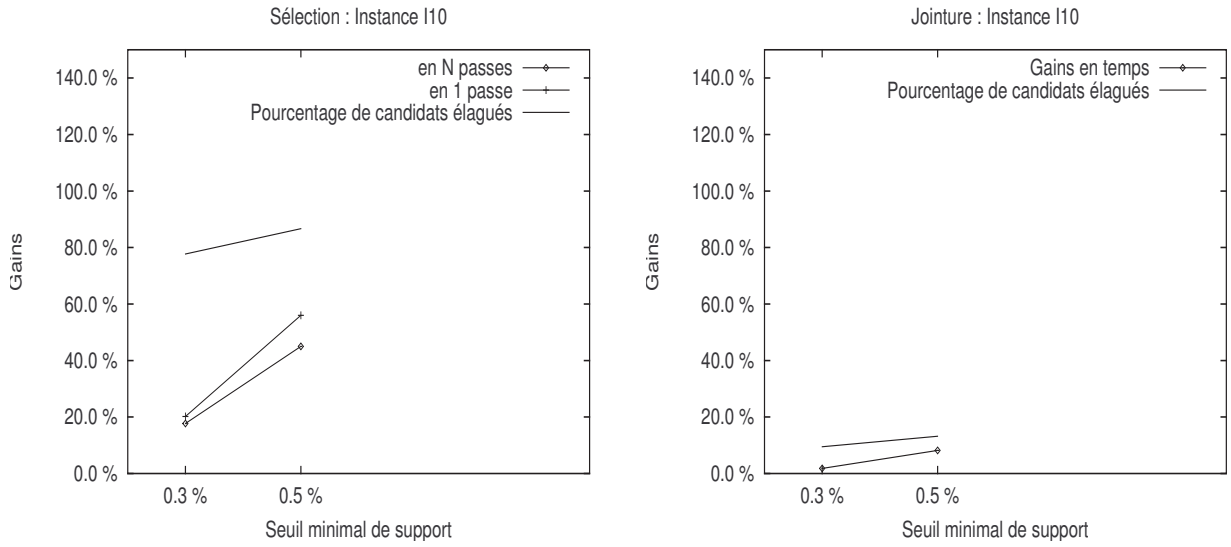


FIG. III.17 – Gains en temps de calcul en fonction du seuil minimal du support

identifiants de requêtes des frontières positives.
L'étude de la complexité revient alors à évaluer le coût des intersections des listes d'identifiants des frontières positives. Pour cela, un élément important est la taille de la liste. En effet, plus la liste est grande, plus l'intersection est longue à calculer. Nous allons donc calculer la taille moyenne d'une liste, et en fonction de cela, nous verrons quels sont les paramètres qui influent sur l'élagage.
Rappelons que la représentation inverse d'une frontière positive consiste à considérer pour chaque condition de sélection atomique S, l'ensemble des identifiants de requêtes qui contiennent S. C'est dire que le nombre de candidats au niveau 1 ($NCand(1)$)du treillis (candidats avec une condition de sélection atomique) correspond au nombre de listes de la représentation inverse.
D'autre part, étant donné que les listes sont constituées par les identifiants de requêtes, si on note $|Bd^+|$ la taille de la frontière positive et TM la taille moyenne d'une requête de la frontière positive, la somme des longueurs des listes est donnée par $Somme = |Bd^+| * TM$.
Ainsi, la taille moyenne d'une liste est égale à :

$$Taille\ moyenne = (|Bd^+| * TM)\ /\ NCand(1)$$

Etant donné qu'au niveau k, un candidat comporte k conditions de sélections, le temps d'élagage au niveau k d'un candidat est proportionnel à $k * Taille\ moyenne$.
On constate ainsi que différents paramètres influent sur le temps d'élagage théorique d'un candidat.
Nous allons calculer ce temps théorique d'élagage pour différentes instances, et comparer son évolution avec celle des temps réels que nous avons obtenus pour nos expériences (voir figure III.18). Le temps réel au niveau k pour un candidat est obtenu par le rapport $Te(k)/Ncand(k)$, où $Te(k)$ est le temps d'élagage au niveau k, et $NCand(k)$ le nombre de candidats au niveau k avant l'élagage.

Si on note T_3, T_1 et T_2 les temps théorique d'élagage pour respectivement I10-1-3, I10-1-5 et I80-1-5, on note que $T_3 > T_1 > T_2$. On peut noter qu'on a la même évolution pour les temps réels. Il faut souligner que le temps théorique que nous avons calculé est un nombre de comparaisons, alors que la temps réel est en unité de temps. Il ne s'agit donc pas des mêmes mesures. De plus

	Bd^+	$NCand(1)$	TM	Niveau k	$k * \frac{(\|Bd^+\| * TM)}{NCand(1)}$	$Te(k)$	$NCand(k)$	$\frac{Te(k)}{NCand(k)}$
I-10-5	7568	100	3	3	681,12	4	111543	35,86
I80-1-5	559	784	2	2	2,85	7	276572	25,3
I10-1-3	22230	100	2,6	3	1733,94	17	119521	142,23

FIG. III.18 – Comparaison de l'évolution du temps théorique d'élagage et du temps réel

les temps théoriques correspondent à des estimations dans le pire des cas. C'est ce qui explique la différence que nous avons entre le temps théorique et le temps réel.

4.5 Conclusion

Nous avons dans cette partie présenté l'implémentation de notre approche itérative par la composition de contextes d'extraction pour une extraction efficace des règles d'association. Cette approche utilise les frontières positives des extractions antérieures pour optimiser le calcul d'une nouvelle extraction. Cette optimisation se caractérise principalement par une représentation inverse de la frontière positive qui permet d'effectuer de manière efficace les tests de comparaison entre une nouvelle requête candidate et les requêtes de cette frontière.

Dans les tests que nous avons effectués, nous avons utilisé deux caches : un cache sur disque qui permet de stocker la frontière positive sous forme d'une base de données ($DMining$), et un cache en mémoire pour stocker la représentation inverse.
La gestion du cache disque ne pose pas de problèmes puisqu'on peut supposer qu'on a toujours assez d'espace disque, mais une question que l'on pourrait se poser est de savoir comment faire lorsque le cache mémoire est plein. Dans ce cas, il faudrait implémenter une stratégie de gestion du cache. On pourrait alors transférer sur disque les frontières positives des réponses des contextes qui sont peu utilisés et garder en mémoire celles des contextes plus utilisés.

Il faut signaler que la poursuite des tests du développement de cette maquette a été interrompue, car il nous semblait nécessaire d'effectuer auparavant une étude plus approfondie du stockage des représentations condensées pour éviter les redondances. Ce point fait l'objet de la partie suivante.

5 Représentations condensées d'un ensemble de réponses à des requêtes d'extraction

Dans cette partie, nous présentons les derniers résultats que nous avons obtenus sur les représentations condensées d'ensembles de réponses à des requêtes d'extraction. Cette présentation se fera sous forme de résumé, le formalisme détaillé de notre approche étant présenté dans un article joint en annexe à ce mémoire.

La découverte de motifs fréquents dans les bases de données est un sujet qui a fait l'objet de plusieurs études dans la communauté scientifique. Même si la plupart des recherches sont orientées sur l'optimisation des algorithmes d'extraction, un autre problème important est le stockage efficace de la réponse à une requête d'extraction, surtout lorsque l'on veut réutiliser ce résultat pour une future extraction. Nous avons vu dans la partie précédente qu'un problème crucial qui se pose à l'extraction itérative de requêtes par la composition de contextes est celui de la redondance dans le stockage des frontières positives. Cette redondance est due au fait que les différentes représentations condensées sont stockées indépendamment les unes des autres.

Nous proposons dans cette partie des représentations condensées pour un ensemble de réponses à des requêtes d'extraction.
Supposons que nous avons :

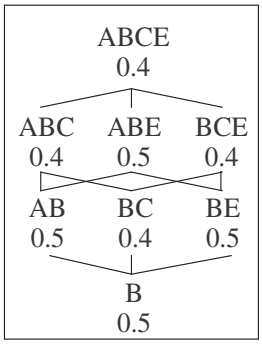
1. Un ensemble $\mathbf{\Delta}$ de tous les ensembles de données à partir desquels les motifs sont découverts. Par exemple, $\mathbf{\Delta}$ peut être vu comme l'ensemble de toutes les instances d'un schéma d'une relation donnée.
2. Un ensemble de motifs $\mathbb{L}$ et un ordre partiel $\preceq$ sur $\mathbb{L}$. Etant donné deux motifs φ_1, φ_2 dans $\mathbb{L}$, on dit que φ_1 est plus spécifique que φ_2 (ou que φ_2 est plus général que φ_1) si on a $\varphi_1 \preceq \varphi_2$.
3. Un ensemble $\mathbb{Q}$ de critères de sélection, un critère de sélection $q \in \mathbb{Q}$ étant une fonction booléenne définie sur $\mathbb{L} \times \mathbf{\Delta}$. De plus, étant donné un motif φ dans $\mathbb{L}$ et un ensemble de données Δ dans $\mathbf{\Delta}$, on dit que φ est intéressant dans Δ par rapport à q si $q(\varphi, \Delta) = true$.
   Un critère de sélection q est anti-monotone si pour tout ensemble de données Δ dans $\mathbf{\Delta}$, on a : pour tous φ_1 et φ_2 dans $\mathbb{L}$, si $q(\varphi_1, \Delta) = true$ et φ_2 est plus général que φ_1, alors $q(\varphi_2, \Delta) = true$.
   Un critère de sélection q est monotone si pour tout ensemble de données Δ dans $\mathbf{\Delta}$, on a : pour tous φ_1 et φ_2 dans $\mathbb{L}$, si $q(\varphi_2, \Delta) = true$ et φ_1 est plus spécifique que φ_2, alors $q(\varphi_1, \Delta) = true$.
4. Un ensemble $\mathbb{F}$ de mesures, une mesure étant une fonction définie de $\mathbb{L} \times \mathbf{\Delta}$ dans $\Re$.

Nous appelons *requête d'extraction simple* un critère de sélection q, et l'ensemble des motifs intéressants dans Δ par rapport à q, noté $sol(q/\Delta)$, est la *réponse de q dans Δ*.

Nous appelons *requête d'extraction étendue* tout couple de la forme (q, f), où $q \in \mathbb{Q}$ et $f \in \mathbb{F}$. *La réponse dans* Δ à une requête d'extraction étendue (q, f), notée $ans(q, f/\Delta)$, est l'ensemble des couples $(\varphi, f(\varphi, \Delta))$ tels que φ est dans $sol(q/\Delta)$.

Par la suite, pour illustrer ces différentes notions et montrer les différentes représentations condensées que nous proposons, nous allons utiliser un exemple dans le cas classique de l'extraction des règles d'associations [2] .

Exemple 5.1 *Etant donné un ensemble d'items Items, l'ensemble des motifs $\mathbb{L}$ considéré dans notre approche est l'ensemble de tous les sous-ensembles de Items, i.e. $\mathbb{L} = 2^{Items}$. De plus, l'ordre partiel que nous considérons sur $\mathbb{L}$ est la relation d'inclusion : étant donné deux motifs φ et φ' dans $\mathbb{L}$, on dit que $\varphi \preceq \varphi'$ si $\varphi' \subseteq \varphi$.*



Tr	Set of Items
1	A
2	DE
3	ABCE
4	ABE
5	ABCDE
6	ACD
7	ABCE
8	AE
9	ABCDE
10	CD

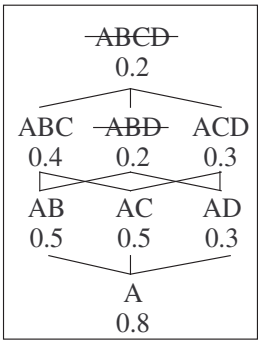


FIG. III.19 – Example of data set and sub-lattices of interesting patterns.

Dans ce contexte, un ensemble de données Δ *est défini par un ensemble de transactions* Tr *et une fonction* it *de* Tr *vers* $\mathbb{L}$*. Etant donné une transaction* $x \in Tr$*,* $it(x)$ *est l'ensemble des items dans la transaction* x*. Le support d'un motif est un exemple de mesure de* $\mathbb{F}$*. Plus précisément, pour tout motif* φ*, le support de* φ *dans* Δ*, noté* $sup(\varphi, \Delta)$*, est défini par :*

$$sup(\varphi, \Delta) = |\{x \in Tr \mid it(x) \preceq \varphi\}|/|Tr|.$$

Notons que, étant donné un seuil minimal de support $minsup$*, nous pouvons considérer le critère de sélection* q *défini par : pour tout motif* $\varphi \in \mathbb{L}$*,* $q(\varphi, \Delta) = true$ *si* $sup(\varphi, \Delta) \geq minsup$*.*

Par la suite, nous considérons le cas où l'ensemble des items est $Items = \{A, B, C, D, E\}$ *et où l'ensemble des transactions est* $Tr = \{1, 2, \ldots, 10\}$*. Nous noterons tout ensemble d'items par la concaténation de ses éléments. Par exemple l'ensemble d'items* $\{A, B, C\}$ *est noté* ABC*. Une fonction* it *de* Tr *dans* $\mathbb{L}$ *qui définit un ensemble de données* Δ *est représentée dans la table de la figure III.19.*

Soit m_1*,* m_2*,* a_1 *et* a_2 *des critères de sélection définis pour tout motif* φ *dans* $\mathbb{L}$ *et pour tout ensemble de données* $\Delta \in \mathbf{\Delta}$ *par :*

- $m_1(\varphi, \Delta) = true$ *si* $B \subseteq \varphi$*,* $m_2(\varphi, \Delta) = true$ *si* $A \subseteq \varphi$*.*
- $a_1(\varphi, \Delta) = \overline{a_1}(\varphi, \Delta) \wedge \widetilde{a_1}(\varphi, \Delta)$*, où* $\overline{a_1}(\varphi, \Delta) = true$ *si* $sup(\varphi, \Delta) \geq 0.4$*, et* $\widetilde{a_1}(\varphi, \Delta) = true$ *si* $\varphi \subseteq ABCE$*.*
- $a_2(\varphi, \Delta) = \overline{a_2}(\varphi, \Delta) \wedge \widetilde{a_2}(\varphi, \Delta)$*, où* $\overline{a_2}(\varphi, \Delta) = true$ *si* $sup(\varphi, \Delta) \geq 0.3$*, et* $\widetilde{a_2}(\varphi, \Delta) = true$ *si* $\varphi \subseteq ABCD$*.*

$q_1 = m_1 \wedge a_1$ *et* $q_2 = m_2 \wedge a_2$ *sont des requêtes d'extraction simples. De plus, on peut montrer facilement à partir de la table de la figure III.19 que :*

- $sol(m_1 \wedge a_1/\Delta) = \{B, AB, BC, BE, ABC, ABE, BCE, ABCE\}$
- $sol(m_2 \wedge a_2/\Delta) = \{A, AB, AC, AD, ABC, ACD\}$

D'autre part, (q_1, sup) *et* (q_2, sup) *sont des exemples de requêtes d'extraction étendues, et on a :*

- $ans(q_1, sup/\Delta) = \{(B, 0.5), (AB, 0.5), (BC, 0.4), (BE, 0.5), (ABC, 0.4), (ABE, 0.5), (BCE, 0.4), (ABCE, 0.4)\}$.
- $ans(q_2, sup/\Delta) = \{(A, 0.5), (AB, 0.5), (AC, 0.5), (AD, 0.3), (ABC, 0.4), (ACD, 0.3)\}$.

Les réponses dans Δ *de* (q_1, sup) *et* (q_2, sup) *sont également représentées figure III.19.*

Par la suite, nous supposons que l'ensemble de données Δ est fixe. Ainsi, Δ sera omis dans les notations qui vont suivre. Par exemple, $sol(q/\Delta)$ sera noté simplement $sol(q)$.
Soit une requête d'extraction simple q définie par une conjonction de critères de sélection monotone et anti-monotone. Nous rappelons que dans ce cas, $sol(q)$ peut être représenté par l'ensemble

$S(q)$ de ses motifs les plus spécifiques (par rapport à l'ordre partiel sur $\mathbb{L}$) et l'ensemble $G(q)$ de ses motifs les plus généraux [47, 63]. En effet, $sol(q)$ peut être regénéré à partir de l'ensemble $\{S(q), G(q)\}$ sans accéder aux données. C'est pourquoi $\{S(q), G(q)\}$ est appelé une représentation condensée de $sol(q)$. Par exemple la requête d'extraction simple $q_1 = m_1 \wedge a_1$ définie dans l'exemple 5.1 est la conjonction d'un critère de sélection monotone m_1 et d'un critère de sélection anti-monotone a_1. $\{S(q_1), G(q_1)\}$ est une représentation condensée de $sol(q_1)$, où $S(q_1) = \{ABCE\}$ et $G(q_1) = \{B\}$, car il est facile de voir que l'on a :

$$sol(q_1) = \{\varphi \mid (\exists \varphi_1 \in S(q_1))(\exists \varphi_2 \in G(q_1))(\varphi_1 \preceq \varphi \preceq \varphi_2)\}.$$

Pour les requêtes d'extraction étendues, nous redéfinissons dans notre formalisme l'ensemble des fermés et l'ensemble des clés introduits dans [6, 16, 75], que nous notons respectivement $SC(q, f)$ et $GK(q, f)$. Plus précisément, étant donné la relation d'ordre partiel $\leq_f$ définie pour tout couple de motifs (φ, φ') par :

$$\varphi \leq_f \varphi' \text{ si } \varphi \preceq \varphi' \text{ et } f(\varphi) = f(\varphi'),$$

on note $SC(q, f) = min_{\leq_f}(sol(q))$ et $GK(q, f) = max_{\leq_f}(sol(q))$.
Nous montrons alors que $ans(q, f)$ peut être régénéré à partir de $\{S(q), G(q), \lambda_{SC}[q, f]\}$, où $\lambda_{SC}[q, f] = \{(\varphi, f(\varphi)) \mid \varphi \in SC(q, f)\}$.

Considérons maintenant le cas d'ensembles de requêtes d'extraction (simples ou étendues). En notant que l'union des représentations condensées de différentes requêtes d'extraction n'est pas une représentation condensée de l'ensemble correspondant de requêtes d'extraction [47], nous généralisons les notions de maximal, de clés et de fermés au cas d'ensembles de requêtes d'extraction. De plus, nous proposons des représentations condensées pour de tels ensembles définies comme suit : étant donné un ensemble $\mathcal{Q}$ de requêtes d'extraction, R est une représentation condensée de l'ensemble des réponses aux requêtes de $\mathcal{Q}$ si les réponses aux requêtes de $\mathcal{Q}$ dans Δ peuvent être calculées en se basant seulement sur la représentation condensée R, i.e., sans accéder aux données.

Exemple 5.2 *Dans le contexte de l'exemple 5.1, soit $q_3 = m_3 \wedge a_3$ et $q_4 = m_4 \wedge a_4$, où m_3, m_4, a_3 et a_4 sont des critères de sélection définis pour tout motif $\varphi \in \mathbb{L}$ par :*
- *$m_3(\varphi, \Delta) = true$ si $A \subseteq \varphi$, et $m_4(\varphi, \Delta) = true$ si $AC \subseteq \varphi$,*
- *$a_3(\varphi, \Delta) = true$ si $sup(\varphi, \Delta) \geq 0.3$ et $\varphi \subseteq ABC$, et $a_4(\varphi, \Delta) = true$ si $sup(\varphi, \Delta) \geq 0.3$ et $\varphi \subseteq ABCD$.*

On a : $S(q_3) = \{ABC\}$, $S(q_4) = \{ABC, ACD\}$, $G(q_3) = \{A\}$ et $G(q_4) = \{AC\}$.
Considérons $\mathcal{Q} = \{q_3, q_4\}$, nous montrons que : $\{\lambda_S[\mathcal{Q}], \lambda_G[\mathcal{Q}]\}$ est une représentation condensée de $sol(\mathcal{Q})$, où :

$$\lambda_S[\mathcal{Q}] = \{(ABC, \{q_3, q_4\}), (ACD, \{q_4\})\} \text{ et}$$
$$\lambda_G[\mathcal{Q}] = \{(A, \{q_3\}), (AC, \{q_4\}).\}$$

On notera qu'en plus des motifs φ des ensembles $S(q_i)$ et $G(q_i)$, $i = 1, 2$, on garde dans la représentation condensée l'ensemble des requêtes $\{q \in \mathcal{Q} \mid \varphi \in sol(q)\}$ pour chaque φ appartenant à $S(q)$ ou $G(q)$. On pourrait dire qu'on ne condense pas davantage puisque tous les motifs de $S(q_i)$ et $G(q_i)$ sont stockés, mais par contre, il n'y a plus de redondance, dans le sens où on ne stocke pas plusieurs fois le même motif.
Dans le cas des requêtes d'extraction étendues, on arrive à condenser davantage les ensembles de fermés ou clés, parce qu'il y a des motifs de $SC(q_i, f)$ ou $GK(q_i, f)$ qu'on ne stocke plus dans la représentation condensée. Cela est dû au fait que dans la représentation condensée, ne sont gardés que les minimaux (par rapport à la relation $\leq_f$) de l'union des $SC(q_i, f)$, notée $\mathcal{SC}(\mathcal{Q}, f)$, ou les maximaux de l'union des $GK(q_i, f)$, notée $\mathcal{GK}(\mathcal{Q}, f)$.

Exemple 5.3 *Soit $\mathcal{Q} = \{q_1, q_2\}$ l'ensemble des requêtes d'extraction simples définies dans l'exemple 5.1. A partir des ensembles suivants :*
- *$S(q_1) = \{ABCE\}$ et $S(q_2) = \{ABC, ACD\}$, $G(q_1) = \{B\}$ et $G(q_2) = \{A\}$,*
- *$SC(q_1, f) = \{ABCE, ABE\}$ et $SC(q_2, f) = \{ABC, ACD, AB, AC, A\}$,*

nous montrons que $\{\lambda_S[\mathcal{Q}],\ \lambda_G[\mathcal{Q}],\ \lambda_{SC}[\mathcal{Q}, f]\}$ est une représentation condensée de $ans(\mathcal{Q}, f)$, où :
- *$\lambda_S[\mathcal{Q}] = \{(ABCE, \{q_1\}), (ABC, \{q_2\}), (ACD, \{q_2\})\}$,*
- *$\lambda_G[\mathcal{Q}] = \{(B, \{q_1\}), (A, \{q_2\})\}$,*
- *$\mathcal{SC}(\mathcal{Q}, f) = \{ABCE, ABE, ACD, AC, A\}$, et*
- *$\lambda_{SC}[\mathcal{Q}, f] = \{(ABCE, 0.4), (ABE, 0.5), (ACD, 0.3), (AC, 0.5), (A, 0.8)\}$.*

On notera que les motifs ABC et AB de $SC(q_2, f)$ n'ont pas été gardés. En effet $ABC \subseteq ABCE$ et $sup(ABC) = sup(ABCE)$, ce qui montre que $ABC \leq_{sup} ABCE$. De même, puisque $AB \subseteq ABE$ et $sup(AB) = sup(ABE)$, on a $AB \leq_{sup} ABE$.

Considérons maintenant l'ensemble des requêtes d'extraction de la forme $q = q_{ind} \wedge a \wedge m$, où :
- q_{ind} ne dépend pas de l'ensemble de données,
- a est de la forme $a(\varphi) = true$ si $f(\varphi) \geq \alpha$ (a est antimonotone),
- m est de la forme $m(\varphi) = true$ si $f(\varphi) \leq \beta$ (m est monotone).

On peut montrer que l'on peut trouver une représentation de $sol(\mathcal{Q})$ plus condensée que $\{\lambda_S[\mathcal{Q}],\ \lambda_G[\mathcal{Q}]\}$.

Exemple 5.4 *Considérons de nouveau l'ensemble $\mathcal{Q} = \{q_1, q_2\}$ défini dans l'exemple 5.1. Nous avons vu que $\lambda_S[\mathcal{Q}] = \{(ABCE, \{q_1\}), (ABC, \{q_2\}), (ACD, \{q_2\})\}$ et $\lambda_G[\mathcal{Q}] = \{(B, \{q_1\}), (A, \{q_2\})\}$. On peut montrer que $(ABC, \{q_2\})$ peut être supprimée de la représentation condensée $\{\lambda_S[\mathcal{Q}],\ \lambda_G[\mathcal{Q}]\}$ parce que, d'une part $ABC \subseteq ABCE$ et d'autre part, pour tout φ, $sup(\varphi) \geq 0.4$ implique $sup(\varphi) \geq 0.3$.*
Ainsi les ensembles $\overline{\lambda}_S[\mathcal{Q}] = \{(ABCE, q_1), (ACD, q_2)\}$, et $\overline{\lambda}_G[\mathcal{Q}] = \{(A, \{q_2\})\}$ constituent une représentation condensée de $sol(\mathcal{Q})$ plus concise que $\{\lambda_S[\mathcal{Q}],\ \lambda_G[\mathcal{Q}]\}$.

Nous montrons de la même façon que $\{\overline{\lambda}_S[\mathcal{Q}], \overline{\lambda}_G[\mathcal{Q}], \lambda_{SC}[\mathcal{Q}, f]\}$ est une représentation condensée de $ans(\mathcal{Q}, f)$ plus concise que $\{\lambda_S[\mathcal{Q}],\ \lambda_G[\mathcal{Q}],\ \lambda_{SC}[\mathcal{Q}, f]\}$.

De plus, dans le papier présenté en Annexe :
- nous proposons pour toute requête d'extraction $q \in \mathcal{Q}$ un algorithme de régénération de la réponse de (q, f) à partir de la représentation condensée étendue $\{\overline{\lambda}_S[\mathcal{Q}], \overline{\lambda}_G[\mathcal{Q}], \lambda_{SC}[\mathcal{Q}, f]\}$ de $ans(\mathcal{Q}, f)$ sans accéder aux données.
- Nous montrons ensuite comment les représentations condensées étendues peuvent être utlisées pour optimiser le calcul itératif de la réponse à une nouvelle requête d'extraction.
- Finalement, nous montrons comment modifier la représentation condensée de la réponse de $\mathcal{Q}$ pour obtenir une représentation condensée de la réponse de $\mathcal{Q} \cup \{q\}$, où q est une requête n'appartenant pas à $\mathcal{Q}$.

Chapitre IV

Conclusion et perspectives

La découverte de connaissances dans une base de données est un processus interactif et itératif, où l'utilisateur, ne sachant pas a priori ce qu'il cherche, pose un certain nombre de requêtes d'extraction pour trouver ce qui l'intéresse. Un des principaux problèmes qui se posent pour l'exécution de ces requêtes d'extraction est leur temps important de calcul. Nous avons également évoqué plusieurs problèmes parmi lesquels celui du stockage efficace de la réponse à une requête d'extraction, surtout lorsque l'on veut réutiliser cette réponse pour une optimisation future, et celui de l'explosion de l'espace de recherche. Il apparaît ainsi plusieurs besoins fondamentaux : le besoin de définition de l'espace de recherche à parcourir, le besoin de langages de requêtes d'extraction et le besoin de réutilisation des résultats déjà calculés pour une optimisation des calculs futurs.

1 Résumé de nos contributions

Dans un premier temps, nous avons proposé dans la partie 2 du chapitre III de ce mémoire un formalisme pour l'extraction itérative de requêtes conjonctives à partir de vues. Cette approche, qui est basée sur un formalisme logique pour la représentation des données et des motifs (qui sont ici des requêtes et des implications entre requêtes), définit l'espace de recherche à parcourir en précisant le type de motifs auxquels on s'intéresse par l'introduction d'un contexte d'extraction. L'approche utilise les représentations condensées des réponses aux requêtes d'extraction déjà calculées pour optimiser le calcul des futures réponses, et se distingue notamment par l'introduction des vues dans l'expression des motifs. Ces vues permettent d'exprimer simplement des requêtes complexes.
Avec cette approche, le principal problème est que, pour toute requête candidate, nous devons tester son inclusion dans toutes les requêtes de la frontière négative de tous les contextes d'extraction comparables déjà calculés. Ce test d'inclusion entre les requêtes se fait par la recherche d'homomorphisme. Ceci est très coûteux, d'autant plus que la recherche d'un homomorphisme entre requêtes est un problème NP-complet.

C'est ainsi que nous avons proposé dans la partie 3 du chapitre III une autre approche pour l'extraction itérative de règles d'association par la composition de contextes d'extraction en utilisant les opérations de l'algèbre relationnelle. Dans cette approche, nous introduisons un langage de manipulations de contextes qui établit un lien entre les requêtes d'extraction, et par conséquent, élimine les problèmes d'efficacité liés aux tests d'inclusion. Ce langage de manipulation de contextes donne à l'utilisateur une plus grande marge de manoeuvre dans la spécification de sa requête d'extraction, et permet de réduire le temps de calcul des requêtes fréquentes des contextes composés en utilisant les requêtes fréquentes stockées des contextes composants.
Nous avons vu qu'il suffisait de stocker et d'utiliser la frontière positive des ensembles de requêtes fréquentes pour optimiser le calcul des requêtes fréquentes lorsque de nouveaux contextes sont définis par composition d'anciens. Cette optimisation étant basée sur la comparaison des requêtes candidates avec toutes les requêtes de la frontière positive, nous avons enfin proposé une méthode efficace permettant d'effectuer cette comparaison, en utilisant une représentation inverse de la frontière positive.
Cependant, un des principaux problèmes que pose cette approche est la redondance dans le stockage des réponses aux requêtes d'extraction, étant donné que ces dernières ne sont pas indépendantes.

Pour résoudre ce problème, nous avons introduit dans la partie 5 du chapitre III la notion de représentation condensée d'un ensemble de réponses à des requêtes d'extraction. Nous avons adapté les notions de représentations condensées que nous avons vues dans la partie 3 du chapitre

II sur l'état de l'art, notamment les maximaux, les fermés et les clés au cas d'ensembles de requêtes d'extraction définies par des critères de sélection monotones et anti-monotones. Nous avons ensuite proposé différentes représentations condensées de la réponse à des ensembles de requêtes.
Ensuite, nous avons montré dans un cas particulier comment les représentations condensées d'un ensemble $\mathcal{Q}$ de requêtes peuvent être utilisées :

- pour optimiser le calcul de la réponse à une nouvelle requête d'extraction,
- pour modifier efficacement la représentation condensée de la réponse de $\mathcal{Q}$ afin d'obtenir une représentation condensée de la réponse de $\mathcal{Q} \cup \{q\}$, où q est une requête n'appartenant pas à $\mathcal{Q}$.

2 Perspectives

Nous allons d'abord parler de perspectives concernant la partie algébrique, i.e. l'approche pour l'extraction itérative de règles d'association par la composition de contextes d'extraction en utilisant les opérations de l'algèbre relationnelle.
Plusieurs directions de recherche peuvent être considérées :

- Notre approche itérative peut être généralisée comme suit : même si le nouveau contexte n'est pas exprimé comme une composition d'autres contextes, on peut chercher une relation entre ce nouveau contexte et les contextes stockés. Dans le cadre des bases de données, il a été montré dans certains cas qu'une requête pouvait être exprimée à partir de vues prédéfinies. Nous pensons que l'on pourrait faire la même chose avec les contextes.
- Étant donné un ensemble de contextes X, on peut chercher un sous-ensemble de X tel que le stockage des requêtes fréquentes correspondantes puisse optimiser le temps de calcul des requêtes fréquentes de tout l'ensemble X.
- Nous avons jusque-là considéré la base de données comme statique, ce qui n'est pas le cas en pratique. Ainsi, les mises à jour devraient être prises en compte. Ce problème est similaire au problème de la maintenance des règles d'associations étudié dans [31, 86].

Les données sur lesquelles nous avons travaillé sont des données synthétiques. Nous avons présenté dans la partie consacrée à l'implémentation notre générateur de données. Il faudrait, bien entendu, faire plus de tests pour avoir plus de résultats, mais aussi il serait intéressant de faire des tests sur des données réelles.

Au vu de notre implémentation, le stockage de la frontière positive dans une base de données ne semble pas être un bon choix. Ce choix a été fait notamment pour exécuter des requêtes SQL sur la base de données afin de récupérer la représentation inverse. Seulement, nous nous sommes rendus compte que le temps de chargement du cache disque en mémoire centrale est très élevé. Il faudrait stocker la représentation inverse dans un fichier, de sorte à pouvoir la récupérer assez rapidement pour le calcul itératif. Cette remarque devra être prise en compte dans la nouvelle méthode de stockage, de même que les résultats sur les représentations condensées d'ensembles de requêtes. En effet, nous avons vu que les différentes réponses aux requêtes d'extraction étaient stockées indépendamment les unes des autres, alors qu'il y a un lien entre elles. Cette dépendance entre les différentes réponses, comme le montrent les représentations d'ensembles de requêtes, devra être prise en compte dans la nouvelle méthode de stockage, ce qui va permettre d'éliminer la redondance.

Dans cette même partie, nous avons introduit un langage de manipulation de contextes que l'on peut appeler un langage de requêtes, mais nous n'en avons pas étudié les propriétés. Cette

étude pourrait permettre de découvrir des propriétés (par exemple de commutation entre la sélection et la jointure entre contextes) en vue de l'optimisation du calcul des requêtes d'extraction, surtout lorsque celles-ci sont complexes. En SQL, lorsqu'une requête comporte plusieurs opérations (sélection, jointure, projection), on peut permuter les opérations pour une optimisation de la requête. On pourrait penser à faire la même chose lorsque l'on combine des contextes entre eux, ce qui pourrait conduire à une optimisation du calcul de la requête correspondant au contexte combiné.

Différentes perspectives se dégagent dans la partie sur les représentations condensées d'ensembles de requêtes.
Dans un premier temps, des tests sont nécessaires pour évaluer les différentes représentations condensées proposées. Nous sommes en train de faire des tests d'abord dans le cadre classique et ensuite dans le cadre approche algébrique pour étudier les gains de notre approche itérative.

En se basant sur nos travaux sur les représentations condensées d'ensembles de requêtes et sur les approches proposées par [76, 77], nous sommes entrain d'étudier comment intégrer les requêtes disjonctives dans notre formalisme. Dans ce sens, nous pouvons noter que, étant donné un ensemble $\mathcal{Q} = \{q_1, ..., q_n\}$ de requêtes d'extraction q_i $(i = 1, ..., n)$, on a : $sol(\mathcal{Q}) = sol(q_1 \vee ... \vee q_n)$. De plus, dans notre approche, la réponse de $sol(q_1 \vee ... \vee q_n)$ peut être calculée itérativement, en calculant à chaque phase k la réponse de q_k en utilisant les représentations condensées de $sol(\{q_1, ..., q_{k-1}\})$ $(k = 2, ..., n)$.

Contrairement à l'approche algébrique, l'approche que nous avons présentée sur les représentations condensées d'ensembles de requêtes ne dispose pas d'un langage de manipulations de requêtes. Nous avons une séquence de requêtes sans lien explicite entre elles. Il est alors intéressant de faire une étude sur l'intégration de cette approche avec celle sur la composition de contextes où nous avons un langage de requêtes.

Bibliographie

[1] H. Adé, L. De Raedt, and M. Bruynooghe. Declarative bias for specific-to-general ilp systems. In *Machine Learning*, 1995.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In AAAI-MIT Press, editor, *In Advances in Knowledge Discovery and Data Mining*, pages 309–328, 1996.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *International conference on Data Engineering*, Taipei, Taiwan, March 1995.

[4] A.J.Knobbe, H. Blockeel, A.P.J.M. Siebes, and D.M.G. Van der Wallen. Multi-relational data mining. Technical report, INS-R9908, ISSN, 1999.

[5] E. Baralis and G. Psaila. Incremental refinement of mining queries. In *DAWAK'99*, pages 173–182, Florence, 1999.

[6] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L Lakhal. Levelwise search of frequent patterns with counting inference. In *16èmes journées de bases de données avancées*, 2000.

[7] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L Lakhal. Pascal : un algorithme d'extraction des motifs fréquents. *Techniques et Sciences Informatiques*, 21 :65–95, 2002.

[8] Yves Bastide, Rafik Taouil, Nicolas Pasquier, Gerd Stumme, and Lotfi Lakhal. Mining frequent patterns with counting inference. *SIGKDD Explorations*, 2(2) :66–75, 2000.

[9] R. J. Bayardo. Efficiently mining long patterns from databases. In *SIGMOD conf*, pages 85–93, June 1998.

[10] K.S. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *ACM SIGMOD*, pages 359–370, Philadelphia, Pennysylvania, USA, 1999.

[11] M. Botta, J.F. Boulicault, C. Masson, and R. Meo. Query languages supporting descriptive rule mining : a comparative study. Technical report, LIRIS-INSA, Lyon, Université de turin, November 2002.

[12] "M. Botta, R. Meo, and M. L. Sapino". "incremental execution of the mine rule operator". Technical Report RT66-2002, Université de Turin, Turin, May 2002.

[13] J. F. Boulicault, M. Klementinen, and H. Mannila. Modeling kdd processes within the inductive database framework. In M. K. Mohania and A. M. Tjoa, editors, *DataWarehousing and Knowledge Discovery (DaWaK 1999)*, pages 293–302, 1999.

[14] J.-F. Boulicaut. *Habilitation thesis*. PhD thesis, INSA-Lyon (France), 2001.

[15] J-F. Boulicaut and A. Bykowski. Frequent closures as a concise representation for binary data mining. In Lecture Notes in Artificial Intelligence Springer-Verlag, editor, *Fourth Pacific-Asia Conference on Knowledge Discovery and Data Mining PaKDD'00*, volume 1805, pages 62–73, Kyoto(Japan), April 18-20 2000.

[16] J-F. Boulicaut, A. Bykowski, and C. Rigotti. Approximation of frequency queries by means of free sets. In Springer-Verlag Lecture Notes in Artificial Intelligence, editor, *Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases PKDD'00*, volume 1910, pages 75–85, Lyon(France), September 13-16 2000.

[17] S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting ans implication rules for market basket data. In *ACM SIGMOD'97*, pages 255–264, 1997.

[18] A. Bykowski and C. Rigotti. A condensed representation to find frequent patterns. In *PODS Int. Conf. Principles of Database Systems*, 2001.

[19] Toon Calders and Bart Goethals. Mining all non derivable frequent itemsets. In *6th European Conference, PKDD 2002*, Helsinki, Finland, August 2002.

[20] A. Casali, R. Cichetti, and L. Lakhal. Treillis relationnel : une structure algébrique pour le data mining multidimensionnel. In *18 journées de bases de données avancées*, Evry, Octobre 2002.

[21] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Ninth ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[22] S.S. Cosmadakis, P.C. Kanellakis, and N. Spyratos. Partition semantics for relations. In *Journal of Computer and System Sciences*, pages 203–233, 1986.

[23] Luc De Raedt. Towards query evaluation and optimization in inductive databases using version spaces. In P.L. Lanzi and R. Meo, editors, *Database Support for Data Mining Applications*, LNCS 2682, pages 117–134. Springer Verlag, 2004.

[24] Luc De Raedt and Stefan Kramer. The levelwise version space algorithm and its application to molecular fragment finding. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 853–862. Morgan Kaufmann, 2001.

[25] L. Dehaspe. *Frequent Pattern Discovery in First-Order-Logic*. PhD thesis, Katholieke Universiteit Leuven, December 1998.

[26] L. Dehaspe and L. De Raedt. Mining association rules in multiple relations. In S. Dzeroski and Springer-Verlag N. Lavrac, editors, *7th International Workshop on Inductive Logic Programming*, volume 1297, pages 125–132, 1997.

[27] L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. In Boston Kluwer Academic Publishers, editor, *Data Mining and Knowledge Discovery*, volume 3, pages 7–36, Boston, 1999.

[28] C.T. Diop. *Etude et mise en oeuvre des aspects itératifs de l'extraction de règles d'association dans une base de données*. PhD thesis, Université de Tours, France, 2003.

[29] C.T. Diop, A. Giacometti, D. Laurent, and N. Spyratos. Extraction incrémentale de règles d'association par combinaison de tâches d'extraction. In *BDA' 2001, 17ièmes Journées de Bases de Données Avancées*, Agadir, Maroc, 29 octobre - 2 novembre 2001.

[30] C.T. Diop, A. Giacometti, D. Laurent, and N. Spyratos. Composition of mining contexts for efficient extraction of association rules. In Springer-Verlag, editor, *EDBT'02*, volume LNCS 2287, pages 106–123, 2002.

[31] Cheung D.W., Han J., Ng V., and Wong C.Y. Maintenance of discovered association rules in large databases : An incremental updating technique. In *12th ICDE*, 1996.

[32] S. Dzeroski. Inductive logic programming an knowledge discovery in databases. In AAAI-MIT Press, editor, *In Advances in Knowledge Discovery and Data Mining*, pages 117–152, 1996.

[33] S. Dzeroski and N. Lavrac. *Relational Data Mining*. Springer, Berlin, 2001.

[34] A. Faye. *Découverte d'Associations entre tables d'une base de données : Une Approche par la Logique du Premier Ordre*. PhD thesis, Université Paris-sud, Paris 11, France, 2000.

[35] A. Faye, A. Giacometti, D. Laurent, and N. Spyratos. Mining rules in databases with multiple tables : Problems and perspectives. In *3rd International Conference on Computing Anticipatory Systems (CASYS'99)*, Liège, Belgique, 9-14 août 1999.

[36] U. Fayyad, G.P. Shapiro, and P. Smyth. From data mining to knowledge discovery : An overview. In AAAI-MIT Press, editor, *In Advances in Knowledge Discovery and Data Mining*, pages 1–34, 1996.

[37] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis : Mathematical Foundations*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997. Translator-C. Franzke.

[38] A. Giacometti, D. Laurent, and C. T. Diop. *Condensed representations of sets of mining queries*, volume LNCS, to appear. Springer-Verlag, 2003.

[39] A. Giacometti, D. Laurent, C. T. Diop, and N. Spyratos. La découverte de règles d'associations entre vues : vers un processus incrémental. In *BDA'2000*, Blois, France, 2000.

[40] A. Giacometti, D. Laurent, C. T. Diop, and N. Spyratos. Mining from views : An incremental approach. *International Journal Information Theories & Applications*, 9 (Également RR LI/E3i, Univ. de Tours), 2002.

[41] Arnaud Giacometti, Dominique Laurent, and Cheikh Talibouya Diop. Condensed representations for sets of mining queries. In *First ECML/PKDD-2002 International Workshop on Knowledge Discovery in Inductive Databases (KDID'02)*, pages 5–19, 2002.

[42] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Datacube : A relational aggregation operator generalizing group by, cross-tab, and sub-totals. In *IEEE ICDE*, pages 152–159, 1996.

[43] Carl A. Gunter, Teow-Hin Ngair, and Devika Subramanian. The common order-theoretic structure of version spaces and atmss. *Artificial Intelligence*, 95(2) :357–407, 1997.

[44] J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane. Dmql : A data mining query language for relational databases. In *SIGMOD Workshop DMKD'96*, pages 27–34, Montreal (Canada), 1996.

[45] J. Han and M. Kamber. *Data Mining : Concepts and Techniques*. Morgan Kaufmann, San Francisco, 2000.

[46] J. Han, J. Pei, G. Dong, and K. Wang. Efficient computation of iceberg cubes with complex measures. In *ACM SIGMOD'01*, Santa Barabara, CA, USA, May 2001.

[47] H. Hirsh. Generalizing version spaces. In *Machine Learning*, volume 17, pages 5–46, 1994.

[48] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. In *Communications of the ACM*, volume 39, pages 58–64, November 1996.

[49] T. Imielinski and A. Virmani. Msql : A query language for database mining. In *Data Mining and Knowledge Discovery*, volume 3, pages 373–408, December 1999.

[50] B. Jeudy. *Optimisation de requêtes inductives : Application à l'extraction sous contraintes de règles d'associations*. PhD thesis, Université Lyon I, INSA de Lyon, France, décembre 2002.

[51] B. Jeudy and J. F. Boulicault. Using condensed representations for interactive association rule mining. In *6th European Conference, PKDD 2002*, Helsinki, Finland, August 2002.

[52] M. Kamber, J. Han, and J. Chiang. Metarule-guided mining of multi-dimensional association rules using data cubes. In *International Conference on Data Mining and Knowledge Discovery (KDD'97)*, pages 207–210, Newport Beach, USA, 1997.

[53] Y. Kodratoff. Research topics in knowledge discovery in data and texts. In *3rd SIGMOD 98 Workshop on Research Issues in Data Mining and Knowledge Discovery*, Seattle, WA, 1998.

[54] Y. Kodratoff. Rating the interest of rules induced from data and within texts. In *12-th IEEE- International Conference on Database and Expert Systems Applications - DEXA 2001*, Munich, Sept. 2001.

[55] M. Kryszkiewicz. Concise representation of frequent patterns bases on disjunction-free generators. In *IEEE Int., Conf., on Data Mining*, pages 305–312, 2001.

[56] P. Langley. *Elements of Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1996.

[57] M. Laporte, N. Novelli, R. Cichetti, and L. Lakhal. Computing full and iceberg datacubes using partitions. In *13th International Symposium, ISMIS*, Lyon, France, June 27-29 2002.

[58] N. Lavrac, P. Flash, and B. Zupan. Rule evaluation measures : A unifying view. In *9th Intl. Workshop ILP'99*, Bled, Slovenia, June 1999.

[59] J. Looyd. *Foundations of logic programming.* Springer Verlag, second extended edition edition, 1987.

[60] H. Mannila. Inductive databases and condensed representations for data mining. In *ILPS'97*, pages 21–30, 1997.

[61] H. Mannila. Theoritical frameworks for data mining. In *SIGKDD Explorations*, pages 30–32, 2000.

[62] H. Mannila and H. Toivonen. Multiple uses of frequent sets and condensed representations (extended abstract). In *KDD'96*, pages 189–194, Portland, Oregon, USA, 1996.

[63] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. C-1997-8, University of Helsinki, Helsinki, 1997.

[64] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. C-1997-8, University of Helsinki, Helsinki, 1997.

[65] R. Meo, G. Psaila, and S. Ceri. A new sql-like operator for mining association rules. In *22nd VLDB Conf.*, Mumbai (Bombay), India, 1996.

[66] R. Meo, G. Psaila, and S. Ceri. An extension to sql for mining association rules. In *Data Mining and Knowledge Discovery*, volume 9, pages 275–300, October 1997.

[67] T.M. Mitchell. Generalization as search. In *Artificial Intelligence*, volume 18, pages 203–226, 1982.

[68] F. Moal. *Langages de biais en apprentissage symbolique.* PhD thesis, Université d'Orléans, Décembre 2000.

[69] T. Morzy, M. Wojciechowski, and M. Zakrzewicz. Materialized data mining views. In Springer-Verlag, editor, *PKDD'00*, volume 1910, pages 65–74, Lyon, France, September 2000. LNAI.

[70] S. Muggleton. Inverse entailment and progol. In *New generation Computing, Special Issue on Logic Programming*, 1995.

[71] B. Nag, P. Deshpande, and D.J. DeWitt. Caching for multi-dimensional data mining queries. In *International Conference on Information Systems, Analysis and Synthesis (SCI'01)*, 2001.

[72] B. Nag, P.M. Deshpande, and D.J. DeWitt. Using a knowledge cache for interactive discovery of association rules. In *5th KDD Conference*, 1999.

[73] C. Nédellec, C. Rouveirol, F. Bergadano, and B. Tausend H. Adé. Declarative bias in ilp. In IOS Press, editor, *Advances in Inductive Logic Programming*, pages 82–103, 1996.

[74] J.S. Park, M.S. Chen, and P.S. Yu. An effective hash-based algorithm for mining association rules. In *ACM-SIGMOD Int.Conf. Management of Data (SIGMOD'95)*, pages 175–186, San Jose, CA, May 1995.

[75] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1), 1999.

[76] L. De Raedt. Query execution and optimization for inductive databases. In *International Workshop DTDM'02, In conjunction with EDBT 2002*, pages 19–28, Praha, CZ, 2002.

[77] L. De Raedt. Towards query evaluation and optimization in inductive databases using version spaces. 2003. (to be published).

[78] L.De Raedt, H. Blockeel, L. Dehaspe, and W.V. Laer. Three companions for data mining in first order logic. In Springer, editor, *Relational Data Mining, S. Dzeroski, N. Lavrac*, Berlin, 2001.

[79] R. Reiter. *On closed World DataBases*. Gallaire et Minker, Plenium Press, New York, 1978.

[80] K. A. Ross and D. Srivastava. Fast computation of sparse datacubes. In *23rd VLDB Conf.*, pages 116–125, Athens, Greece, 1997.

[81] S. Sarawagi, R. Agrawal, and A. Gupta. On computing the datacube. Technical Report RJ10026, IBM Almaden Research Center, San Jose, CA, 1996.

[82] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Int.Conf. Very Large Database*, pages 432–443, Zurich, Switzerland, Sept. 1995.

[83] N. Spyratos. The partition model : A deductive database model. In *ACM TODS*, volume 12, pages 1–37, 1987.

[84] R. Srikant and R. Agrawal. Mining generalized association rules. In *VLDB-95*, Zurich, Switzerland, September 1995.

[85] R. Srikant and R. Agrawal. Mining generalized association rules. Technical Report RJ 9963, IBM Almaden Research Center, San Jose, California, San Jose, California 1995.

[86] S. Thomas, S. Bodagala, K. Alsabti, and Sanjay Ranka. An efficient algorithm for mining association rules in large databases. In *VLDB Conference*, pages 432–444, 1997.

[87] H. Toivonen. Sampling large databases for association rules. In *Int. Conf. Very Large Databases*, pages 134–145, Bombay, India, Sept. 1996.

[88] F. Torre. *Intégration des biais de langage à l'algorithme générer et tester ; Contribution à l'apprentissage disjonctif.* PhD thesis, Université Paris 11 (Orsay), Paris, Fev. 2000.

[89] S. Tsur and al. Query flocks : A generalization of association-rule mining. In *SIGMOD Conference*, pages 1–12, 1998.

[90] T. Turmeaux, D. Cassard, A. Salleb, and C. Vrain. Apprentissage de règles caractéristiques. In *Journées francophones d'Extraction et de gestion des Connaissances EGC'2003*, pages 437–448, 2003.

[91] J.D. Ullman. *Principles of Databases and Knowledge-Base Systems*, volume 1. Computer Science Press, Maryland 20850, U.S.A, 1988.

Annexe A

Iterative Computation of Mining Queries based on Condensed Representations

Résumé

In this paper, we propose a general framework for iterative computation of mining queries based on condensed representations. To this end, we first adapt the standard notions of maximal, closed and key patterns introduced in previous works, to the more general case of sets of mining queries defined by monotonic and anti-monotonic selection predicates.

Then, we show in a particular case how condensed representations of a set $\mathcal{Q}$ of mining queries can be used : (i) to optimize the computation of the answer of a new mining query q, (ii) to efficiently modify the condensed representation of the answer of $\mathcal{Q}$ to obtain a condensed representation of the answer of $\mathcal{Q} \cup \{q\}$.

1 Introduction

In the past decades, the problem of discovery of interesting patterns in large databases has motivated many research efforts. Whereas these works have focussed mainly on the efficiency of the algorithms [2, 8, 75], some other issues have been recently considered, among which (i) the problem of efficient storage of the result of a mining query [8, 15] and (ii) the problem of efficient computation of sequence of mining queries using the results of previous queries [5, 51, 71].

In this paper, we propose a general framework for iterative computation of mining queries based on condensed representations of sets of mining queries. In a first part, noting that the union of condensed representations of different mining queries is *not* a condensed representation of the corresponding set of mining queries ([47]), we first extend the notions of maximal, minimal, closed and key patterns to the case of sets of mining queries defined by monotonic and anti-monotonic selection predicates. Then, we propose condensed representations for such sets, in the sense that, given a set $\mathcal{Q}$ of mining queries, the answers of the queries in $\mathcal{Q}$ can be computed based *only* on the condensed representation, i.e., without any access to the data set. This work is a continuation of [41]. In this paper, a condensed representation of the answer of $\mathcal{Q}$ is a set of partial functions defined over a subset of the answers of the queries in $\mathcal{Q}$.

In a second part, we restrict our approach to the case of conjunctive queries involving only syntactic constraints and a monotonic increasing function (such as the support function). In this case, we show how condensed representations of a set $\mathcal{Q}$ of mining queries can be used :

1. To optimize the computation of the answer of a new mining query q.
2. To efficiently modify the condensed representation of the answer of $\mathcal{Q}$ to obtain a condensed representation of the answer of $\mathcal{Q} \cup \{q\}$.

Comparing our approach to that of [24, 76], we note that in [24, 76] the authors also consider conjunctive queries made of monotonic and anti-monotonic primitives, which correspond to what we call simple mining queries. Moreover, it is shown in [24, 76] that the answer to one such query can be represented by its minimal and maximal elements only. However, contrary to the present paper, the case of *sets* of queries is not considered.

On the other hand, in [15], the authors consider conjunctive queries, but they only consider the case of anti-monotonic constraints. Then, they use a caching technique to store condensed representations of the answers to these queries to optimize the computation of new mining queries. However, each answer is condensed separately and stored in the cache, whereas our approach allows to benefit from relationships between the queries in order to further condense the answers to the queries.

Thus, our approach can be seen as an extension of [24, 76] and [15].

The paper is organized as follows : In Section 2, we give the formal definitions of the basic concepts of our approach. Section 3 introduces condensed representations of sets of mining queries. In Section 4, given the condensed representations of the answer of a set of mining queries $\mathcal{Q}$, we propose an algorithm to regenerate the answer of any mining query in $\mathcal{Q}$. Then, in Section 5, we present an algorithm to optimize the computation of a new mining query. Finally, in Section 6, we show how to update the condensed representation of a set of mining queries when a new mining query is considered. In Section 7, we conclude the paper and we propose further research directions based on this work.

2 Mining Query and Condensed Representations

2.1 Basic Definitions

In our formalism, we assume that we are given :

1. A data $\mathbf{\Delta}$ of all data sets from which the patterns are to be discovered. For instance, $\mathbf{\Delta}$ can be thought of as being the set of all instances of a given relation schema.
2. A set of patterns $\mathbb{L}$ and a partial ordering $\preceq$ over $\mathbb{L}$. Given two patterns φ_1, φ_2 in $\mathbb{L}$, we say that φ_1 is more specific than φ_2 (or that φ_2 is more general than φ_1) if we have $\varphi_1 \preceq \varphi_2$. Moreover, we assume that there exist a minimal element and a maximal element in $\mathbb{L}$, denoted $\bot$ and $\top$ respectively, such that for every pattern $\varphi \in \mathbb{L}$, $\bot \preceq \varphi \preceq \top$.
3. A set of selection predicates $\mathbb{Q}$, a selection predicate $q \in \mathbb{Q}$ being a boolean function defined over $\mathbb{L} \times \mathbf{\Delta}$. Moreover, given a pattern φ in $\mathbb{L}$ and a data set Δ in $\mathbf{\Delta}$, we say that φ is interesting in Δ with respect to q if $q(\varphi, \Delta) = true$.
4. A set of measure functions $\mathbb{F}$, a measure function being a function defined from $\mathbb{L} \times \mathbf{\Delta}$ to $\Re$.

In the following example, that will be used as a running example throughout the paper, we illustrate these notions in the classical association rule mining problem of [2].

Running Example 1 *Given a set of items Items, the set of patterns $\mathbb{L}$ considered in our approach is the set of all subsets of Items, i.e., $\mathbb{L} = 2^{Items}$. Moreover, the partial ordering over $\mathbb{L}$ that we consider is set inclusion : given two patterns φ and φ' in $\mathbb{L}$, we say that $\varphi \preceq \varphi'$ if $\varphi' \subseteq \varphi$. Finally, the minimal element $\bot$ of $\mathbb{L}$ is the set of items Items, whereas its maximal element $\top$ is the empty set.*

In this context, a data set Δ is defined by a set of transactions Tr and a function it from Tr to $\mathbb{L}$. Given a transaction $x \in Tr$, $it(x)$ is the set of items in transaction x. The support of a pattern is an example of measure function of $\mathbb{F}$. More precisely, for every pattern φ, the support of φ in Δ, denoted by $sup(\varphi, \Delta)$, is defined by :

$$sup(\varphi, \Delta) = |\{x \in Tr \mid it(x) \preceq \varphi\}|/|Tr|.$$

Note that, given a minimal support threshold minsup, we can consider the selection predicate q defined by : for every pattern $\varphi \in \mathbb{L}$, $q(\varphi, \Delta) = true$ if $sup(\varphi, \Delta) \geq minsup$.

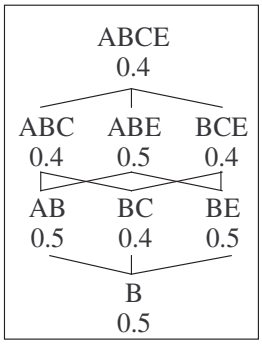
In the rest of the paper, we consider the case where the set of items is $Items = \{A, B, C, D, E\}$ and where the set of transactions is $Tr = \{1, 2, \ldots, 10\}$. For the sake of simplicity, sets of items are denoted by the concatenation of their elements, e.g. the set of items $\{A, B, C\}$ is denoted by ABC. A function it from Tr to $\mathbb{L}$ that defines a data set Δ is represented in the table of Figure A.1. □

In this paper, we only consider only selection predicates that are monotonic or anti-monotonic, and measure functions that are monotonic increasing.

Definition 1 - Monotonicity. *Let q be a selection predicate and f be a measure function.*

- *q is* monotonic *if for every data set Δ in $\mathbf{\Delta}$, we have : $(\forall(\varphi_1, \varphi_2) \in \mathbb{L}^2)(\varphi_1 \preceq \varphi_2 \wedge q(\varphi_2, \Delta) = true \rightarrow q(\varphi_1, \Delta) = true)$.*
- *q is* anti-monotonic *if for every data set Δ in $\mathbf{\Delta}$, we have : $(\forall(\varphi_1, \varphi_2) \in \mathbb{L}^2)(\varphi_1 \preceq \varphi_2 \wedge q(\varphi_1, \Delta) = true \rightarrow q(\varphi_2, \Delta) = true)$.*
- *f is* monotonic increasing *if for every data set Δ in $\mathbf{\Delta}$, we have : $(\forall(\varphi_1, \varphi_2) \in \mathbb{L}^2)(\varphi_1 \preceq \varphi_2 \rightarrow f(\varphi_1, \Delta) \leq f(\varphi_2, \Delta)$.*

In the following, we denote by $\mathbb{A}$ the set of all anti-monotonic selection predicates and by $\mathbb{M}$ the set of all monotonic selection predicates. Moreover, we denote by $\mathbb{I}$ the set of all monotonic increasing measure functions. We also consider selection predicates that are data independent.



Tr	Set of Items
1	A
2	DE
3	ABCE
4	ABE
5	ABCDE
6	ACD
7	ABCE
8	AE
9	ABCDE
10	CD

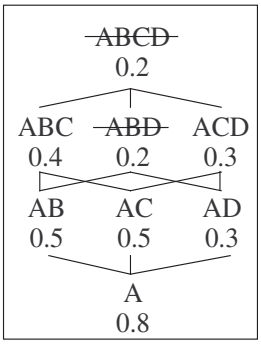


FIG. A.1 – Example of data set and sub-lattices of interesting patterns.

Definition 2 - Data Independency. *Let q be a selection predicate in $\mathbb{Q}$. q is* data independent *(or* independent *for short) if there exists a function $\widetilde{q}$ from $\mathbb{L}$ to $\{true, false\}$ such that for every data set Δ in $\mathbf{\Delta}$ and every pattern φ in $\mathbb{L}$, $q(\varphi, \Delta) = \widetilde{q}(\varphi)$.*

In the literature [14], independent selection predicates are also called *syntactical constraints*. In the following, we denote by $\widetilde{\mathbb{Q}}$ the set of all independent selection predicates, and by $\overline{\mathbb{Q}}$ the complement of $\widetilde{\mathbb{Q}}$ in $\mathbb{Q}$, i.e., $\overline{\mathbb{Q}} = \mathbb{Q} \setminus \widetilde{\mathbb{Q}}$.

Example 1 *In the context of our Running Example 1, let $q_1 = m_1 \wedge a_1$ and $q_2 = m_2 \wedge a_2$ be selection predicates defined for every pattern φ in $\mathbb{L}$ and every data set $\Delta \in \mathbf{\Delta}$ by :*

- *$m_1(\varphi, \Delta) = true$ if $B \subseteq \varphi$, $m_2(\varphi, \Delta) = true$ if $A \subseteq \varphi$.*
- *$a_1(\varphi, \Delta) = \overline{a_1}(\varphi, \Delta) \wedge \widetilde{a_1}(\varphi, \Delta)$, where $\overline{a_1}(\varphi, \Delta) = true$ if $sup(\varphi, \Delta) \geq 0.4$, and $\widetilde{a_1}(\varphi, \Delta) = true$ if $\varphi \subseteq ABCE$.*
- *$a_2(\varphi, \Delta) = \overline{a_2}(\varphi, \Delta) \wedge \widetilde{a_2}(\varphi, \Delta)$ where $\overline{a_2}(\varphi, \Delta) = true$ if $sup(\varphi, \Delta) \geq 0.3$, and $\widetilde{a_2}(\varphi, \Delta) = true$ if $\varphi \subseteq ABCD$.*

It is easy to see that the selection predicates m_i ($i = 1, 2$) are monotonic, whereas the selection predicates $\widetilde{a_i}$ and $\overline{a_i}$ ($i = 1, 2$) are anti-monotonic. Moreover, the selection predicates m_i and $\widetilde{a_i}$ ($i = 1, 2$) are data independent. □

Given a measure function $f \in \mathbb{I}$, let $\overline{\mathbb{A}}_f$ be the set of anti-monotonic selection predicates q defined for every $\varphi \in \mathbb{L}$ by : $q(\varphi) = true$ if $f(\varphi, \Delta) \geq \alpha(q)$ where $\alpha(q)$ is a constant in $\Re$. Let $\overline{\mathbb{M}}_f$ be the set of monotonic selection predicates q defined for every $\varphi \in \mathbb{L}$ by : $q(\varphi) = true$ if $f(\varphi, \Delta) \leq \beta(q)$ where $\beta(q)$ is a constant in $\Re$.

In the rest of the paper, we only consider selection predicates q of the form $q = \widetilde{q} \wedge \overline{q}$ where $\widetilde{q}$ is an independent selection predicate, and $\overline{q} = \overline{m} \wedge \overline{a}$ with $\overline{m} \in \overline{\mathbb{M}}_f$ and $\overline{a} \in \overline{\mathbb{A}}_f$. We denote by $\mathbb{Q}_f$ this set of predicates, i.e.

$$\mathbb{Q}_f = \{\widetilde{q} \wedge \overline{m} \wedge \overline{a} \mid \widetilde{q} \in \widetilde{\mathbb{Q}}, \overline{m} \in \overline{\mathbb{M}}_f \text{ and } \overline{a} \in \overline{\mathbb{A}}_f\}$$

In our approach, selection predicates are compared according to the following definition.

Definition 3 - Selectivity. *Let q_1 and q_2 be two selection predicates. q_1 is* more selective *than q_2, denoted by $q_1 \sqsubseteq q_2$, if for every data set Δ in $\mathbf{\Delta}$ and every pattern φ in $\mathbb{L}$, we have : if $q_1(\varphi, \Delta) = true$, then $q_2(\varphi, \Delta) = true$.*

We can note that $\overline{\mathbb{A}}_f$ and $\overline{\mathbb{M}}_f$ are totally ordered sets of predicates with respect to $\sqsubseteq$. Indeed, for every $(q_1, q_2) \in {\mathbb{Q}_f}^2$, we have :

$$\overline{a}_1 \sqsubseteq \overline{a}_2 \text{ if } \alpha(q_2) \leq \alpha(q_1) \qquad \text{and} \qquad \overline{m}_1 \sqsubseteq \overline{m}_2 \text{ if } \beta(q_1) \leq \beta(q_2).$$

In the rest of the paper, we consider a *fixed* data set Δ in $\mathbf{\Delta}$. Therefore, for notational convenience, we shall omit Δ in the subsequent definitions and propositions. For instance, referring to the previous two definitions, $q(\varphi, \Delta)$ and $f(\varphi, \Delta)$ will be simply denoted by $q(\varphi)$ and $f(\varphi)$, respectively.

In our approach, we define two types of mining query.

Definition 4 - Mining Query. *A* simple mining query *is a selection predicate q. Given a data set Δ, the* answer of q in Δ*, denoted by $sol(q)$, is defined by :*

$$sol(q) = \{\varphi \in \mathbb{L} \mid q(\varphi) = true\}.$$

$sol(q)$ denotes the set of all interesting patterns *in $\mathbb{L}$ with respect to q.*

An extended mining query *is a pair (q, f) where q is a selection predicate and f is a measure function. Given a data set Δ, the* answer of (q, f) in Δ*, denoted by $ans(q, f)$, is defined by :*

$$ans(q, f) = \{(\varphi, f(\varphi)) \mid \varphi \in sol(q)\}.$$

Note that an algorithm proposed in [14] can compute directly $sol(q)$ and $ans(q, f)$ if $q = m \wedge a$ with $m \in \mathbb{M}$ and $a \in \mathbb{A}$.

Example 2 *Let q_1 and q_2 be the selection predicates defined in Example 1. q_1 and q_2 are simple mining queries in $\mathbb{Q}_{sup}$. Moreover, it is easy to see from the table in Figure A.1 that :*

$$\begin{aligned} sol(m_1 \wedge a_1/\Delta) = &\quad \{B, AB, BC, BE, ABC, ABE, BCE, ABCE\} \\ sol(m_2 \wedge a_2/\Delta) = &\quad \{A, AB, AC, AD, ABC, ACD\} \end{aligned}$$

On the other hand, (q_1, sup) and (q_2, sup) are examples of extended mining queries, and we have :

$$\begin{aligned} ans(q_1, sup/\Delta) = &\quad \{(B, 0.5),\ (AB, 0.5),\ (BC, 0.4),\ (BE, 0.5),\ (ABC, 0.4), \\ &\quad (ABE, 0.5),\ (BCE, 0.4),\ (ABCE, 0.4)\} \\ ans(q_2, sup/\Delta) = &\quad \{(A, 0.5),\ (AB, 0.5),\ (AC, 0.5),\ (AD, 0.3),\ (ABC, 0.4), \\ &\quad (ACD, 0.3)\} \end{aligned}$$

The answers in Δ of (q_1, sup) and (q_2, sup) are also represented in Figure A.1. □

2.2 Maximal, Closed and Key Patterns

In this paper, we consider only simple mining queries that are defined by conjunction of anti-monotonic and monotonic selection predicates. In this case, the answer of a simple mining query can be represented by its most specific and most general patterns [47, 63].

Definition 5 *Let $q = m \wedge a$ be simple mining queries with $m \in \mathbb{M}$ and $a \in \mathbb{A}$.*

- *The set of* most specific patterns in *$sol(q)$, denoted by $S(q)$, is defined by :*

$$S(q) = min_{\preceq}(sol(q)) = \{\varphi \in sol(q) \mid (\nexists \varphi' \in sol(q))(\varphi' \prec \varphi)\}.$$

- *The set of* most general patterns in *$sol(q)$, denoted by $G(q)$, is defined by :*

$$G(q) = max_{\preceq}(sol(q)) = \{\varphi \in sol(q) \mid (\nexists \varphi' \in sol(q))(\varphi \prec \varphi')\}.$$

The following lemma shows that $sol(q)$ can be regenerated from the set $\{S(q), G(q)\}$ without accessing the data set. This is why $\{S(q),\ G(q)\}$ is called in the literature a condensed representation of $sol(q)$.

Lemma 1 *Let $q = m \wedge a$ be a simple mining query with $m \in \mathbb{M}$ and $a \in \mathbb{A}$. We have : $sol(q) = \{\varphi \in \mathbb{L} \mid (\exists \varphi_s \in S(q))(\exists \varphi_g \in G(q))(\varphi_s \preceq \varphi \preceq \varphi_g)\}$.*

Therefore, we have the following proposition.

Proposition 2.1 *Let $q = m \wedge a$ be a simple mining query with $m \in \mathbb{M}$ and $a \in \mathbb{A}$. The set $\{G(q),\ S(q)\}$ is a condensed representation of $sol(q)$, i.e. $G(q),\ S(q) \models sol(q)$.*

Proof : *Let us consider the function F defined by : $F(X_1, X_2) = \{\varphi \in \mathbb{L} \mid (\exists \varphi_1 \in X_1)(\exists \varphi_2 \in X_2)(\varphi_1 \preceq \varphi \preceq \varphi_2)\}$. Using Lemma 1, we have $sol(q) = F(S(q), G(q))$. Moreover, F is independent from the data set Δ since $\preceq$ does not depend on Δ. Finally, we have $S(q) \cup G(q) \subseteq sol(q)$, which completes the proof.*

Example 3 *Let q_1 and q_2 be the simple mining queries as given in Example 1. We have : $G(q_1) = \{B\}$, $S(q_1) = \{ABCE\}$, $G(q_2) = \{A\}$, and $S(q_2) = \{ABC, ACD\}$.* □

Now, we give alternative definitions of the notions of closed and key patterns introduced in [8, 15, 75]. To this end, given a measure function f, we consider the partial ordering $\leq_f$ defined for every pair of patterns (φ, φ') by :

$$\varphi \leq_f \varphi' \ \text{ if } \ \varphi \preceq \varphi' \text{ and } f(\varphi) = f(\varphi').$$

Definition 6 *Let q be a mining query in $\mathbb{Q}$ and f be a measure function in $\mathbb{F}$. Let Δ be a data set and φ be a pattern in $\mathbb{L}$.*

- *The set of all* interesting closed patterns *in Δ with respect to q and f, denoted by $SC(q, f)$, is defined by :*
$$SC(q, f) = min_{\leq_f}(sol(q)).$$
- *The set of all* interesting key patterns *in Δ with respect to q and f, denoted by $GK(q, f)$, is defined by :*
$$GK(q, f) = max_{\leq_f}(sol(q)).$$

It can be shown that our notions of interesting closed patterns and interesting key patterns coincide with those of [8, 15, 75] in the context of classical association rules mining [2].

Moreover, it is easily seen that for every extended mining query (q, f) with $q \in \mathbb{Q}$ and $f \in \mathbb{I}$, we have $S(q) \subseteq SC(q, f)$ and $G(q) \subseteq GK(q, f)$. More precisely, the following lemma holds.

Lemme 1 *Let q be a selection predicate in $\mathbb{Q}$ and f be a monotonic increasing measure function in $\mathbb{I}$. We have :*

$$S(q) = min_{\preceq}(SC(q, f)) \ \text{ and } \ G(q) = max_{\preceq}(GK(q, f)).$$

Proof : *We first show that $S(q) \subseteq min_{\preceq}(SC(q, f))$. Let $\varphi \in S(q)$. There does not exist a pattern $\varphi' \in sol(q)$ such that $\varphi' \prec \varphi$. Therefore, there does not exist a pattern $\varphi' \in sol(q)$ such that $\varphi' \prec \varphi$ and $f(\varphi') = f(\varphi)$, which shows that $\varphi \in SC(q, f)$. Assume now that $\varphi \notin min_{\preceq}(SC(q, f))$. Then, there exists $\varphi' \in SC(q, f)$ such that $\varphi' \prec \varphi$, which is in contradiction with the hypothesis $\varphi \in S(q)$. Hence, we have $S(q) \subseteq min_{\preceq}(SC(q, f))$.*

Now, we show that $min_{\preceq}(SC(q, f)) \subseteq S(q)$. Let $\varphi \in min_{\preceq}(SC(q, f))$. Assume that $\varphi \notin S(q)$. Then, there exists $\varphi' \in S(q)$ such that $\varphi' \prec \varphi$. Since it has been shown above that $S(q) \subseteq min_{\preceq}(SC(q, f))$, we have that $\varphi' \in SC(q, f)$. This is in contradiction with the hypothesis $\varphi \in min_{\preceq}(SC(q, f))$. Hence, we have $min_{\preceq}(SC(q, f)) \subseteq S(q)$.

Thus the proof that $S(q) = min_{\preceq}(SC(q, f))$ is complete. In the same way, it can be shown that $G(q) = max_{\preceq}(GK(q, f))$, which completes the proof.

The following lemma states that given any pattern φ in $sol(q)$, $f(\varphi)$ can be computed based on $SC(q, f)$ or $GK(q, f)$.

Lemma 2 *Let q be a selection predicate in $\mathbb{Q}$ and f be a monotonic increasing measure function in $\mathbb{I}$. For every interesting pattern φ in $sol(q)$, we have :*
- $f(\varphi) = max\{f(\varphi') \mid \varphi' \in SC(q,f)$ *and* $\varphi' \preceq \varphi\}$, *and*
- $f(\varphi) = min\{f(\varphi') \mid \varphi' \in GK(q,f)$ *and* $\varphi \preceq \varphi'\}$

where min and max denote respectively the minimum and maximum functions according to the standard ordering of real numbers.

Proof : *Let $\varphi \in sol(q)$ and $X(\varphi) = \{\varphi' \in sol(q) \mid \varphi' \preceq \varphi$ and $f(\varphi') = f(\varphi)\}$. Since $\varphi \in X(\varphi)$, we know that $Y(\varphi) = min_{\preceq}(X(\varphi))$ is not empty. Given any $\varphi'' \in Y(\varphi)$, assume that $\varphi'' \notin SC(q,f)$. Then, there exists $\varphi' \in sol(q)$ such that $\varphi' \prec \varphi''$ and $f(\varphi') = f(\varphi'')$, which shows that $\varphi' \in X(\varphi)$ and contradicts the fact that φ'' is minimal in $X(\varphi)$ with respect to $\preceq$. Hence, there exists $\varphi_c \in SC(q,f)$ such that $\varphi_c \preceq \varphi$ and $f(\varphi_c) = f(\varphi)$.*
On the other hand, for every $\varphi' \in SC(q,f)$ such that $\varphi' \preceq \varphi$, we have $f(\varphi') \leq f(\varphi)$. Therefore, we have $f(\varphi) = max\{f(\varphi') \mid \varphi' \in SC(q,f)$ and $\varphi' \preceq \varphi\}$. Since the fact that $f(\varphi) = min\{f(\varphi') \mid \varphi' \in GK(q,f)$ and $\varphi \preceq \varphi'\}$ can be shown in the same way, the proof is complete.

Let (q,f) be an extended mining query. In the following, we denote by $\lambda_{SC}[q,f]$ and $\lambda_{GK}[q,f]$ the sets defined by :
- $\lambda_{SC}[q,f] = \{(\varphi, f(\varphi)) \mid \varphi \in SC(q,f)\}$, and
- $\lambda_{GK}[q,f] = \{(\varphi, f(\varphi)) \mid \varphi \in GK(q,f)\}$.

Based on the previous two lemmas, it is easy to see that the following proposition holds.

Proposition 2.2 *Let $q = m \wedge a$ be a simple mining query with $m \in \mathbb{M}$, $a \in \mathbb{A}$, and let f be a monotonic increasing measure function in $\mathbb{I}$. The sets $\{S(q),\ G(q),\ \lambda_{SC}[q,f]\}$ and $\{S(q),\ G(q),\ \lambda_{GK}[q,f]\}$ are extended condensed representations of $ans(q,f)$, i.e.*
- $S(q),\ G(q),\ \lambda_{SC}[q,f] \models_e ans(q,f)$, *and*
- $S(q),\ G(q),\ \lambda_{GK}[q,f] \models_e ans(q,f)$.

Proof : *Let us consider the function F defined by : $F(X_1, X_2, Y) = \{(\varphi, \alpha) \in \mathbb{L} \times \Re \mid (\exists \varphi_1 \in X_1)(\exists \varphi_2 \in X_2)(\varphi_1 \preceq \varphi \preceq \varphi_2)$ and $\alpha = max\{\alpha' \mid (\varphi', \alpha') \in Y \wedge \varphi' \preceq \varphi\}\}$. Using Lemma 1 and Lemma 2, we have $ans(q,f) = F(S(q), G(q), \lambda_{SC}[q,f])$. Moreover, F is independent from the data set Δ since $\preceq$ does not depend on Δ, and $S(q) \cup G(q) \cup SC(q,f) \subseteq sol(q)$. Therefore, $\{S(q),\ G(q),\ \lambda_{SC}[q,f]\}$ is an extended condensed representation of $ans(q,f)$. Since the fact that $S(q),\ G(q),\ \lambda_{GK}[q,f] \models_e ans(q,f)$ can be shown in the same way, the proof is complete.*

Exemple 2.1 *Let q_1 be the simple mining query as defined in our Running Example 1. We can see that :*
- $GK(q_1, sup) = \{B, BC\}$,
- $SC(q_1, sup) = \{ABE, ABCE\}$,
- $\lambda_{GK}[q_1, sup] = \{(B, 0.5),\ (BC, 0.4)\}$, *and*
- $\lambda_{SC}[q_1, sup] = \{(ABE, 0.5), (ABCE, 0.4)\}$.

Recalling that $G(q_1) = \{B\}$ and $S(q_1) = \{ABCE\}$, and using Proposition 2.2, we know that :
- $S(q_1),\ G(q_1),\ \lambda_{SC}[q_1, sup] \models_e ans(q_1, sup)$, *and*
- $S(q_1),\ G(q_1),\ \lambda_{GK}[q_1, sup] \models_e ans(q_1, sup)$.

3 Condensed Representations of Sets of Mining Queries

In this section, we show how condensed representation can be defined in the case of sets of mining queries.

3.1 Definitions

Definition 7 - Set of Mining Queries. *Let $\mathcal{Q} = \{q_1, \ldots, q_n\}$ be a set of mining queries. Given a data set Δ, the* answer of $\mathcal{Q}$ in Δ*, denoted by $sol(\mathcal{Q})$, is the set defined by :*

$$sol(\mathcal{Q}) = \bigcup_{q \in \mathcal{Q}} sol(q)$$

Let f be a measure function in $\mathbb{F}$. The answer of $(\mathcal{Q}, f)$ in Δ*, denoted by $ans(\mathcal{Q}, f)$, is the set defined by :*

$$ans(\mathcal{Q}, f) = \bigcup_{q \in \mathcal{Q}} ans(q, f)$$

Let λ be a partial function defined over $X \subseteq \mathbb{L}$. The graph of λ, denoted by $Graph(\lambda)$, is defined by : $Graph(\lambda) = \{(\varphi, \lambda(\varphi)) \mid \varphi \in X\}$. In the rest of the paper, for notational convenience, we shall use the symbol of a function to refer to its graph, i.e. $\lambda = \{(\varphi, \lambda(\varphi)) \mid \varphi \in X\}$.

Definition 8 - Condensed Representation. *Let $\Lambda = \{\lambda_1, \ldots, \lambda_p\}$ be a set of partial functions λ_i defined over $X_i \subseteq \mathbb{L}$ $(i = 1, \ldots, p)$. Given a data set Δ and a set of mining queries $\mathcal{Q}$, Λ is a* condensed representation *of $sol(\mathcal{Q})$, denoted by $\Lambda \models sol(\mathcal{Q})$, if :*

- *$X_1 \cup \ldots \cup X_p \subseteq sol(\mathcal{Q})$, and*
- *there exists a function F independent from Δ such that $sol(\mathcal{Q}) = F(\Lambda)$.*

Given a measure function f in $\mathbb{F}$, Λ is an extended condensed representation *of $ans(\mathcal{Q}, f)$, denoted by $\Lambda \models_e ans(\mathcal{Q}, f)$, if :*

- *$X_1 \cup \ldots \cup X_p \subseteq sol(\mathcal{Q})$, and*
- *there exists a function F independent from Δ such that $ans(\mathcal{Q}, f) = F(\Lambda)$.*

Let $\Lambda = \{\lambda_1, \ldots, \lambda_p\}$ be a set of partial functions λ_i defined over $X_i \subseteq \mathbb{L}$ $(i = 1, \ldots, p)$, and let $\Lambda' = \{\lambda'_1, \ldots, \lambda'_n\}$ be a set of partial functions λ'_i defined over $X'_i \subseteq \mathbb{L}$ $(i = 1, \ldots, n)$. We say that Λ is a condensed representation (extended or not) more concise than Λ' if $(X_1 \cup \ldots \cup X_p) \subseteq (X'_1 \cup \ldots \cup X'_n)$.

Given a set of mining queries $\mathcal{Q}$ and a measure function f, we study condensed representations of $sol(\mathcal{Q})$ and extended condensed representations of $ans(\mathcal{Q}, f)$.

3.2 Maximal Patterns

Let $\mathcal{Q} = \{q_1, \ldots, q_n\}$ be a set of mining queries. On one hand, we know that for every query $q_i \in \mathcal{Q}$ $(i = 1, \ldots, n)$, it is possible to regenerate $sol(q_i)$ from the set $\{S(q_i),\ G(q_i)\}$. Indeed, we have $sol(q_i) = \{\varphi \in \mathbb{L} \mid (\exists \varphi_s \in S(q_i))(\exists \varphi_g \in G(q_i))(\varphi_s \preceq \varphi \preceq \varphi_g)\}$.

On the other hand, it is well known [47] that $sol(\mathcal{Q})$ can not be regenerated from the set $\{S(q_1) \cup \ldots \cup S(q_n),\ G(q_1) \cup \ldots \cup G(q_n)\}$. However, if for every φ in $S(q_1) \cup \ldots \cup S(q_n)$ or in $G(q_1) \cup \ldots \cup G(q_n)$, we keep track of the queries q_i the pattern φ comes from, then $sol(\mathcal{Q})$ and $ans(\mathcal{Q}, f)$ can be condensed. For this reason, we define the partial functions $\lambda_S[Q]$ and $\lambda_G[Q]$ as follows :

Definition 9 *Let $\mathcal{Q} = \{q_1, \ldots, q_n\}$ be a set of mining queries $q_i \in \mathbb{Q}$ $(i = 1, \ldots, n)$. Let $\mathcal{S}(\mathcal{Q})$ and $\mathcal{G}(\mathcal{Q})$ be the sets defined by :*

$$\mathcal{S}(\mathcal{Q}) = \bigcup_{q \in \mathcal{Q}} S(q) \quad \textit{and} \quad \mathcal{G}(\mathcal{Q}) = \bigcup_{q \in \mathcal{Q}} G(q)$$

The partial functions $\lambda_S[Q]$ and $\lambda_G[Q]$ are defined by :

$$\lambda_S[Q] = \{(\varphi, \{q \in \mathcal{Q} \mid \varphi \in S(q)\}) \mid \varphi \in \mathcal{S}(\mathcal{Q})\}$$
$$\lambda_G[Q] = \{(\varphi, \{q \in \mathcal{Q} \mid \varphi \in G(q)\}) \mid \varphi \in \mathcal{G}(\mathcal{Q})\}$$

Given these definitions, we have the following proposition.

Proposition 1 *Let $\mathcal{Q} = \{q_1, \ldots, q_n\}$ be a set of mining queries $q_i = m_i \wedge a_i$ with $m_i \in \mathbb{M}$ and $a_i \in \mathbb{A}$ $(i = 1, \ldots, n)$. The set $\{\lambda_S[Q],\ \lambda_G[Q]\}$ is a condensed representation of $sol(\mathcal{Q})$, i.e. $\lambda_S[Q],\ \lambda_G[Q] \models sol(\mathcal{Q})$.*

Proof : *Let us consider the function defined by :*
$F(\lambda_1, \lambda_2) = \bigcup_{q \in f(\lambda_1, \lambda_2)} \{\varphi \in \mathbb{L} \ / \ (\exists \varphi_g \in g(\lambda_1, q))(\exists \varphi_s \in s(\lambda_2, q))(\varphi_s \preceq \varphi \preceq \varphi_g)$, *where :*
- $f(\lambda_1, \lambda_2) = \{q \in Q \ / \ (\exists (\varphi_1, Q_1) \in \lambda_1)(q \in Q_1)$ *or* $(\exists (\varphi_2, Q_2) \in \lambda_2)(q_2 \in Q_2)\}$;
- $g(\lambda_1, q) = G(q) = \{\varphi_1 \in \mathbb{L} \ / \ (\exists (\varphi_1, Q_1) \in \lambda_1)(q \in Q_1)\}$;
- $s(\lambda_2, q) = S(q) = \{\varphi_2 \in \mathbb{L} \ / \ (\exists (\varphi_2, Q_2) \in \lambda_2)(q \in Q_2)\}$.

We have $F(\lambda_G[Q], \lambda_S[Q]) = sol(\mathcal{Q})$, and moreover, it is clear that $\lambda_G[Q] \cup \lambda_S[Q]) \subseteq sol(\mathcal{Q})$. Therefore, $\{\lambda_S[Q], \lambda_G[Q]\}$ is a condensed representation of $sol(\mathcal{Q})$.

Example 4 *In the context of our Running Example 1, let $q_3 = m_3 \wedge a_3$ and $q_4 = m_4 \wedge a_4$ where m_3, m_4, a_3 and a_4 are selection predicates defined for every pattern $\varphi \in \mathbb{L}$ by :*
- *$m_3(\varphi, \Delta) = true$ if $A \subseteq \varphi$, and $m_4(\varphi, \Delta) = true$ if $AC \subseteq \varphi$,*
- *$a_3(\varphi, \Delta) = true$ if $sup(\varphi, \Delta) \geq 0.3$ and $\varphi \subseteq ABC$, and $a_4(\varphi, \Delta) = true$ if $sup(\varphi, \Delta) \geq 0.3$ and $\varphi \subseteq ABCD$.*

We note that m_3 and m_4 are monotonic selection predicates such that $m_4 \sqsubseteq m_3$, whereas a_3 and a_4 are anti-monotonic selection predicates such that $a_3 \sqsubseteq a_4$. We can see that $S(q_3) = \{ABC\}$, $S(q_4) = \{ABC, ACD\}$, $G(q_3) = \{A\}$ and $G(q_4) = \{AC\}$. Considering $\mathcal{Q} = \{q_3, q_4\}$, we have :

$$\lambda_S[\mathcal{Q}] = \{(ABC, \{q_3, q_4\}), (ACD, \{q_4\})\} \quad \text{and} \quad \lambda_G[\mathcal{Q}] = \{(A, \{q_3\}), (AC, \{q_4\})\}$$

Using Proposition 1, we can see that $\lambda_S[Q],\ \lambda_G[Q] \models sol(\mathcal{Q})$. □

3.3 Closed and Key Patterns

In this subsection, we consider the case of extended condensed representations of a set $\mathcal{Q} = \{q_1, \ldots, q_n\}$ of simple mining queries with $q_i \in \mathbb{Q}$ $(i = 1, \ldots, n)$ involving a monotonic increasing measure function f in $\mathbb{I}$. To this end, recalling that $SC(q_i, f)$ and $GK(q_i, f)$ are the sets of all interesting closed and key patterns in Δ with respect to q_i $(i = 1, \ldots, n)$ and f, we define the sets $\mathcal{SC}(\mathcal{Q}, f)$, $\mathcal{GK}(\mathcal{Q}, f)$ and the partial functions $\lambda_{SC}[\mathcal{Q}, f]$ and $\lambda_{GK}[\mathcal{Q}, f]$ as follows :

Definition 10 *Let $\mathcal{Q} = \{q_1, \ldots, q_n\}$ be a set of mining queries $q_i \in \mathbb{Q}$ $(i = 1, \ldots, n)$. The sets $\mathcal{SC}(\mathcal{Q}, f)$ and $\mathcal{SC}(\mathcal{Q}, f)$ are defined by :*
- *$\mathcal{SC}(\mathcal{Q}, f) = min_{\leq_f}(\bigcup_{q \in \mathcal{Q}} SC(q, f))$, and*
- *$\mathcal{GK}(\mathcal{Q}, f) = max_{\leq_f}(\bigcup_{q \in \mathcal{Q}} GK(q, f))$.*

Moreover, the partial functions $\lambda_{SC}[\mathcal{Q}, f]$ and $\lambda_{GK}[\mathcal{Q}, f]$ are defined by :
- *$\lambda_{SC}[\mathcal{Q}, f] = \{(\varphi, f(\varphi)) \mid \varphi \in \mathcal{SC}(\mathcal{Q}, f)\}$, and*
- *$\lambda_{GK}[\mathcal{Q}, f] = \{(\varphi, f(\varphi)) \mid \varphi \in \mathcal{GK}(\mathcal{Q}, f)\}$.*

Based on Lemma 2, we can state the following proposition :

Proposition 2 *Let $\mathcal{Q} = \{q_1, \ldots, q_n\}$ be a set of simple mining queries with $q_i \in \mathbb{Q}$ $(i = 1, \ldots, n)$ and f be a monotonic increasing measure function in $\mathbb{I}$. For every $i = 1, \ldots, n$ and $\varphi \in sol(q_i)$, we have :*

- $f(\varphi) = max_{\preceq}\{f(\varphi') \mid \varphi' \in \mathcal{SC}(\mathcal{Q}, f)$ *and* $\varphi' \preceq \varphi\}$, *and*
- $f(\varphi) = min_{\preceq}\{f(\varphi') \mid \varphi' \in \mathcal{GK}(\mathcal{Q}, f)$ *and* $\varphi \preceq \varphi'\}$.

PROOF : *Let φ_i in $sol(q_i)$. Using Lemma 2, we know that :*

$$f(\varphi_i) = max\{f(\varphi'_i) \mid \varphi'_i \in SC(q_i, f) \text{ and } \varphi'_i \preceq \varphi_i\}$$

Let $\varphi'_i \in SC(q_i, f)$ such that $\varphi'_i \preceq \varphi_i$ and $f(\varphi'_i) = f(\varphi_i)$. Given the definition of $\mathcal{SC}(\mathcal{Q}, f)$, there exists $\varphi'_j \in \mathcal{SC}(\mathcal{Q}, f)$ such that $\varphi'_j \leq_f \varphi'_i$, i.e. $\varphi'_j \preceq \varphi'_i$ and $f(\varphi'_j) = f(\varphi'_i)$. Thus, there exists $\varphi'_j \in \mathcal{SC}(\mathcal{Q}, f)$ such that $\varphi'_j \preceq \varphi_i$ and $f(\varphi'_j) = f(\varphi_i)$. Finally, for every $\varphi' \in \mathcal{SC}(\mathcal{Q}, f)$ such that $\varphi' \preceq \varphi_i$, we have $f(\varphi') \leq f(\varphi_i)$ since f is a monotonic increasing function. It follows that :

$$f(\varphi_i) = f(\varphi'_j) = max\{f(\varphi') \mid \varphi'_i \in \mathcal{SC}(\mathcal{Q}, f) \text{ and } \varphi' \preceq \varphi_i\}$$

which completes the proof.

PROOF : *The second proof uses similar arguments as that of Proposition 2, and thus is omitted.*

Using propositions 1 and 2, the following theorem holds.

Theorem 1 *Let $\mathcal{Q} = \{q_1, \ldots, q_n\}$ be a set of mining queries with $q_i = m_i \wedge a_i$ where $m_i \in \mathbb{M}$ and $a_i \in \mathbb{A}$ $(i = 1, \ldots, n)$. Let f be a monotonic increasing measure function in $\mathbb{I}$, and let Λ be a condensed representation of $sol(\mathcal{Q})$. The sets $\Lambda \cup \{\lambda_{SC}[\mathcal{Q}, f]\}$ and $\Lambda \cup \{\lambda_{GK}[\mathcal{Q}, f]\}$ are extended condensed representations of $ans(\mathcal{Q}, f)$.*

PROOF : Λ is a condensed representation of $sol(\mathcal{Q}) \Rightarrow (\exists F')(sol(\mathcal{Q}) = F'(\Lambda))$.

$$\begin{aligned} ans(\mathcal{Q}, f) &= \bigcup_{q \in Q} ans(q, f) \\ &= \bigcup_{q \in Q}\{(\varphi, f(\varphi)) \mid \varphi \in sol(q)\} \\ &= \{(\varphi, f(\varphi)) \mid \varphi \in F'(\Lambda)\} \\ &= \{(\varphi, \alpha) \mid \varphi \in F'(\Lambda) \wedge \\ &\quad (\exists \varphi' \in \mathbb{L})(\alpha = max_{\preceq}\{\alpha' \mid ((\varphi', \alpha') \in \lambda_{SC}[Q, f] \text{ and } \varphi' \preceq \varphi)\})\} \end{aligned}$$

Let $\Lambda' = \{\lambda_1, ..., \lambda_n, \lambda_{n+1}\}$ a set of partial functions defined on $X_1, .., X_n, X_{n+1}$.
$F(\lambda') = \{(\varphi, \alpha) \mid \varphi \in F'(\{\lambda_1, ..., \lambda_n\}) \wedge \alpha = max_{\preceq}\{\alpha' \mid (\exists \varphi' \in \mathbb{L})((\varphi', \alpha') \in \lambda_{n+1} \text{ and } \varphi' \preceq \varphi)\}\}$.
So, if we consider a set $\Lambda = \{\lambda_1, ..., \lambda_n\}$ of partial functions defined on $X_1, ..., X_n$, if $\Lambda \models sol(\mathcal{Q})$, then :

- $ans(\mathcal{Q}, f) = F(\Lambda \cup \{\lambda_{SC}[Q, f]\})$ and,
- $X_1 \cup ... \cup X_n \subseteq sol(\mathcal{Q})$.

Moreover, $\lambda_{SC}[Q, f]$ is defined on $SC(Q, f)$ and $SC(Q, f) \subseteq sol(\mathcal{Q})$, so $X_1 \cup ... \cup X_n \cup SC(Q, f) \subseteq sol(\mathcal{Q})$. Therefore, $\Lambda \cup \{\lambda_{SC}[Q, f]\}$ is an extended representation of $ans(\mathcal{Q}, f)$.
In the same way, it can be proved that $\Lambda \cup \{\lambda_{GK}[Q, f]\}$ is a condensed representation of $ans(\mathcal{Q}, f)$.

Example 5 *Let $\mathcal{Q} = \{q_1, q_2\}$ be the set of simple mining queries as defined in Example 1. We recall that :*

- $S(q_1) = \{ABCE\}$ *and* $S(q_2) = \{ABC, ACD\}$, $G(q_1) = \{B\}$ *and* $G(q_2) = \{A\}$,
- $SC(q_1, f) = \{ABCE, ABE\}$ *and* $SC(q_2) = \{ABC, ACD, AB, AC, A\}$.

Therefore, we have :

- $\lambda_S[\mathcal{Q}] = \{(ABCE, \{q_1\}), (ABC, \{q_2\}), (ACD, \{q_2\})\}$,
- $\lambda_G[\mathcal{Q}] = \{(B, \{q_1\}), (A, \{q_2\})\}$,
- $\mathcal{SC}(\mathcal{Q}, f) = \{ABCE, ABE, ACD, AC, A\}$, *and*
- $\lambda_{SC}[\mathcal{Q}, f] = \{(ABCE, 0.4), (ABE, 0.5), (ACD, 0.3), (AC, 0.5), (A, 0.8)\}$.

Then using Theorem 1, we know that $\{\lambda_S[\mathcal{Q}],\ \lambda_G[\mathcal{Q}],\ \lambda_{SC}[\mathcal{Q}, f]\}$ is an extended condensed representation of $ans(\mathcal{Q}, f)$, i.e. $\lambda_S[\mathcal{Q}],\ \lambda_G[\mathcal{Q}],\ \lambda_{SC}[\mathcal{Q}, f] \models_e ans(\mathcal{Q}, f)$. □

3.4 Further Improvements

Let f be a monotonic increasing measure function. In this section, we show how condensed representations can be made more concise when every mining query belongs to $\mathbb{Q}_f$.

Let $\mathcal{Q} = \{q_1, \ldots, q_n\}$ be a set of predicates in $\mathbb{Q}_f$. In what follows, we show how to define condensed representations of $sol(\mathcal{Q})$ that are *more concise* than $\{\lambda_S[\mathcal{Q}], \lambda_G[\mathcal{Q}]\}$. To this end, we define two partial orderings, denoted by $\leq_{\overline{\mathbb{A}}}$ and $\leq_{\overline{\mathbb{M}}}$ as follows : for all (φ_i, q_i) and (φ_j, q_j) in $\mathbb{L} \times \mathbb{Q}_f$:

$$\begin{array}{lll} (\varphi_i, q_i) \leq_{\overline{\mathbb{A}}} (\varphi_j, q_j) & \text{if} & \varphi_i \preceq \varphi_j \text{ and } \overline{a}_i \sqsubseteq \overline{a}_j \\ (\varphi_i, q_i) \leq_{\overline{\mathbb{M}}} (\varphi_j, q_j) & \text{if} & \varphi_i \preceq \varphi_j \text{ and } \overline{m}_j \sqsubseteq \overline{m}_i \end{array}$$

Then, we define the sets $\lambda^*_S[\mathcal{Q}]$, $\overline{\lambda}_S[\mathcal{Q}]$, $\lambda^*_G[\mathcal{Q}]$ and $\overline{\lambda}_G[\mathcal{Q}]$ by :

$$\begin{array}{lll} \lambda^*_S[\mathcal{Q}] = \{(\varphi, q) \mid \varphi \in \mathcal{S}(\mathcal{Q}) \wedge q \in \lambda_S[\mathcal{Q}](\varphi)\} & \text{and} & \overline{\lambda}_S[\mathcal{Q}] = min_{\leq_{\overline{\mathbb{A}}}}(\lambda^*_S[\mathcal{Q}]) \\ \lambda^*_G[\mathcal{Q}] = \{(\varphi, q) \mid \varphi \in \mathcal{G}(\mathcal{Q}) \wedge q \in \lambda_G[\mathcal{Q}](\varphi)\} & \text{and} & \overline{\lambda}_G[\mathcal{Q}] = max_{\leq_{\overline{\mathbb{M}}}}(\lambda^*_G[\mathcal{Q}]). \end{array}$$

Since $\overline{\mathbb{A}}_f$ and $\overline{\mathbb{M}}_f$ are totally ordered sets of predicates with respect to $\sqsubseteq$, we can easily prove the following lemma :

Lemma 3 *Given a measure function $f \in \mathbb{I}$, let $\mathcal{Q}$ be a set of queries in $\mathbb{Q}_f$.*
- *For all (φ_i, q_i) and (φ_j, q_j) in $\overline{\lambda}_S[\mathcal{Q}]$, if $\varphi_i = \varphi_j$, then $q_i = q_j$.*
- *For all (φ_i, q_i) and (φ_j, q_j) in $\overline{\lambda}_G[\mathcal{Q}]$, if $\varphi_i = \varphi_j$, then $q_i = q_j$.*

Proof : *Suppose there exist (φ_1, q_1) and (φ_2, q_2) in $\overline{\lambda}_S[\mathcal{Q}]$, with $\varphi_1 = \varphi_2 = \varphi$ and $q_1 \neq q_2$. This means that $\overline{\lambda}_S[\mathcal{Q}]$ can be written as follows :*
$\overline{\lambda}_S[\mathcal{Q}] = \{(\varphi, q_1), (\varphi, q_2), ...\}$, with $q_1 \neq q_2$.
But, $\overline{\lambda}_S[\mathcal{Q}] = min_{\leq_{\overline{\mathbb{A}}}}\lambda^*_S[\mathcal{Q}]$. Therefore, $\overline{a_1}$ and $\overline{a_2}$ are not comparable.
On the other hand, according to the definition, $\overline{a_1}$ and $\overline{a_2}$ are of the form : $f(\varphi) \geq \alpha$, meaning that they are comparable. This is a contradiction with the hypothesis that they are not comparable. So, for all (φ_i, q_i) and (φ_j, q_j) in $\overline{\lambda_S}[\mathcal{Q}]$, if $\varphi_i = \varphi_j$ then $q_i = q_j$.

Thus, the sets $\overline{\lambda}_S[\mathcal{Q}]$ and $\overline{\lambda}_G[\mathcal{Q}]$ are graphs of partial functions defined over $X \subseteq \mathbb{L}$.

Moreover, the following lemma states that, for every $q \in \mathcal{Q}$, $sol(q)$ can be computed based on $\overline{\lambda}_S[\mathcal{Q}]$ and $\overline{\lambda}_G[\mathcal{Q}]$, only.

Lemma 4 *Given a measure function $f \in \mathbb{I}$, let $\mathcal{Q} = \{q_1, \ldots, q_n\}$ be a set of mining queries $q_i \in \mathbb{Q}_f$ $(i = 1, \ldots, n)$. For every q in $\mathcal{Q}$, we have :*

$$\begin{array}{rl} sol(q) = \{\varphi \in sol(\widetilde{q}) \mid & (\exists (\varphi_i, q_i) \in \overline{\lambda}_S[\mathcal{Q}])((\varphi_i, q_i) \leq_{\overline{\mathbb{A}}} (\varphi, q)) \text{ and} \\ & (\exists (\varphi_j, q_j) \in \overline{\lambda}_G[\mathcal{Q}])((\varphi, q) \leq_{\overline{\mathbb{M}}} (\varphi_j, q_j))\}. \end{array}$$

Proof : *Let $\overline{X}(q)$ be the set defined by :*

$$\begin{array}{rl} \overline{X}(q) = \{\varphi \in sol(\widetilde{q}) \mid & (\exists (\varphi_i, q_i) \in \overline{\lambda}_S[\mathcal{Q}])((\varphi_i, q_i) \leq_{\overline{\mathbb{A}}} (\varphi, q) \text{ and} \\ & (\exists (\varphi_j, q_j) \in \overline{\lambda}_G[\mathcal{Q}])((\varphi, q) \leq_{\overline{\mathbb{M}}} (\varphi_j, q_j))\}. \end{array}$$

We first show that $\overline{X}(q) \subseteq sol(q)$. Let $\varphi \in \overline{X}(q)$. Then there exist $(\varphi_i, q_i) \in \overline{\lambda}_S[\mathcal{Q}]$ and $(\varphi_j, q_j) \in \overline{\lambda}_G[\mathcal{Q}]$ such that $(\varphi_i, q_i) \leq_{\overline{\mathbb{A}}} (\varphi, q)$ and $(\varphi, q) \leq_{\overline{\mathbb{M}}} (\varphi_j, q_j)$.
On one hand, we know that $q_i(\varphi_i) = true$. Thus, we have $\overline{a_i}(\varphi_i) = true$. It follows that $\overline{a}(\varphi_i) = true$ since $\overline{a_i} \sqsubseteq \overline{a}$, and so, $\overline{a}(\varphi) = true$ since $\varphi_i \preceq \varphi$ and $\overline{a}$ is anti-monotonic.

On the other hand, we know that $q_j(\varphi_j) = true$. *Thus, we have* $\overline{m_j}(\varphi_j) = true$. *It follows that* $\overline{m}(\varphi_j) = true$ *since* $\overline{m_j} \sqsubseteq \overline{m}$, *and so,* $\overline{m}(\varphi) = true$ *since* $\varphi \preceq \varphi_j$ *and* $\overline{m}$ *is monotonic. Therefore, we have* $\overline{q}(\varphi) = true$. *Since* $\varphi \in sol(\widetilde{q})$, *we have* $q(\varphi) = \overline{q}(\varphi) \wedge \widetilde{q}(\varphi) = true$, *which shows that* $\overline{X}(q) \subseteq sol(q)$.

Now, we show that $sol(q) \subseteq \overline{X}(q)$. *Let* $\varphi \in sol(q)$. *Then there exist* $\varphi_s \in S(q)$ *and* $\varphi_g \in G(q)$ *such that* $\varphi_s \preceq \varphi \preceq \varphi_g$. *Moreover, we have* $(\varphi_s, q) \in \lambda^*_S[\mathcal{Q}]$ *and* $(\varphi_g, q) \in \lambda^*_G[\mathcal{Q}]$.
Given the definitions of $\lambda^*_S[\mathcal{Q}]$ *and* $\lambda^*_G[\mathcal{Q}]$, *there exist* $(\varphi_i, q_i) \in \lambda^*_S[\mathcal{Q}]$ *and* $(\varphi_j, q_j) \in \lambda^*_G[\mathcal{Q}]$ *such that* $(\varphi_i, q_i) \leq_{\overline{\mathbb{A}}} (\varphi_s, q)$ *and* $(\varphi_g, q) \leq_{\overline{\mathbb{M}}} (\varphi_j, q_j)$. *Moreover, we have* $(\varphi_s, q) \leq_{\overline{\mathbb{A}}} (\varphi, q)$ *since* $\varphi_s \preceq \varphi$, *and* $(\varphi, q) \leq_{\overline{\mathbb{M}}} (\varphi_g, q)$ *since* $\varphi \preceq \varphi_g$.
Thus, $(\varphi_i, q_i) \leq_{\overline{\mathbb{A}}} (\varphi, q)$ *and* $(\varphi, q) \leq_{\overline{\mathbb{M}}} (\varphi_j, q_j)$. *As* $\varphi \in sol(q)$ *and as* $sol(q) \subseteq sol(\widetilde{q})$, *it follows that* $\varphi \in \overline{X}(q)$, *which shows that* $sol(q) \subseteq \overline{X}(q)$. *Thus, the proof is complete.*

As a consequence of lemmas 3 and 4, we have the following theorem :

Theorem 2 *Given a measure function* $f \in \mathbb{I}$, *let* $\mathcal{Q} = \{q_1, \ldots, q_n\}$ *be a set of mining queries* $q_i \in \mathbb{Q}_f$ $(i = 1, \ldots, n)$. *The set* $\{\overline{\lambda}_S[\mathcal{Q}],\ \overline{\lambda}_G[\mathcal{Q}]\}$ *is a condensed representation of* $sol(\mathcal{Q})$, *i.e.* $\overline{\lambda}_S[\mathcal{Q}],\ \overline{\lambda}_G[\mathcal{Q}] \models sol(\mathcal{Q})$. *Moreover, the condensed representation* $\{\overline{\lambda}_S[\mathcal{Q}],\ \overline{\lambda}_G[\mathcal{Q}]\}$ *is more concise than* $\{\lambda_S[\mathcal{Q}],\ \lambda_G[\mathcal{Q}]\}$.

PROOF : *Let us consider the function* F *defined by :*

$$F(\mathcal{X}_1, \mathcal{X}_2) = \{ \quad (\varphi, q) \in \mathbb{L} \times \mathbb{Q} \mid \varphi \in sol(\widetilde{q}) \text{ and } (\exists (\varphi_1, q_1) \in \mathcal{X}_1)((\varphi_1, q_1) \leq_{\overline{\mathbb{A}}} (\varphi, q)) \text{ and } (\exists (\varphi_2, q_2) \in \mathcal{X}_2)((\varphi, q) \leq_{\overline{\mathbb{M}}} (\varphi_2, q_2))\}$$

Using Lemma 4, we can easily see that $sol(\mathcal{Q}) = F(\overline{\lambda}_S[\mathcal{Q}], \overline{\lambda}_G[\mathcal{Q}])$. *Moreover,* F *is independent from the data set* Δ *since* $\preceq$ *and* $\sqsubseteq$ *do not depend on* Δ. *Finally, for every pair* (φ, q) *in* $\overline{\lambda}_S[\mathcal{Q}]$ *or* $\overline{\lambda}_G[\mathcal{Q}]$, *we know that* $(\varphi, q) \in sol(\mathcal{Q})$. *Thus, we have* $\pi_{\mathbb{L}}(\overline{\lambda}_S[\mathcal{Q}] \cup \overline{\lambda}_G[\mathcal{Q}]) \subseteq \pi_{\mathbb{L}}(sol(\mathcal{Q}))$, *which shows that* $\overline{\lambda}_S[\mathcal{Q}], \overline{\lambda}_G[\mathcal{Q}] \models sol(\mathcal{Q})$.
Moreover, $\pi_{\mathbb{L}}(\overline{\lambda}_S[\mathcal{Q}]) \subseteq \pi_{\mathbb{L}}(\lambda^*_S[\mathcal{Q}]) = \pi_{\mathbb{L}}(\lambda_S[\mathcal{Q}])$. *In the same way,* $\pi_{\mathbb{L}}(\overline{\lambda}_G[\mathcal{Q}]) \subseteq \pi_{\mathbb{L}}(\lambda^*_G[\mathcal{Q}]) = \pi_{\mathbb{L}}(\lambda_G[\mathcal{Q}])$. *Thus,* $\pi_{\mathbb{L}}(\overline{\lambda}_S[\mathcal{Q}]) \cup \pi_{\mathbb{L}}(\overline{\lambda}_G[\mathcal{Q}]) \subseteq \pi_{\mathbb{L}}(\lambda_S[\mathcal{Q}]) \cup \pi_{\mathbb{L}}(\lambda_G[\mathcal{Q}])$, *which completes the proof.*

Example 6 *Let* $\mathcal{Q} = \{q_1, q_2\}$ *be the set of simple mining queries* $q_i = m_i \wedge a_i$ *where* m_i *and* a_i $(i = 1, 2)$ *are defined in Example 1. We recall from Example 5 that :* $\lambda_S[\mathcal{Q}] = \{(ABCE, \{q_1\}), (ABC, \{q_2\}), (ACD, \{q_2\})\}$ *and* $\lambda_G[\mathcal{Q}] = \{(B, \{q_1\}, (A, \{q_2\})\}$.

Then, we have : $\lambda^*_S[\mathcal{Q}] = \{(ABCE, q_1), (ABC, q_2), (ACD, q_2)\}$ *and* $\lambda^*_G[\mathcal{Q}] = \{(B, q_1),\ (A, q_2)\}$. *Since* $ABC \subseteq ABCE$ $(ABCE \preceq ABC)$ *and* $\overline{a_1} \sqsubseteq \overline{a_2}$ $(\alpha(q_2) < \alpha(q_1))$, *we have* $(ABCE, q_1) \leq_{\overline{\mathbb{A}}} (ABC, q_2)$. *Thus, the pair* (ABC, q_2) *does not belong to* $\overline{\lambda}_S[\mathcal{Q}]$. *Hence, we have :* $\overline{\lambda}_S[\mathcal{Q}] = \{(ABCE, q_1), (ACD, q_2)\}$.

On the other hand, since the pairs (B, q_1) *and* (A, q_2) *are not comparable with respect to* $\leq_{\overline{\mathbb{M}}}$, *we have :* $\overline{\lambda}_G[\mathcal{Q}] = \lambda^*_G[\mathcal{Q}]$.

In conclusion, using Theorem 2, we can see that $\overline{\lambda}_S[\mathcal{Q}],\ \overline{\lambda}_G[\mathcal{Q}] \models sol(\mathcal{Q})$. □

4 Regeneration Algorithm

Let f *be a monotonic increasing measure function in* $\mathbb{I}$ *and* $\mathcal{Q} = \{q_1, \ldots, q_n\}$ *be a set of mining queries* $q_i \in \mathbb{Q}_f$ $(i = 1, \ldots, n)$.

In this section, for every $q \in \mathcal{Q}$, *we propose an algorithm to regenerate the answer of* (q, f) *from the extended condensed representation* $\{\overline{\lambda}_S[\mathcal{Q}],\ \overline{\lambda}_G[\mathcal{Q}],\ \lambda_{SC}[\mathcal{Q}, f]\}$ *of* $ans(\mathcal{Q}, f)$ *(without accessing the database* Δ*). Let* $q = a \wedge m$ *be a mining query in* $\mathcal{Q}$. *Based on Lemma 4, we have :*

$$sol(q) = \{\varphi \in sol(\widetilde{q}) \mid a^*(\varphi) = true \text{ and } m^*(\varphi) = true\}.$$

where a^* and m^* are the selection predicates defined by :

- *For every $\varphi \in \mathbb{L}$, $a^*(\varphi) = true$ if $(\exists(\varphi_i, q_i) \in \overline{\lambda}_S[\mathcal{Q}])((\varphi_i, q_i) \leq_{\overline{\mathbb{A}}} (\varphi, q))$, i.e if there exists (φ_i, q_i) in $\overline{\lambda}_S[\mathcal{Q}]$ such that $\varphi_i \preceq \varphi$ and $\overline{a_i} \sqsubseteq \overline{a}$.*
- *For every $\varphi \in \mathbb{L}$, $m^*(\varphi) = true$ if $(\exists(\varphi_j, q_j) \in \overline{\lambda}_G[\mathcal{Q}])((\varphi, q) \leq_{\overline{\mathbb{M}}} (\varphi_j, q_j))$, i.e if there exists (φ_j, q_j) in $\overline{\lambda}_G[\mathcal{Q}]$ such that $\varphi \preceq \varphi_j$ and $\overline{m_j} \sqsubseteq \overline{m}$.*

We can easily see that a^* is an anti-monotonic independent selection predicate, i.e. $a^* \in \widetilde{\mathbb{A}}$, whereas m^* is a monotonic independent selection predicate, i.e. $m^* \in \widetilde{\mathbb{M}}$. Moreover, we can show that for every $q \in \mathcal{Q}$, q is equivalent to $(\widetilde{a} \wedge a^*) \wedge (\widetilde{m} \wedge m^*)$, meaning that $sol(q) = sol((\widetilde{a} \wedge a^*) \wedge (\widetilde{m} \wedge m^*))$. Given this equivalence, we propose the following procedure to compute $ans(q, f)$. Each step of the procedure is detailed next.

Procedure *Regeneration*

Input :	*a mining query $q \in \mathcal{Q}$*
Output :	*the set $ans(q, f)$*
Use :	*the extended condensed representation of $ans(\mathcal{Q}, f)$*

1. *Compute $S(\widetilde{a})$ and $G(\widetilde{m})$ identifying the syntactic parts of q*
2. *Compute $S(a^*)$ and $G(m^*)$ (using Lemma 5 below)*
3. *Compute $S(\widetilde{a} \wedge a^*)$ and $G(\widetilde{m} \wedge m^*)$ (using Lemma 6 below)*
4. *Compute $S(q)$ and $G(q)$ (using Lemma 7 below)*
5. *Return* **RGN**$(G(q), S(q), \lambda_{SC}[\mathcal{Q}, f)])$

We now detail the different steps of this procedure. In order to compute the sets $S(a^*)$ and $G(m^*)$ (Step 2), we use the following lemma :

Lemma 5 *Let $\mathcal{Q} = \{q_1, \ldots, q_n\}$ be a set of mining queries $q_i \in \mathbb{Q}_f$ $(i = 1, \ldots, n)$. For every $q \in \mathcal{Q}$ such that $q = a \wedge m$ with $a \in \mathbb{A}$ and $m \in \mathbb{M}$, we have :*

$$\begin{aligned} S(a^*) &= min_{\preceq}\{\varphi_i \in \mathbb{L} \mid (\exists q_i \in \mathcal{Q})((\varphi_i, q_i) \in \overline{\lambda}_S[\mathcal{Q}] \text{ and } \overline{a_i} \sqsubseteq \overline{a})\} \\ G(m^*) &= max_{\preceq}\{\varphi_i \in \mathbb{L} \mid (\exists q_i \in \mathcal{Q})((\varphi_i, q_i) \in \overline{\lambda}_G[\mathcal{Q}] \text{ and } \overline{m_i} \sqsubseteq \overline{m})\} \end{aligned}$$

Proof : *Given a query $q = m \wedge a$ in $\mathcal{Q}$, let $X(a^*) = \{\varphi_i \in \mathbb{L} \mid (\exists q_i \in \mathcal{Q})((\varphi_i, q_i) \in \overline{\lambda_S}[\mathcal{Q}] \text{ and } \overline{a_i} \sqsubseteq \overline{a})\}$ and $Y(a^*) = min_{\preceq} X(a^*)$. We have to prove that $S(a^*) = Y(a^*)$.*

We first show that $Y(a^*) \subseteq S(a^*)$. Let $\varphi \in Y(a^*)$. Then there exists $q_i \in \mathcal{Q}$ such that $(\varphi, q_i) \in \overline{\lambda_S}[\mathcal{Q}]$ and $\overline{a_i} \sqsubseteq \overline{a}$. Thus, we have $(\varphi, q_i) \in \overline{\lambda_S}[\mathcal{Q}]$ and $(\varphi, q_i) \leq_{\overline{\mathbb{A}}} (\varphi, q)$, which shows that $\varphi \in sol(a^*)$. Assume now that $\varphi \notin S(a^*)$. Then, there exists $\varphi' \in sol(a^*)$ such that $\varphi' \prec \varphi$. On the other hand, since $\varphi' \in sol(a^*)$, there exists $(\varphi'_i, q'_i) \in \overline{\lambda_S}[\mathcal{Q}]$ such that $(\varphi'_i, q'_i) \leq_{\overline{\mathbb{A}}} (\varphi', q)$. It follows that $(\varphi'_i, q'_i) \leq_{\overline{\mathbb{A}}} (\varphi'_i, q)$. Therefore, we have $\varphi'_i \in X(a^*)$ and $\varphi'_i \preceq \varphi' \prec \varphi$, which contradicts the fact that φ is minimal in $X(a^*)$ with respect to $\preceq$. Hence, we have $\varphi \in S(a^*)$ and $Y(a^*) \subseteq S(a^*)$.

Now, we show that $S(a^*) \subseteq Y(a^*)$. Let $\varphi \in S(a^*)$. Then there exists $(\varphi_i, q_i) \in \overline{\lambda_S}[\mathcal{Q}]$ such that $(\varphi_i, q_i) \leq_{\overline{\mathbb{A}}} (\varphi, q)$, which shows that $\varphi \in X(a^*)$. Assume that $\varphi \notin Y(a^*)$. Then, there exists $\varphi' \in X(a^*)$ such that $\varphi' \prec \varphi$. Since $\varphi' \in X(a^*)$, there exists $q_i \in \mathcal{Q}$ such that $(\varphi', q_i) \in \overline{\lambda_S}[\mathcal{Q}]$ and $\overline{a_i} \sqsubseteq \overline{a}$. It follows that $(\varphi', q_i) \leq_{\overline{\mathbb{A}}} (\varphi', q)$. Thus, we have $\varphi' \in sol(a^*)$ and $\varphi' \preceq \varphi$, which contradicts the fact that φ is minimal in $sol(a^*)$ with respect to $\preceq$. Hence, we have $\varphi \in Y(a^*)$ and $S(a^*) \subseteq Y(a^*)$, which completes the proof that $S(a^*) = min_{\preceq}\{\varphi_i \in \mathbb{L} \mid (\exists q_i \in \mathcal{Q})((\varphi_i, q_i) \in \overline{\lambda_S}[\mathcal{Q}] \text{ and } \overline{a_i} \sqsubseteq \overline{a})\}$.

The proof that $G(m^*) = max_{\preceq}\{\varphi_i \in \mathcal{L} \mid (\exists q_i \in \mathcal{Q})((\varphi_i, q_i) \in \overline{\lambda_G}[\mathcal{Q}] \text{ and } \overline{m_i} \sqsubseteq \overline{m})\}$ uses similar arguments, and thus is omitted.

Steps 3 and 4 of Procedure Regeneration *are implemented based on the following definition and lemmas.*

Definition 11 *Let φ_1 and φ_2 be two patterns in $\mathbb{L}$.*

- *The* most general specializations *of φ_1 and φ_2, denoted by $mgs(\varphi_1, \varphi_2)$, is defined by : $mgs(\varphi_1, \varphi_2) = max_{\preceq}\{\varphi \in \mathbb{L} \mid \varphi \preceq \varphi_1 \text{ and } \varphi \preceq \varphi_2\}$.*
- *The* most specialized generalizations *of φ_1 and φ_2, denoted by $msg(\varphi_1, \varphi_2)$, is defined by : $msg(\varphi_1, \varphi_2) = min_{\preceq}\{\varphi \in \mathbb{L} \mid \varphi_1 \preceq \varphi \text{ and } \varphi_2 \preceq \varphi\}$.*

In the literature [43], the set $mgs(\varphi_1, \varphi_2)$ is also called the quasi-meet *of φ_1 and φ_2, whereas the set $msg(\varphi_1, \varphi_2)$ is called the* quasi-join *of φ_1 and φ_2. Note that if $\mathbb{L}$ is a lattice, then $mgs(\varphi_1, \varphi_2)$ and $msg(\varphi_1, \varphi_2)$ are singletons, i.e. the quasi-meets and quasi-joins correspond to meet and join.*

The following lemma is introduced in [23].

Lemma 6 *Let a_1 and a_2 be two mining queries in $\mathbb{A}$, and let m_1 and m_2 be two mining queries in $\mathbb{M}$. We have :*

- *$S(a_1 \wedge a_2) = min_{\preceq}\{\varphi \in msg(\varphi_1, \varphi_2) \mid \varphi_1 \in S(a_1) \text{ and } \varphi_2 \in S(a_2)\}$, and*
- *$G(m_1 \wedge m_2) = max_{\preceq}\{\varphi \in mgs(\varphi_1, \varphi_2) \mid \varphi_1 \in G(m_1) \text{ and } \varphi_2 \in G(m_2)\}$.*

Proof : *Given two mining queries a_1 and a_2 in $\mathbb{A}$, let $X(a_1 \wedge a_2) = \{\varphi \in msg(\varphi_1, \varphi_2) \mid \varphi_1 \in S(a_1) \text{ and } \varphi_2 \in S(a_2)\}$ and $Y(a_1 \wedge a_2) = min_{\preceq}X(a_1 \wedge a_2)$. We have to prove that $S(a_1 \wedge a_2) = Y(a_1 \wedge a_2)$.*

We first show that $Y(a_1 \wedge a_2) \subseteq S(a_1 \wedge a_2)$. Let $\varphi \in Y(a_1 \wedge a_2)$. Then there exist $\varphi_1 \in S(a_1)$ and $\varphi_2 \in S(a_2)$ such that $\varphi \in msg(\varphi_1, \varphi_2)$. Thus, we have $\varphi_1 \preceq \varphi$, $\varphi_2 \preceq \varphi$. Moreover, a_1 and a_2 are anti-monotonic. Therefore, $\varphi \in sol(a_1)$ and $\varphi \in sol(a_2)$, which shows that $\varphi \in sol(a_1 \wedge a_2)$. Assume now that $\varphi \notin S(a_1 \wedge a_2)$. Then, there exists $\varphi' \in sol(a_1 \wedge a_2)$ such that $\varphi' \prec \varphi$. Moreover, since $\varphi' \in sol(a_1 \wedge a_2) = sol(a_1) \cap sol(a_2)$, there exists $\varphi'_1 \in S(a_1)$ and $\varphi'_2 \in S(a_2)$ such that $\varphi'_1 \preceq \varphi'$ and $\varphi'_2 \preceq \varphi'$. Let $\varphi^* \in msg(\varphi'_1, \varphi'_2)$ [1]. Thus, we have $\varphi^* \in X(a_1 \wedge a_2)$ and $\varphi^* \preceq \varphi' \prec \varphi$ which contradicts the fact that φ is minimal in $X(a_1 \wedge a_2)$ with respect to $\preceq$. Hence, we have $\varphi \in S(a_1 \wedge a_2)$ and thus, $Y(a_1 \wedge a_2) \subseteq S(a_1 \wedge a_2)$.

Now, we show that $S(a_1 \wedge a_2) \subseteq Y(a_1 \wedge a_2)$. Let $\varphi \in S(a_1 \wedge a_2)$. We know that $\varphi \in sol(a_1)$ and $\varphi \in sol(a_2)$. Thus, there exist $\varphi_1 \in S(a_1)$ and $\varphi_2 \in S(a_2)$ such that $\varphi_1 \preceq \varphi$ and $\varphi_2 \preceq \varphi$. If $\varphi \notin msg(\varphi_1, \varphi_2)$, then there exists $\varphi' \in \mathbb{L}$ such that $\varphi' \prec \varphi$, $\varphi_1 \preceq \varphi'$ and $\varphi_2 \preceq \varphi'$. Thus, we have $\varphi' \in sol(a_1 \wedge a_2)$, which contradicts the fact that φ is minimal in $sol(a_1 \wedge a_2)$ with respect to $\preceq$. Hence, we have $\varphi \in msg(\varphi_1, \varphi_2)$ and $\varphi \in X(a_1 \wedge a_2)$. Assume now that $\varphi \notin Y(a_1 \wedge a_2)$. Then, there exist $\varphi'_1 \in S(a_1)$, $\varphi'_2 \in S(a_2)$ and $\varphi' \in msg(\varphi'_1, \varphi'_2)$ such that $\varphi' \prec \varphi$. Since $\varphi' \in msg(\varphi'_1, \varphi'_2)$, we have $\varphi'_1 \preceq \varphi'$ and $\varphi'_2 \preceq \varphi'$. Moreover, a_1 and a_2 are anti-monotonic. Thus, we have $\varphi' \in sol(a_1 \wedge a_2)$ and $\varphi' \prec \varphi$, which contradicts the fact that φ is minimal in $sol(a_1 \wedge a_2)$ with respect to $\preceq$. Hence, we have $\varphi \in Y(a_1 \wedge a_2)$ and thus, $S(a_1 \wedge a_2) \subseteq Y(a_1 \wedge a_2)$. This completes the proof that $S(a_1 \wedge a_2) = Y(a_1 \wedge a_2)$.

The proof that $G(m_1 \wedge m_2) = \{\varphi \in mgs(\varphi_1, \varphi_2) \mid \varphi_1 \in G(m_1) \text{ and } \varphi_2 \in G(m_2)\}$ uses similar arguments, and thus is omitted.

Lemma 7 *Let $q = a \wedge m$ be a mining query with $a \in \mathbb{A}$ and $m \in \mathbb{M}$. We have :*

- *$S(q) = \{\varphi \in S(a) \mid (\exists \varphi' \in G(m))(\varphi \preceq \varphi')\}$, and*
- *$G(q) = \{\varphi \in G(m) \mid (\exists \varphi' \in S(a))(\varphi' \preceq \varphi)\}$.*

Proof : *Given a mining query $q = m \wedge a$ with $m \in \mathbb{M}$ and $a \in \mathbb{A}$, let $X(q) = \{\varphi \in S(a) \mid (\exists \varphi' \in G(m))(\varphi \preceq \varphi')\}$. We have to prove that $X(q) = sol(q)$.*

We first show that $S(q) \subseteq X(q)$. Let $\varphi \in S(q)$. We have $\varphi \in sol(q) \subseteq sol(a)$. Assume that $\varphi \notin S(a)$. Then, there exists $\varphi' \in sol(a)$ such that $\varphi' \prec \varphi$. Moreover, $\varphi' \in sol(m)$ since $\varphi' \prec \varphi$,

[1] In this paper, we assume that there exist a minimal and a maximal element in $\mathbb{L}$. Therefore, $msg(\varphi'_1, \varphi'_2)$ can not be empty and φ^* exists

$\varphi \in sol(m)$ *and* m *is monotonic. Thus, we have* $\varphi' \in sol(a) \cap sol(m) = sol(q)$ *and* $\varphi' \prec \varphi$*, which contradicts the fact that* φ *is minimal in* $sol(q)$ *with respect to* $\preceq$*. Hence, we have* $\varphi \in S(a)$*. On the other hand, we have* $\varphi \in sol(m)$*. Therefore, there exists* $\varphi' \in G(m)$ *such that* $\varphi \preceq \varphi'$*, which completes the proof that* $\varphi \in X(q)$ *and* $S(q) \subseteq G(q)$*.*

Now, we show that $X(q) \subseteq S(q)$*. Let* $\varphi \in X(q)$*. Then, we have* $\varphi \in S(a)$ *and there exists* $\varphi' \in G(m)$ *such that* $\varphi \preceq \varphi'$*. Therefore, we have* $\varphi \in sol(m) \cap sol(a) = sol(q)$*. Assume now that* $\varphi \notin S(q)$*. Then, there exists* $\varphi' \in sol(q)$ *such that* $\varphi' \prec \varphi$*. Since* $\varphi' \in sol(q) \subseteq sol(a)$*, there exists* $\varphi_a \in S(a)$ *such that* $\varphi_a \preceq \varphi'$*. Thus, we have* $\varphi_a \preceq \varphi' \prec \varphi$ *and* $\varphi_a \in sol(a)$*, which contradicts the fact that* φ *is minimal in* $sol(a)$ *with respect to* $\prec$*. Hence, we have* $\varphi \in S(q)$ *and* $X(q) \subseteq S(q)$*, which completes the proof that* $X(q) = sol(q)$*.*

The proof that $G(q) = \{\varphi \in G(m) \mid (\exists \varphi' \in S(a))(\varphi' \preceq \varphi)\}$ *uses similar arguments, and thus is omitted.*

In procedure Regeneration*, Lemma 6 is used at Step 3, whereas Lemma 7 is used at Step 4.*

Finally, we present the Function RGN *used at Step 5 of Procedure* Regeneration*. This function uses the operator of specialization* ρ *defined for every pattern* $\varphi \in \mathbb{L}$ *by :*

$$\rho(\varphi) = max_{\preceq}\{\varphi' \in \mathbb{L} \mid \varphi' \prec \varphi\}.$$

We can easily see that for every extended mining query (q, f)*, function RGN computes level by level the answer of* (q, f)*. Let* φ *be a pattern in* X_k *with* $k \geq 1$*. There exists a pattern* $\varphi_0 \in X_0$ *such that* $\varphi \in \rho^k(\varphi_0)$*. Therefore,* φ *is more specific than a pattern* $\varphi_0 \in G(q)$*. Moreover, there exists a pattern* $\varphi_y \in Y$ *such that* $\varphi_y \preceq \varphi$ *(see Step 7). Thus,* φ *is more general than a pattern* $\varphi_y \in S(q)$*. Finally, we have* $\varphi \in sol(q)$ *and we can compute* $f(\varphi)$ *at Step 4 using Lemma 2.*

Function $RGN(X, Y, Z)$

1. *Let* $M = \emptyset$*,* $X_0 = X$ *and* $k = 0$
2. **While** $(X_k \neq \emptyset)$ **do begin**
3. **For** *every pattern* $\varphi \in X_k$ **do begin**
4. $\alpha(\varphi) = max_{\preceq}\{f(\varphi') \mid (\varphi', f(\varphi')) \in Z \wedge \varphi' \preceq \varphi\}$
5. $M = M \cup \{(\varphi, \alpha(\varphi))\}$
6. **End for**
7. $X_{k+1} = \bigcup_{\varphi \in X_k}\{\varphi' \in \rho(\varphi) \mid (\exists \varphi_y \in Y)(\varphi_y \preceq \varphi')\}$
8. $k = k + 1$
9. **End while**
10. **Return** M

Example 7 *Let* $\mathcal{Q} = \{q_1, q_2\}$ *be the set of simple mining queries defined in Example 1. We recall from Examples 5 and 6 that :*

- $\overline{\lambda}_S[\mathcal{Q}] = \{(ABCE, q_1), (ACD, q_2)\}$, $\overline{\lambda}_G[\mathcal{Q}] = \{(A, q_2), (B, q_1)\}$*, and*
- $\lambda_{SC}[\mathcal{Q}, f] = \{(ABCE, 0.4), (ABE, 0.5), (ACD, 0.3), (AC, 0.5), (A, 0.8)\}$.

Moreover, we know that $\mathcal{C}[\mathcal{Q}, f] = \{\overline{\lambda}_S[\mathcal{Q}], \overline{\lambda}[\mathcal{Q}], \lambda_{SC}[\mathcal{Q}, f]\}$ *is an extended condensed representation of* $ans(\mathcal{Q}, f)$*. In the following, we show how we can regenerate* $ans(q_1, f)$ *from* $\mathcal{C}[\mathcal{Q}, f]$ *using procedure Regeneration.*

1. *At first, we know that* $G(\widetilde{m_1}) = \{B\}$ *and* $S(\widetilde{a_1}) = \{ABCE\}$*.*
2. *Since* $\overline{a_1} \sqsubseteq \overline{a_2}$ *and* $\overline{m_2} \sqsubseteq \overline{m_1}$*, we have :*

$$\begin{array}{rcl} S(a_1^*) & = & min_{\preceq}\{\varphi_i \in \mathbb{L} \mid (\exists q_i \in \mathcal{Q})((\varphi_i, q_i) \in \overline{\lambda}_S[\mathcal{Q}]) \text{ and } \overline{a_i} \sqsubseteq \overline{a_1})\} \\ & = & min_{\preceq}\{ABCE\} = \{ABCE\} \\ G(m_1^*) & = & max_{\preceq}\{\varphi_i \in \mathbb{L} \mid (\exists q_i \in \mathcal{Q})((\varphi_i, q_i) \in \overline{\lambda}_G[\mathcal{Q}]) \text{ and } \overline{m_i} \sqsubseteq \overline{m_1})\} \\ & = & min_{\preceq}\{A, B\} = \{A, B\} \end{array}$$

3. *In the context of our Running Example 1, the set of patterns $\mathbb{L}$ is a lattice, and for every pair $(\varphi_1, \varphi_2) \in \mathbb{L}$, we have $msg(\varphi_1, \varphi_2) = \{\varphi_1 \cap \varphi_2\}$ and $mgs(\varphi_1, \varphi_2) = \{\varphi_1 \cup \varphi_2\}$. Therefore, we have :*

$$\begin{array}{rcl} S(a_1) = S(\widetilde{a_1} \wedge a_1^*) & = & min_{\preceq}\{\varphi_1 \cap \varphi_2 \mid \varphi_1 \in S(\widetilde{a_1}) \text{ and } \varphi_2 \in S(a_1^*)\} \\ & = & \{ABCE\} \text{ and} \\ G(m_1) = G(\widetilde{m_1} \wedge m_1^*) & = & max_{\preceq}\{\varphi_1 \cup \varphi_2 \mid \varphi_1 \in G(\widetilde{m_1}) \text{ and } \varphi_2 \in G(m_1^*)\} \\ & = & max_{\preceq}\{B, AB\} = \{B\} \end{array}$$

4. *Finally, since $B \subseteq ABCE$, we can see that $S(q_1) = S(a_1)$ and $G(q_1) = G(m_1)$.*
5. *Now, we regenerate $ans(q_1, f)$ using function RGN, i.e. $ans(q_1, f) = RGN(G(q_1), S(q_1), \lambda_{SC}[\mathcal{Q}, f]))$. Due to a lack of space, the execution of this function is not detailed here.* □

5 Query Optimization using Condensed Representations

In this section, we show how extended condensed representations can be used to optimize the iterative computation of the answer of mining queries. In our framework, this problem can be stated as follows : given a data set Δ, a set $\mathcal{Q} = \{q_1, \ldots, q_n\}$ of mining queries and f a monotonic increasing measure function in $\mathbb{I}$, how to optimize the computation of a new extended mining query (q, f) using the extended condensed representations of $ans(\mathcal{Q}, f)$?

In the following, we propose a solution of this general problem in the case where every mining query q belongs to $\mathbb{Q}_f$. Given a pattern $\varphi \in \mathbb{L}$, this means that if we know the value of $f(\varphi, \Delta)$, then for every mining query q, the evaluation of $q(\varphi, \Delta)$ can be done without accessing the data set Δ.

In order to optimize the computation of $ans(q, f)$, we propose to proceed in two steps :

1. *In a first step, we start by computing for each query $q_i \in \mathcal{Q}$, the intersection between $ans(q, f)$ and $ans(q_i, f)$. This step is realized without accessing the data set Δ. Indeed, its result $MC = \bigcup_{q_i \in \mathcal{Q}}(ans(q, f) \cap ans(q_i, f))$ is computed using only the extended condensed representations of $ans(\mathcal{Q}, f)$.*
2. *In a second step, we compute $ans(q, f)$ using any level wise algorithm modified as follows. In the* evaluation step *of candidate patterns, we start by searching those candidate patterns φ that belong to MC. Indeed, if a candidate pattern φ belongs to MC, then we already know $f(\varphi, \Delta)$. Therefore, it is not necessary to access the data set to evaluate if it is interesting or not.*

In the following, we present the functions INTER *and* GEN *used to compute $MC = \bigcup_{q_i \in \mathcal{Q}}(ans(q, f) \cap ans(q_i, f))$ from the extended condensed representations of $ans(\mathcal{Q}, f)$.*

Function *INTER*

Input :	*a new mining query q in $\mathbb{Q}_f$*
Output :	*the set $MC = \bigcup_{q_i \in \mathcal{Q}}(ans(q, f) \cap ans(q_i, f))$*
Use :	*the extended condensed representation of $ans(\mathcal{Q}, f)$*

1. **Let** $MC = \emptyset$
2. *Compute $S(\widetilde{a})$ and $G(\widetilde{m})$ identifying the syntactic part of q*
3. **For** *every query $q_i \in \mathcal{Q}$ such that $(sol(\widetilde{q}) \cap sol(\widetilde{q_i}) \neq \emptyset)$* **do begin**
4. **If** $([\alpha(q), \beta(q))] \cap [\alpha(q_i), \beta(q_i)] \neq \emptyset)$ **then begin**
5. *Regenerate $S(a_i)$ and $G(m_i)$ from $\overline{\lambda}_S[\mathcal{Q}]$ and $\overline{\lambda}_G[\mathcal{Q}]$*
6. *Compute $S(\widetilde{a} \wedge a_i)$ and $G(\widetilde{m} \wedge m_i)$ (using Lemma 6)*
7. $INT = \mathbf{GEN}(\alpha(q), \beta(q), G(\widetilde{m} \wedge m_i), S(\widetilde{a} \wedge a_i), \lambda_{SC}[\mathcal{Q}, f])$
8. $MC = MC \cup INT$
9. **End if**
10. **End for**

In function INTER*, it is easy to see that if* $sol(\widetilde{q}) \cap sol(\widetilde{q_i}) = \emptyset$ *(see Step 3) or* $[\alpha(q), \beta(q)] \cap [\alpha(q_i), \beta(q_i)] = \emptyset$ *(see Step 4), then* $ans(q, f) \cap ans(q_i, f) = \emptyset$*. If not, we compute* $ans(q, f) \cap ans(q_i, f)$ *as follows. At Step 5, the computation of* $S(a_i)$ *and* $G(m_i)$ *from* $\overline{\lambda}_S[\mathcal{Q}]$ *and* $\overline{\lambda}_G[\mathcal{Q}]$ *is done in the same way as in Procedure* Regeneration*. Then, the computation of* $S(\widetilde{a} \wedge a_i)$ *and* $G(\widetilde{m} \wedge m_i)$ *is realized using Lemma 6 presented in Section 4.*

Function $GEN(\alpha, \beta, X, Y, Z)$

1. *Let* $M = \emptyset$, $X_0 = X$ *and* $k = 0$
2. **While** $(X_k \neq \emptyset)$ **do begin**
3. *Let* $F_k = \emptyset$
4. **For** *every pattern* $\varphi \in X_k$ **do begin**
5. $\alpha(\varphi) = max_{\preceq}\{f(\varphi') \mid (\varphi', f(\varphi')) \in Z \wedge \varphi' \preceq \varphi\}$
6. **If** $(\alpha \leq \alpha(\varphi))$ **then begin**
7. $F_k = F_k \cup \{\varphi\}$
8. **If** $(\alpha(\varphi) \leq \beta)$ **then** $M = M \cup \{(\varphi, \alpha(\varphi))\}$
9. **End if**
10. **End for**
11. $X_{k+1} = \bigcup_{\varphi \in F_k}\{\varphi' \in \rho(\varphi) \mid (\exists \varphi_y \in Y)(\varphi_y \preceq \varphi')\}$
12. $X_{k+1} = X_{k+1} \setminus \{\varphi' \in X_{k+1} \mid (\exists \varphi \in X_k \setminus F_k)(\varphi' \in \rho(\varphi))\}$
13. $k = k + 1$
14. **End while**
15. **Return** M

We now emphasize the differences between functions RGN *and* GEN*. In function RGN, let us consider that* $X = G(q_i)$ *and* $Y = S(q_i)$ *where* $q_i \in \mathcal{Q}$*, and let* φ *be a pattern in* X_k*. If there exists a pattern* $\varphi_y \in Y$ *such that* $\varphi_y \preceq \varphi$*, then* φ *is interesting with respect to* q_i*, i.e.* $q_i(\varphi, \Delta) = true$*.*

In function GEN, let us consider that $X = G(q_i \wedge \widetilde{q})$ *and* $Y = S(q_i \wedge \widetilde{q})$ *where* $q_i \in \mathcal{Q}$*, and let* φ *be a pattern in* X_k*. If there exists a pattern* $\varphi_y \in Y$ *such that* $\varphi_y \preceq \varphi$*, then we know that* φ *is interesting with respect to* $(q_i \wedge \widetilde{q})$*, i.e.* $[q_i \wedge \widetilde{q}](\varphi, \Delta) = true$*. Therefore, in order to prove that* $\varphi \in sol(q_i, \Delta) \cap sol(q, \Delta)$*, we only have to test if* $f(\varphi, \Delta) = \alpha(\varphi)$ *belongs to the interval* $[\alpha, \beta]$ *where* $\alpha = \alpha(q)$ *and* $\beta = \beta(q)$ *(see steps 6 and 8). Finally, if* $\alpha(\varphi) < \alpha$*, we know that no specialization of* φ *is interesting with respect to* q*. It follows that at steps 6, 10 and 11, we only consider specialization of* φ *if* $\alpha \leq \alpha(\varphi)$*.*

Example 8 *Let* q_1 *and* q_2 *be the simple mining queries of Example 1. Assume that we have already computed the extended condensed representation of* $(\mathcal{Q}, f)$ *where* $\mathcal{Q} = \{q_2\}$*. In this example, we show how this extended condensed representation can be used to optimize the computation of* $ans(q_1, f)$*. Let* $\mathcal{Q} = \{q_2\}$*. In this case, we have :*

- $\overline{\lambda}_S[\mathcal{Q}] = \{(ABC, q_2), (ACD, q_2)\}$, $\overline{\lambda}_G[\mathcal{Q}] = \{(A, q_2)\}$ *and*
- $\lambda_{SC}[\mathcal{Q}, f] = \{(ABC, 0.4), (ACD, 0.3), (AB, 0.5), (AC, 0.5), (A, 0.8)\}$.

Now, we show how we can use function $INTER$ *to compute* $ans(q_1, f) \cap ans(q_2, f)$*. At first, using* $\overline{\lambda}_S[\mathcal{Q}]$ *and* $\overline{\lambda}_G[\mathcal{Q}]$*, we compute* $S(a_2) = \{ABC, ACD\}$ *and* $G(m_2) = \{A\}$*. On the other hand, we know that :* $S(\widetilde{a_1}) = \{ABCE\}$ *and* $G(\widetilde{m_1}) = \{B\}$*. Therefore, we have :*

$$\begin{array}{lcl} S(\widetilde{a_1} \wedge a_2) & = & min_{\preceq}\{\varphi \cap \varphi' \mid \varphi \in S(\widetilde{a_1}) \text{ and } \varphi' \in S(a_2)\} \\ & = & min_{\preceq}\{ABCE \cap ABC, ABCE \cap ACD\} \\ & = & min_{\preceq}\{ABC, AC\} = \{ABC\} \text{ and} \\ G(\widetilde{m_1} \wedge m_2) & = & max_{\preceq}\{\varphi \cup \varphi' \mid \varphi \in G(\widetilde{m_1}) \text{ and } \varphi' \in G(m_2)\} \\ & = & max_{\preceq}\{A \cup B\} = \{AB\} \end{array}$$

Now, we call function GEN with $\alpha(q_1) = 0.4$*,* $\beta(q_1) = 1.0$*,* $X = \{AB\}$*,* $Y = \{ABC\}$ *and* $Z = \{(ABC, 0.4), (ACD, 0.3), (AB, 0.5), (AC, 0.5), (A, 0.8)\}$.

At the first iteration, we have $k = 0$, $X_0 = \{AB\}$ *and* $F_0 = \emptyset$. *Let* $\varphi = \{AB\}$. *We compute* $\alpha(AB) = max_{\leq}\{f(ABC), f(AB)\} = 0.5$, $F_0 = \{AB\}$ *and* $M = \{(AB, 0.5)\}$.
At the second iteration, we have $k = 1$ *and* $X_1 = \{ABC\}$. *Let* $F_1 = \emptyset$ *and* $\varphi = \{ABC\}$. *We compute* $\alpha(ABC) = max_{\leq}\{f(ABC)\} = 0.4$, $F_1 = \{ABC\}$ *and* $M = M \cup \{(AB, 0.5)\} = \{(ABC, 0.4), (AB, 0.5)\}$. *Then, we have* $X_2 = \emptyset$ *which completes the execution of function GEN.*

As a consequence, when we compute $ans(q_1, f)$ *using the extended condensed representation of* $ans(\mathcal{Q}, f)$, *it is not necessary to evaluate the supports of patterns ABC and AB in MC. Indeed, we show that their supports can be inferred from the extended condensed representation of* $ans(\mathcal{Q}, f)$ *without accessing the data set.* □

6 Update of Extended Condensed Representation

Given a measure function f *in* $\mathbb{I}$, *let* $\mathcal{Q}$ *be a set of mining queries in* $\mathbb{Q}_f$. *In this section, we show how to efficiently update the extended condensed representation* $\{\overline{\lambda}_S[\mathcal{Q}],\ \overline{\lambda}_G[\mathcal{Q}],\ \lambda_{SC}[\mathcal{Q}], f\}$ *of* $ans(\mathcal{Q}, f)$ *so as to obtain an extended condensed representation of* $ans(\mathcal{Q} \cup \{q\}, f)$ *where* q *is a new mining query in* $\mathbb{Q}_f$.

6.1 Update of Set of Minimal Patterns

Given a measure function f *in* $\mathbb{I}$, *let* $\mathcal{Q}$ *be a set of mining queries in* $\mathbb{Q}_f$. *The function Update*$[S]$ *shows how to update a set* $\overline{\lambda}_S[\mathcal{Q}]$ *of minimal patterns when a new mining query* $q \in \mathbb{Q}_f$ *is inserted in* $\mathcal{Q}$. *This function computes the set* $\overline{\lambda}_S[\mathcal{Q} \cup \{q\}]$ *from* $S(q)$ *and* $\overline{\lambda}_S[\mathcal{Q}]$. *By definition, we have :*

$$\overline{\lambda}_S[\mathcal{Q} \cup \{q\}] = min_{\leq_{\overline{\mathbb{A}}}}(\overline{\lambda}_S[\mathcal{Q}] \cup \{(\varphi, q) \mid \varphi \in S(q)\}).$$

Let φ *be a pattern in* $S(q)$. *At Step 3, we test if there exists a pair* $(\varphi_i, q_i) \in \overline{\lambda}_S[\mathcal{Q}]$ *such that* $(\varphi_i, q_i) \leq_{\overline{\mathbb{A}}} (\varphi, q)$, *i.e.* $\varphi_i \preceq \varphi$ *and* $\alpha(q) \leq \alpha(q_i)$. *If not (see Step 4), then the pair* (φ, q) *has to be inserted in* $\overline{\lambda}_S[\mathcal{Q}]$ *(at Step 5, this pair is temporarily stored in variable Ins). Then, if* (φ, q) *is inserted in* $\overline{\lambda}_S[\mathcal{Q}]$, *we test if there exist some pairs* $(\varphi_i, q_i) \in \overline{\lambda}_S[\mathcal{Q}]$ *such that* $(\varphi, q) \leq_{\overline{\mathbb{A}}} (\varphi_i, q_i)$. *Indeed, these pairs must be deleted from* $\overline{\lambda}_S[\mathcal{Q}]$ *(see Step 8 and 11). At Step 6, we first consider the pairs* $(\varphi_i, q_i) \in \overline{\lambda}_S[\mathcal{Q}]$ *such that* $\varphi_i = \varphi$. *Then, we use the following lemma to optimize the number of comparisons between* (φ, q) *and the pairs* $(\varphi_i, q_i) \in \overline{\lambda}_S[\mathcal{Q}]$.

Lemma 8 *Let* q_1 *and* q_2 *be two queries in* $\mathbb{Q}_f$. *Let* $\varphi_1 \in S(q_1)$ *and* $\varphi_2 \in S(q_2)$. *If* $(\varphi_1, q_1) \leq_{\overline{\mathbb{A}}} (\varphi_2, q_2)$ *and* $\varphi_1 \neq \varphi_2$, *then* $\varphi_1 \notin sol(\widetilde{q_2})$.

Proof : *Given two mining queries* q_1 *and* q_2 *in* $\mathbb{Q}_f$, *let* $\varphi_1 \in S(q_1)$ *and* $\varphi_2 \in S(q_2)$ *such that* $(\varphi_1, q_1) \leq_{\overline{\mathbb{A}}} (\varphi_2, q_2)$ *and* $\varphi_1 \neq \varphi_2$. *There exist* $\alpha_1, \alpha_2, \beta_1, \beta_2$ *in* $\Re$, $\widetilde{q_1}$ *and* $\widetilde{q_2}$ *in* $\widetilde{\mathbb{Q}}$ *such that for all pattern* $\varphi \in \mathbb{L}$ *:*

$$q_1(\varphi) = \widetilde{q_1}(\varphi) \wedge (\alpha_1 \leq f(\varphi) \leq \beta_1)$$
$$q_2(\varphi) = \widetilde{q_2}(\varphi) \wedge (\alpha_2 \leq f(\varphi) \leq \beta_2)$$

Assume that $\varphi_1 \in sol(\widetilde{q_2})$. *Since* $(\varphi_1, q_1) \leq_{\overline{\mathbb{A}}} (\varphi_2, q_2)$, *we have* $\varphi_1 \prec \varphi_2$ *and* $\alpha_2 \leq \alpha_1$. *Therefore,* $\alpha_2 \leq \alpha_1 \leq f(\varphi_1)$ *since* $\varphi_1 \in sol(\overline{a_2})$. *On the other hand, we have* $\varphi_1 \prec \varphi_2$, $\varphi_2 \in sol(q_2) \subseteq sol(\overline{m_2})$ *and* $\overline{m_2}$ *is monotonic. Therefore, we have* $\varphi_1 \in sol(\overline{m_2})$. *It follows that* $\varphi_1 \in sol(\overline{m_2}) \cap sol(\overline{a_2}) \cap sol(\widetilde{q_2}) = sol(q_2)$ *and* $\varphi_1 \prec \varphi_2$, *which contradicts the fact that* φ_2 *is minimal in* $sol(q_2)$ *with respect to* $\preceq$. *Hence, we can not have* $\varphi_1 \in sol(\widetilde{q_2})$, *which completes the proof.*

Example 9 *Let* q_1 *and* q_2 *be the simple mining queries of Example 1. Assume that we have already computed* $\overline{\lambda}_S[\mathcal{Q}]$ *where* $\mathcal{Q} = \{q_1\}$. *In this example, we show how this set can be updated after the computation of* $S(q_2)$. *First, we recall that :* $\overline{\lambda}_S[\mathcal{Q}] = \{(ABCE, q_1)\}$ *and* $S(q_2) = \{ABC, ACD\}$.

Now, we execute function Update$[S]$. *Let* $\lambda_S = \overline{\lambda}_S[\mathcal{Q}] = \{(ABCE, q_1)\}$ *and* $Ins = \emptyset$.

- *Let $\varphi = ABC$. Since $ABCE \preceq ABC$ and $\alpha(q_2) \leq \alpha(q_1)$, we have $X = \{(ABCE, q_1)\}$. Thus, $X \neq \emptyset$ and the pair (ABC, q_2) is not inserted in λ_S (Step 4).*
- *Let $\varphi = ACD$. Since ACD is not comparable with $ABCE$, we have $X = \emptyset$. Thus, the pair (ACD, q_2) is inserted in Ins (Step 5), i.e $Ins = \{(ACD, q_2)\}$. Then, since $\alpha(q_1) > \alpha(q_2)$, we have $Y = Z = \emptyset$. Therefore, no pair (φ_1, q_1) is deleted from λ_S.*

Finally, we have $\overline{\lambda}_S[\mathcal{Q} \cup \{q_2\}) = \lambda_S \cup Ins = \{(ABCE, q_1), (ACD, q_2)\}$. □

Function *Update*$[S]$

Input : *the sets $S(q)$ and $\overline{\lambda}_S[\mathcal{Q}]$ where q is a new mining query in $\mathbb{Q}_f$ and $\mathcal{Q} = \{q_1, \ldots, q_n\}$ with $q_i \in \mathbb{Q}_f$ $(i = 1, \ldots, n)$*
Output : *the set $\overline{\lambda}_S[\mathcal{Q} \cup \{q\}]$*

1. *Let $\lambda_S = \overline{\lambda}_S[\mathcal{Q}]$ and $Ins = \emptyset$*
2. **For** *every pattern $\varphi \in S(q)$* **do begin**
3. $X = \{(\varphi_i, q_i) \in \lambda_S \mid \varphi_i \prec \varphi$ *and* $\alpha(q) \leq \alpha(q_i)\}$
4. **If** $(X = \emptyset)$ **then begin**
5. $Ins = Ins \cup \{(\varphi, q)\}$
6. $Y = \{(\varphi_i, q_i) \in \lambda_S \mid \varphi_i = \varphi$ *and* $\alpha(q_i) \leq \alpha(q)\}$
7. **If** $(Y \neq \emptyset)$ **then**
8. $\lambda_S = \lambda_S \setminus Y$
9. **Else begin**
10. $Z = \{(\varphi_i, q_i) \in \lambda_S \mid \varphi \notin sol(\widetilde{q_i})$ *and* $\alpha(q_i) \leq \alpha(q)\}$
11. $\lambda_S = \lambda_S \setminus \{(\varphi_i, q_i) \in Z \mid \varphi \prec \varphi_i\}$
12. **End else**
13. **End if**
14. **End for**
15. **Return** $\lambda_S \cup Ins$

In the same way, we shall define a function Update$[G]$ to update the set $\overline{\lambda}_G[\mathcal{Q}]$ when a new mining query is inserted in $\mathcal{Q}$. Due to lake of space, this function is omitted here.

6.2 Update of Set of Closed Patterns

In this subsection, we present a function Update$[SC]$ to update efficiently a set $\lambda_{SC}[\mathcal{Q}, f]$ of closed patterns when a new mining query $q \in \mathbb{Q}_f$ is inserted in $\mathcal{Q}$. To this end, given a set Z of closed patterns, we assume that for every $\varphi \in Z$, we also store its upper and lower neighbors. The set of upper and lower neighbors of φ in Z, denoted $up(\varphi)$ and $down(\varphi)$ respectively, are defined by :

$$\begin{aligned} up(\varphi) &= max_{\preceq}\{\varphi' \in Z \mid \varphi' \prec \varphi\} \\ down(\varphi) &= min_{\preceq}\{\varphi' \in Z \mid \varphi \prec \varphi'\} \end{aligned}$$

Let $\lambda_{SC} = \lambda_{SC}[\mathcal{Q}, f]$ and $Z = \mathcal{SC}(\mathcal{Q}, f)$. For every pattern $\varphi \in SC(q, f)$, the function UpSC is used at Step 3 to compute the set of minimal patterns φ' in Z that are more general than φ, i.e. $X = UpSC(\varphi, Z) = min_{\preceq}\{\varphi' \in Z \mid \varphi \prec \varphi'\}$. In the same way, the function DownSC is used at Step 4 to compute the set of maximal patterns φ' in Z that are more specific than φ, i.e. $Y = DownSC(\varphi, Z) = max_{\preceq}\{\varphi' \in Z \mid \varphi' \prec \varphi\}$. We can note that these functions proceed level by level. The function UpSC starts from the most general patterns in Z, whereas the function DownSC starts from the minimal pattern in Z.

Then, at Step 5, we test if there exists a pattern $\varphi' \in Y$ such that $f(\varphi) = f(\varphi')$. If not, the pair $(\varphi, f(\varphi))$ has to be inserted in $\lambda_{SC}[\mathcal{Q}, f]$ (see Step 6). Then, we test if there exist some patterns $\varphi' \in X$ such that $f(\varphi) = f(\varphi')$. Indeed, for every pattern $\varphi' \in X$ such that $f(\varphi) = f(\varphi')$, we have $\varphi <_f \varphi'$. Therefore, these patterns must be deleted from λ_{SC} (see Step 9). Finally, we

note that the functions Insert and Delete are used both to insert or delete patterns in λ_{SC}, and to update the sets of neighbors of patterns in λ_{SC}.

Function *Update*$[SC]$

Input : *the set $SC(q, f)$ and $\lambda_{SC}[\mathcal{Q}, f]$ where q is a new mining query in $\mathbb{Q}_f$ and $\mathcal{Q} = \{q_1, \ldots, q_n\}$ where $q_i \in \mathbb{Q}_f$ $(i = 1, \ldots, n)$*
Output : *the set $\lambda_{SC}[\mathcal{Q} \cup \{q\}, f]$*

1. *Let $\lambda_{SC} = \lambda_{SC}[\mathcal{Q}, f]$ and $Z = \mathcal{SC}(\mathcal{Q}, f)$*
2. **For** *every pattern $\varphi \in SC(q, f)$* **do begin**
3. *Let $X = UpSC(\varphi, Z)$*
4. *Let $Y = DownSC(\varphi, Z)$*
5. **If** $(\forall \varphi' \in Y)(f(\varphi) \neq f(\varphi'))$ **then begin**
6. $Insert(\varphi, \lambda_{SC}, Y)$
7. **For** *every $\varphi' \in X$* **do**
8. **If** $(f(\varphi') = f(\varphi))$ **then**
9. $Delete(\varphi', \lambda_{SC}, \varphi)$
10. **End if**
11. **End for**
12. **Return** λ_{SC}

Example 10 *Let q_1 and q_2 be the simple mining queries of Example 1. Assume that we have already computed $\lambda_{SC}[\mathcal{Q}, f]$ where $\mathcal{Q} = \{q_2\}$. In this example, we show how this set can be updated after the computation of $SC(q_1, f)$. At first, we recall that :*

- $\lambda_{SC}[\{q_2\}, f] = \{(A, 0.8), (AB, 0.5), (AC, 0.5), (ABC, 0.4), (ACD, 0.3)\}$,
- $SC(q_1, f) = \{ABE, ABCE\}$.

Now, we execute function Update$[SC]$.

- *For $\varphi = ABCE$, we compute at Steps 3 and 4 the sets $X = UpSC(ABCE, Z) = ABC$ and $Y = DownSC(ABCE, Z) = \emptyset$. At Step 6, since $Y = \emptyset$, $(ABCE, 0.4)$ is inserted in λ_{SC}. Then, since $ABC \in X$ and $f(ABC) = f(ABCE)$, $(ABC, 0.4)$ is deleted from λ_{SC}.*
- *For $\varphi = ABE$, we compute at Steps 3 and 4 the sets $X = UpSC(ABE, Z) = AB$ and $Y = DownSC(ABCE, Z) = \emptyset$. At Step 6, since $Y = \emptyset$, $(ABE, 0.5)$ is inserted in λ_{SC}. Then, since $AB \in X$ and $f(AB) = f(ABE)$, $(AB, 0.5)$ is deleted from λ_{SC}.*

Finally, we have : $\lambda_{SC} = \{(A, 0.8), (AC, 0.5), (ABE, 0.5), (ACD, 0.3), (ABCE, 0.4)\}$. □

Function $UpSC(\varphi, Z)$

1. *Let* $X_0 = max_{\preceq}(Z)$
2. $X_0 = \{\varphi' \in X_0 \mid \varphi \prec \varphi')\}$
3. $k = 0$
4. **While** $X_k \neq \emptyset$ **do**
5. $X_{k+1} = \bigcup_{\varphi' \in X_k} up(\varphi')$
6. $X_{k+1} = \{\varphi' \in X_{k+1} \mid \varphi \prec \varphi')\}$
7. $X_k = X_k \setminus (\bigcup_{\varphi' \in X_{k+1}} down(\varphi'))$
8. $k = k + 1$
9. **End while**
10. **Return** $\bigcup_{j=0}^{k} X_j$

Function $DownSC(\varphi, Z)$

1. *Let* $Y_0 = min_{\preceq}(Z)$
2. $Y_0 = \{\varphi \in Y_0 \mid \varphi' \prec \varphi)\}$
3. $k = 0$
4. **While** $Y_k \neq \emptyset$ **do**
5. $Y_{k+1} = \bigcup_{\varphi' \in Y_k} down(\varphi')$
6. $Y_{k+1} = \{\varphi' \in Y_{k+1} \mid \varphi' \prec \varphi)\}$
7. $Y_k = Y_k \setminus (\bigcup_{\varphi' \in Y_{k+1}} up(\varphi'))$
8. $k = k + 1$
9. **End while**
10. **Return** $\bigcup_{j=0}^{k} Y_j$

Function $Insert(\varphi, var\ \lambda_{SC}, Y)$

1. $\lambda_{SC} = \lambda_{SC} \cup \{(\varphi, f(\varphi))\}$
2. $up(\varphi) = up(\varphi) \cup Y$
3. **For** *every* $\varphi' \in Y$ **do**
4. $\quad down(\varphi') = down(\varphi') \cup \{\varphi\}$

Function $Delete(\varphi', var\ \lambda_{SC}, \varphi)$

1. $\lambda_{SC} = \lambda_{SC} \setminus \{(\varphi', f(\varphi')\}$
2. $down(\varphi) = down(\varphi) \cup down(\varphi')$
3. **For** *every* $\varphi^* \in down(\varphi')$ **do**
4. $\quad up(\varphi^*) = (up(\varphi^*) \setminus \{\varphi'\}) \cup \{\varphi\}$
5. **For** *every* $\varphi^* \in up(\varphi')$ **do**
6. $\quad down(\varphi^*) = down(\varphi^*) \setminus \{\varphi'\}$

7 Conclusion

In this paper, we have proposed a general framework for iterative computation of mining queries based on condensed representations. To this end, we first adapt the standard notions of maximal, closed and key patterns introduced in previous works [8, 75]), to the more general case of sets of mining queries defined by monotonic and anti-monotonic selection predicates.

Then, we show in a particular case how condensed representations of a set $\mathcal{Q}$ of mining queries can be used : (i) to optimize the computation of the answer of a new mining query q, (ii) to efficiently modify the condensed representation of the answer of $\mathcal{Q}$ to obtain a condensed representation of the answer of $\mathcal{Q} \cup \{q\}$.

Based on this work and the approach proposed by [76, 23], we are currently investigating how disjunctive queries can be integrated in our framework. In this respect, we can note that given a set $\mathcal{Q} = \{q_1, \ldots, q_n\}$ of mining queries q_i $(i = 1, \ldots, n)$, we have : $sol(\mathcal{Q}) = sol(q_1 \vee \ldots \vee q_n)$. Moreover, in our approach, the answer of $sol(q_1 \vee \ldots \vee q_n)$ can be computed iteratively, computing at step k the answer of q_k using the condensed representation of $sol(\{q_1, \ldots, q_{k-1}\})$ $(k = 2, \ldots, n)$.

Moreover, it is clear that some tests are necessary to evaluate the various condensed representations proposed in the present paper, and the benefit of our iterative algorithms. To this end, we are implementing our approach in the context of our previous work [30], where mining queries are composed through relational operators.