République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderahmane Mira de Béjaïa

Faculté des Sciences Exactes

Département d'Informatique

École Doctorale Réseaux et Systèmes Distribués

Mémoire de Magistère

En Informatique

Option

Réseaux et Systèmes Distribués

Thème

Mise en œuvre de techniques d'optimisation de requêtes pour l'énumération de motifs intéressants dans des grandes bases de données

Présenté par

Ryme CHELOUAH

Devant le jury composé de :

Président	M.O. Bibi	Professeur	Université de Béjaïa, Algérie
Rapporteur	J.M. Petit	Professeur	Université de Lyon INSA, France
Examinateur	A. Boukerram	Maître de conférences	Université de Setif, Algérie
Examinateur	A. Mellit	Maître de conférences	Université de Jijel, Algérie
Invité	A. Kouhoul	Maître assistant	Université de Béjaïa, Algérie

Promotion 2006-2007

Dédicaces

A mes chers parents

A mes sœurs

A ma future belle famille

A tous mes amis

Remerciements

Je tiens à remercier et à exprimer ma gratitude à mes encadreurs le professeur Jean Marc PETIT et Abderaouf KOUHOUL pour m'avoir fait confiance, orienté et conseillé tout au long de ce travail.

Je remercie également les membres de jury, le professeur M.O. Bibi, Mr A. Boukerram et Mr A. Mellit, qui me font honneur en acceptant d'examiner et de juger mon travail.

Mes vifs remerciements vont également à tous les enseignants, qui ont assuré notre formation durant ce magistère.

Enfin, je remercie tous ceux qui ont contribué de près ou de loin à l'accomplissement de ce travail, ma sœur Sihem pour son aide et ses encouragements, mon futur mari, ma petite sœur et mes camarades et collègues de la promotion.

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithme

Introduction

Dans les entreprises, la quantité de données est importante et ces dernières s'accumulent avec une très grande vitesse et continue de l'être [3][4]. Face à ce déluge de données, le besoin de rentabiliser la collection de ces données en les transformant en informations utilisables par l'entreprise apparaît de plus en plus urgent. D'où l'émergence d'un nouveau domaine de recherche à l'intersection de plusieurs disciplines telles que les bases de données, l'intelligence artificielle, les statistiques : il s'agit de l'extraction de connaissance à partir des données.

La fouille de données ou data mining est une étape du processus d'extraction de connaissances à partir des données, qui consiste à appliquer des méthodes et techniques de découverte produisant un ensemble de motifs (modèles) sur les données [3][4]. Ces motifs constituent le résultat de la fouille et montrent les relations existantes dans les données. La fouille de données est considérée [19] comme l'une des dix technologies émergentes du XXIe siècle. Elle est utilisée dans plusieurs entreprises pour identifier des clients à forte valeur et adapter leur offre pour augmenter les ventes et minimiser les risques de perte grâce à la détection de fraude.

Depuis son émergence, de nombreux travaux de recherche ont contribué pour répondre aux différents problèmes de la fouille de données. Les solutions proposées sont destinées à répondre à des problèmes bien particuliers et leur adaptation pour résoudre d'autres problèmes s'avère difficile. Le projet iZi [7][8][5] remédie à ce problème, en proposant un outil facilitant l'implémentation de solutions pour des problèmes de fouille de données. L'idée est d'utiliser les algorithmes d'extraction de motifs fréquents pour résoudre des problèmes équivalents d'extraction de motifs intéressants dits représentables par des ensembles.

En 1996, Imielinski et al. ont proposé un nouveau concept ; il s'agit des bases de données inductives [10]. Ce concept propose d'intégrer dans un cadre commun la fouille de données et les bases

de données. Cette notion a soulevé ainsi plusieurs problèmes comme la définition de langages de requêtes data mining et d'optimisation.

Contribution

Dans ce mémoire, nous avons étudié des langages requête data mining et quelques algorithmes et techniques d'optimisation pour les problèmes en fouille de données.
Une approche pour l'optimisation des requêtes data mining a été proposée ; nous avons défini un langage déclaratif pour exprimer des requêtes data mining pour une classe particulière de problèmes d'extraction de motifs, dits " Représentables par des ensembles ".
Nous avons utilisé la logique DML pour représenter les requêtes utilisateur et étudier leurs formes logiques. La logique DML permet d'identifier deux propriétés importantes pour l'optimisation des requêtes : monotonie/anti-monotonie, ce sont les deux propriétés qu'exploitent les algorithmes génériques de la librairie iZi. Nous avons étendu également DML avec une classe importante de contraintes, dites succinctes ayant un pouvoir d'élagage de l'espace de recherche important. Une fois les formes logiques sont étudiées, un algorithme d'évaluation de la librairie iZi est choisi et un plan physique est généré pour répondre aux requêtes data mining.

Plan du mémoire

Ce mémoire se compose de cinq chapitres :
Le premier chapitre est une introduction au data mining, dans lequel nous avons défini la fouille de données et ses domaines d'application et présenté le processus d'extraction de connaissances à partir des données. Le second chapitre est consacré au cadre théorique, définissant les problèmes d'extraction de motifs intéressants, défini par Mannila et al. en 1997. A ce niveau, nous avons présenté également le projet iZi. Dans le chapitre trois, nous avons étudié quelques langages requêtes data mining comme nous avons établi par la suite une étude comparative sur ces langages et classé ces derniers en deux approches. Nous avons étudié dans le chapitre quatre quelques techniques d'optimisation et algorithmes pour résoudre des problèmes de fouille de données. Dans le chapitre cinq, nous proposons un langage requête pour la classe des problèmes d'extraction de motifs intéressants représentables par des ensembles et nous présentons également l'approche d'optimisation proposée. Enfin, le mémoire s'achève avec une conclusion et quelques

perspectives que nous envisageons pour améliorer notre proposition.

Chapitre 1

Introduction au data mining

1.1 Introduction

Le volume de données dans les entreprises est important, les données s'accumulent avec une très grande vitesse et continue de l'être. Ces données peuvent être stockées de manières diverses, dans des bases de données relationnelles, dans des entrepôts de données (datawarehouse), dans des bases de données transactionnelles,...etc. Le besoin de rentabiliser la collection de ces données en les transformant en informations utilisables par l'entreprise apparaît de plus en plus urgent. D'où la nécessité d'un ensemble de techniques et de méthodes d'extraction de connaissances des données collectées pour répondre à ce besoin, l'ensemble de ces techniques et méthodes est appelé le data mining.

1.2 Domaines d'application

Le data mining s'est manifesté [11][4] dans différents domaines d'application, entre autres on cite :

- **Grandes distributions :** analyser le comportement des clients à partir des tickets de caisse ; rechercher des similarités des consommateurs en fonction de critères géographiques ou sociodémographiques ; prédire le potentiel d'achat des clients au cours des prochains mois.
- **Banques :** modélisation prédictive des clients présentant des risques de clôture.
- **Laboratoires pharmaceutiques :** identifier les meilleures thérapies pour différentes maladies ; effectuer une analyse comportementale des officines dans la diffusion d'un nouveau

médicament.

- **Web mining et commerce électronique :** étudier les séquences de clic des clients par exemple pour analyser les caractéristiques des acheteurs et adapter le contenu du site.
- **Astronomie :** identification et classification des objets célestes.
- **Sécurité informatique :** recherche de transactions frauduleuses et suivi des opérations des traders.

1.3 Extraction de connaissances à partir des données

L'extraction de connaissances à partir des données ECD (KDD - Knowledge Discovery in Databases -) a été défini comme étant un processus non trivial pour l'identification de motifs valides, nouveaux, utiles et compréhensibles à partir de données [3][4].

La qualification non triviale signifie qu'une phase de recherche est effectuée et il ne s'agit pas d'un simple calcul. Les motifs découverts doivent être valides sur les nouvelles données avec un certain degré de certitude. Il est aussi intéressant que les motifs découverts soient nouveaux et utiles afin que l'utilisateur puisse en bénéficier. Enfin, les motifs doivent être compréhensibles si nécessaire après un post-traitement.

Le processus d'extraction de connaissances à partir des données (ECD) est itératif et interactif, il est possible de revenir à une étape antérieure comme l'utilisateur peut aussi intervenir et interagir avec le système.

Le terme processus implique que KDD comprend plusieurs étapes [3][4] :

1. La première étape est la compréhension du domaine d'application et l'identification des objectifs des utilisateurs.
2. Sélection des données : cette étape consiste à sélectionner, parmi toutes les données, celles sur lesquelles l'extraction de connaissances sera effectuée.
3. Nettoyage de données et prétraitement : une fois l'ensemble de données est sélectionné, il peut être nécessaire d'appliquer un prétraitement. Les données peuvent ne pas être décrites complètement et qu'il y ait des valeurs manquantes à certains attributs. Des défauts de mesures peuvent provoquer des erreurs qu'on qualifie de bruits. Cette étape consiste alors

à éliminer le bruit (nettoyage de données) présent dans les données et à la gestion des données manquantes.

4. Projection et réduction de données : Cette étape consiste à déterminer les caractéristiques utiles pour représenter les données et cela en fonction des objectifs définis dans l'étape précédente.

   Les données à ce niveau peuvent être transformées par des méthodes de réduction et de projection pour réduire le nombre de variables, discrétiser les valeurs d'un certain attribut continu.

5. Déterminer une correspondance entre les objectifs définis dans la première étape et une méthode data mining particulière : classification, segmentation...etc.

6. Fouille de données ou data mining : cette étape constitue le cœur du processus d'extraction de connaissances à partir des données. C'est l'étape où s'effectue l'extraction automatique des motifs à partir des données.

7. Interprétation des modèles extraits : il s'agit de voir si la connaissance extraite est nouvelle et pertinente pour le domaine considéré ; les modèles extraits sont représentés sous forme de graphes, de relations avec un degré de certitude.

8. Intégration de la connaissance : intégrer et exploiter la nouvelle connaissance dans un autre système pour un usage particulier.

Le processus KDD est itératif, nous constatons qu'au niveau de l'étape d'interprétation des résultats, il est possible de retourner à n'importe quelle étape de 1 - 6 pour d'autres itérations, tel qu'il est illustré sur la figure 1.1 [3][4].

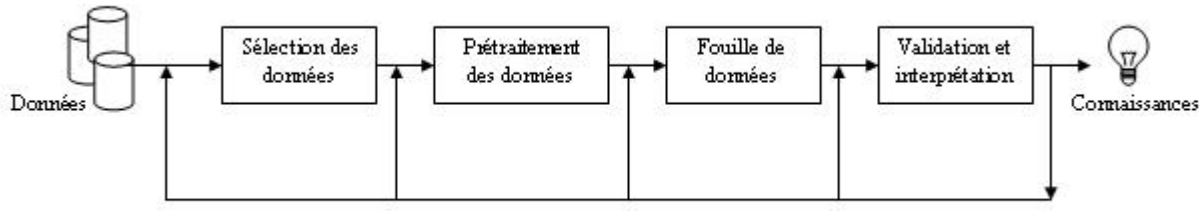

FIG. 1.1 – Les étapes du processus d'extraction de connaissances à partir des données.

1.4 Data mining

Data mining ou fouille de données est une étape du processus KDD, elle consiste à appliquer des algorithmes d'analyse et de découverte pour l'énumération des motifs à partir des données [3][4].

Les principales tâches que le data mining est amené à accomplir sont [19][11] :

Classification-Prédiction

La classification et la prédiction (régression) sont deux opérations qui visent à estimer la valeur d'une variable (dite cible ou à expliquer) d'un individu en fonction de la valeur d'un certain nombre d'autres variables du même individu (indiquées comme variables explicatives). La valeur à expliquer est qualitative dans le cas de la classification et continue dans le cas de la prédiction.
La classification permet de placer chaque individu de la population étudiée dans une classe, parmi plusieurs classes prédéfinies, en fonction des caractéristiques de l'individu indiquées comme variables explicatives. L'affectation à une classe à partir des caractéristiques explicatives se fait par une formule, un algorithme ou un ensemble de règles qui constitue un modèle et qu'il faut découvrir.

Segmentation

La segmentation ou le clustering est utilisée quand nous disposons d'un grand volume de données au sein duquel nous cherchons à distinguer des sous ensembles homogènes susceptibles de traitements et d'analyses différenciés.
Cette opération consiste à regrouper des objets en un nombre limité de groupes (segments ou clusters) ayant deux propriétés. D'une part, ils ne sont pas prédéfinis mais découverts au cours de l'opération et d'une autre part, les groupes de la segmentation regroupent les objets ayant des caractéristiques similaires et séparent les objets ayant des caractéristiques différentes.
La segmentation est beaucoup utilisée en marketing, médecine, sciences humaines... En marketing, elle vise à la recherche des différents profils de clients constituant une clientèle. Après avoir détecté les classes résumant sa clientèle, l'entreprise peut par exemple élaborer pour chacune d'elles une offre. Elle peut aussi suivre l'évolution de sa clientèle au fil des mois.

Association

Cette tache consiste à trouver des associations existantes dans des variables. L'application principale est l'analyse du panier de la ménagère qui consiste à rechercher les associations entre les produits achetés sur les tickets de caisse.

1.5 Conclusion

Dans ce chapitre, nous avons étudié le processus d'extraction de connaissances en présentant les différentes étapes qui le constituent. Notre travaille se focalise sur l'étape fouille de données qui est le cœur du processus ECD. Le cadre théorique définissant les problèmes d'extraction de motifs intéressants fera l'objet du prochain chapitre.

Chapitre 2

Cadre théorique d'extraction de connaissances

2.1 Introduction

Plusieurs problèmes de fouille de données sont équivalents au problème d'extraction des motifs fréquents [12]. Dans ce chapitre nous allons présenter le cadre théorique défini par Mannila et al [13] et généralisant ces problèmes en problèmes d'extraction de motifs intéressants. Nous allons étudier également quelques problèmes de fouille de données et les formaliser avec ce cadre théorique.

2.2 Définitions

Soient r une base de données, $\mathcal{L}$ un langage fini représentant l'ensemble des motifs et Q un prédicat permettant d'évaluer si un motif φ est vrai ou intéressant dans r. La tâche de fouille de données consiste à extraire les motifs intéressants de r relativement à $\mathcal{L}$ et Q, appelée théorie de $\mathcal{L}$, r et Q [13], définie par :

$$\text{Th}(\mathcal{L},\text{r},\text{Q})=\{\ \varphi \in \mathcal{L} |\ \text{Q}(\text{r},\ \varphi) \text{ est vrai }\}.$$

Supposons en plus qu'une relation de spécialisation / généralisation, notée $\preceq$ est définie sur les éléments de $\mathcal{L}$. On dira que φ est plus général (resp. plus spécifique) que θ si $\varphi \preceq \theta$ (resp. $\theta \preceq \varphi$).

Si le prédicat Q est monotone (resp. anti-monotone) par rapport à l'ordre $\preceq$, alors pour chaque φ, $\theta \in \mathcal{L}$ tels que $\varphi \preceq \theta$, nous avons :

$$Q(r, \varphi) \text{ est vrai (resp. faux)} \Longrightarrow Q(r, \theta) \text{ est vrai (resp. faux).}$$

2.3 Bordures d'une théorie

Lorsque le prédicat est anti-monotone, l'ensemble $Th(\mathcal{L}, r, Q)$ est dit fermé par le bas. Dans ce cas, il peut alors être représenté de façon équivalente par ses bordures positive et négative :

- Sa bordure positive, notée $Bd^+(Th(\mathcal{L}, r, Q))$, est composée des motifs intéressants les plus spécialisés par rapport à la relation d'ordre. Elle est définie par :
$$Bd^+(Th(\mathcal{L}, r, Q)) = \{\sigma \in Th(\mathcal{L}, r, Q) \mid \nexists \varphi \in Th(\mathcal{L}, r, Q), \sigma \preceq \varphi\}$$
- Sa bordure négative, notée $Bd^-(Th(\mathcal{L}, r, Q))$, est composée des motifs non intéressants les plus généraux par rapport à la relation d'ordre. Elle est définie par :
$$Bd^-(Th(\mathcal{L}, r, Q)) = \{\sigma \in \mathcal{L} \setminus Th(\mathcal{L}, r, Q) \mid \nexists \varphi \in \mathcal{L} \setminus Th(\mathcal{L}, r, Q), \varphi \preceq \sigma\}$$

L'union de ces deux bordures constitue la bordure de la théorie $Th(\mathcal{L}, r, Q)$.

Si le prédicat est monotone, l'ensemble $Th(\mathcal{L}, r, Q)$ est dit fermé par le haut et la notion de bordure existe aussi. La bordure positive est alors composée des motifs intéressants les plus généraux et la bordure négative des motifs non intéressants les plus spécialisés.

2.4 Représentation ensembliste

Soit $(\mathcal{L}, \preceq)$ l'ensemble ordonné composé des mots du langage $\mathcal{L}$, et d'un ordre partiel (relation de spécialisation / généralisation) $\preceq$ sur $\mathcal{L}$. Soit également R un ensemble fini d'éléments.
Il est parfois possible de définir un isomorphisme f de $(\mathcal{L}, \preceq)$ vers $(\mathcal{P}(R), \subseteq)$, ayant pour propriété de conserver l'ordre des éléments :

$$X \preceq Y \Longleftrightarrow f(X) \subseteq f(Y).$$

Si de plus la fonction f^{-1} est calculable, on dit alors que $(\mathcal{L}, \preceq)$ a une représentation ensembliste. Une condition nécessaire est que le nombre total de motifs dans $\mathcal{L}$ soit une puissance de 2. L'espace de recherche alors peut être représenté sous forme d'un treillis des parties de R.

2.5 Exemples de problèmes

Dans cette section, nous présentons quelques problèmes d'extraction de motifs intéressant appartenant au cadre théorique étudié ci-dessus.

2.5.1 Problèmes liés aux motifs fréquents

La découverte des motifs fréquents est introduite dans le cadre de la découverte des règles d'association [1]. Ce problème est à l'origine lié au besoin d'analyser des données de type transactions d'un supermarché pour déduire le comportement d'achat des clients.

Principe

Soit R un ensemble fini de symboles appelés articles. Une transaction est un sous ensemble de R. Une base de données de transactions r sur R est un ensemble de transactions.

- Un sous ensemble $X = \{i_1, i_2, ...i_k\} \subseteq$ R est appelé un motif ou k-motif s'il contient k articles.
- On note $cover(r, X) = \{\ t \in r / X \subseteq t\}$: le multi ensemble de transactions contenant le motif X.
- Le support d'un motif X dans r représente le nombre de transactions qui contiennent X : $support(r, X) = |cover(r, X)|$
- Un motif est dit fréquent si son support est supérieur à un certain seuil.

La propriété être motif fréquent dans une base de données r est anti-monotone relativement à la relation $\subseteq$:

$$\text{Soient X} \subseteq \text{R, Y} \subseteq \text{R avec X} \subseteq \text{Y, Y est fréquent} \Longrightarrow \text{X est fréquent.}$$

La base de données peut contenir de millions de transactions et l'espace de recherche est exponentiel dans le nombre d'articles présents dans la base de données.

L'espace de recherche des motifs peut être représenté par un treillis des parties de R : composé de $2^{|R|}$ motifs. Par conséquent lorsque R est grand l'espace de recherche devient très important.

Dans ce cas, la bordure positive de l'ensemble des motifs fréquents F est constituée des éléments maximaux par inclusion de F et sa bordure négative est constituée des éléments minimaux par inclusion n'appartenant pas à F. Les bordures sont donc :

$$Bd^{+} = \{X \in F / \forall Y, Y \supseteq X, Y \notin F\}$$
$$Bd^{-} = \{X \in 2^{R} \setminus F / \forall Y \subseteq X, Y \in F\}$$

2.5.2 Découverte des clés dans une base de données

Soit r une relation définie sur un schéma de relation R. Les attributs associés à R sont notés schéma(R). Nous allons introduire ci-dessous la définition de la clé minimale et de la super clé.

- X est une super clé de r si on a pour tout u, v ∈ r, u[X]≠v[X].
- X est une clé minimale de r si X est une super clé de r et ∀ Y ⊆ X, ∃ u, v ∈ r, u[Y]=v[Y].

On note SCle(r) et Cle(r) l'ensemble des supers clés et clés de r. Le problème consiste à extraire toutes les clés minimales de r.
Les super clés possibles sont tous les sous ensembles du schéma(R). Donc P(schema(R)) est l'ensemble des motifs du langage. L'ordre partiel est l'inclusion.
Le prédicat est "être une super clé" et est noté P(X, r). Ce dernier est monotone par rapport à l'inclusion :

$$\forall \text{ X, Y tel que X} \subseteq \text{Y, P(X, r) vrai} \Longrightarrow \text{P(Y, r) est vrai.}$$

"être une super clé" est un prédicat monotone [13]. En effet,
X est une super clé de r donc $|\pi_X(r)| = |r|$
Comme $X \subseteq Y \Longrightarrow |\pi_X(r)| \leq |\pi_Y(r)|$
$|\pi_Y(r)| = |r|$ donc Y est une super clé.

Les clés minimales d'une relation sont les super clés minimales par inclusion. L'ensemble des clés minimales correspond alors à la bordure positive de l'ensemble des super clés :

$$Cle(r) = Bd^{+}(SCle(r))$$

2.5.3 Découverte des dépendances fonctionnelles (DF) de la couverture canonique

La couverture canonique est l'ensemble des DF avec un seul attribut en partie droite et partie gauche minimale. La découverte de la couverture canonique des DF est un problème appartenant au cadre théorique étudié [13].

Soit r une relation définie sur un schéma de relation R. Les attributs associés à R sont notés $schema(R)$. Soit une dépendance fonctionnelle $X \longrightarrow B$, avec $X \subseteq schema(R)$ et $B \in schema(R)$ fixé. La dépendance est vraie dans r, si pour tout tuple u, $v \in r$, on a : si u et v ont la même valeur pour les attributs de X, alors ils ont la même valeur pour l'attribut B : $u[X] = v[X] \Longrightarrow u[B] = v[B]$.

Avec le cadre théorique, ce problème pourra s'exprimer ainsi [13] : Le langage $\mathcal{L}$ se constitue de l'ensemble des parties de $schema(R) \setminus B$ ou bien $\mathcal{P}(schema(R) \setminus B)$. La base de données est la relation r. Le prédicat de sélection p(r, X) est vrai si et seulement si $X \longrightarrow B$ est vrai dans r, avec $X \subseteq schema(R) \setminus B$.

La relation d'ordre est l'inclusion $\subseteq$ et le prédicat p est monotone par rapport à l'inclusion. La théorie $Th(r, \mathcal{P}(schema(R) \setminus B), p)$ est l'ensemble des parties gauches des DF $X \longrightarrow B$, $X \subseteq schema(R) \setminus B$, satisfaites dans r.

La bordure positive sera composée alors des plus petites parties gauches vérifiant la dépendance fonctionnelle et la bordure négative se constitue des plus grandes parties gauches qui ne vérifient pas la dépendance fonctionnelle.

2.5.4 Découverte des dépendances d'inclusion

Soient r et s deux relations définies respectivement sur les schémas relationnels R et S. Une dépendance d'inclusion DI est une expression de la forme $R[X] \subseteq S[Y]$, où X et Y sont des séquences d'attributs des schémas R et S respectivement telles que $\mid X \mid = \mid Y \mid$. La dépendance d'inclusion $R[X] \subseteq S[Y]$ est satisfaite dans (r, s) si toutes les valeurs de X dans r sont aussi des valeurs de Y dans s.

Ce problème peut être reformulé dans le cadre théorique [13]. Le langage $\mathcal{L}$ comprend toutes les dépendances d'inclusion non triviales. Un ordre partiel sur les DI peut être défini de cette manière :
Si i et j sont deux DI, $j \preceq i$ si j peut être obtenu en effectuant la même projection sur les deux parties de i.

Le prédicat de sélection $P(r, s, R[X] \subseteq S[Y])$ est vrai si la dépendance $R[X] \subseteq S[Y]$ est satisfaite dans (r, s). Le prédicat $R[X] \subseteq S[Y]$ est anti-monotone par rapport à $\preceq$ ainsi la notion de bordure peut être appliquée pour ce problème. La bordure positive $Bd^+(I)$ est l'ensemble des dépendances d'inclusion satisfaites les plus spécialisées et la bordure négative $Bd^-(I)$ est l'ensemble des dépendances d'inclusion non satisfaites les plus générales par rapport à l'ordre utilisé.

Pour obtenir une représentation ensembliste de ce problème, nous procèdons comme suit : Soit I_1 l'ensemble des dépendances d'inclusion satisfaites de taille 1. La fonction ens permet d'associer à chaque DI de $\mathcal{L}$, un ensemble de DI unaires de la manière suivante : $ens(i) = \{j \in I_1 \mid j \preceq i\}$.
Par exemple, si on considère la DI $R[ABC] \subseteq S[FGH]$, avec $\{A, B, C\} \in R$ et $\{F, G, H\} \in S$. On aura alors $ens(R[ABC] \subseteq S[FGH]) = \{R[A] \subseteq S[F], R[B] \subseteq S[G], R[C] \subseteq S[H]\}$.

Pour que la fonction ens soit bijective, on doit considérer que les DI ayant la partie gauche respectant un ordre fixe.

2.6 Le projet iZi

La mise en œuvre de solutions dans le cadre de la recherche en fouille de données est freinée par le temps nécessaire au développement de programmes opérationnels. Différentes implémentations des problèmes en fouille de données ont été proposées mais ces dernières sont destinées à résoudre un problème bien particulier. L'adaptation de ces programmes pour résoudre un autre problème, que le problème auquel le programme est destiné, s'avère difficile. Cette constatation a été observée au sein de l'équipe de Jean Marc Petit qui a travaillé sur l'adaptation d'algorithmes de fouille de données pour résoudre des problèmes théoriquement similaire [7][8][5].

Le cadre théorique utilisé dans ce projet est celui des problèmes, représentables par des ensembles, d'extraction de motifs intéressants défini dans [13] que nous avons présenté dans la section précédente.

L'objectif du projet iZi est de proposer un outil facilitant l'implémentation de solutions en utilisant les algorithmes d'extraction de motifs fréquents pour résoudre des problèmes équivalents d'extraction de motifs intéressants représentables par des ensembles [7][8][5].

Pour cela, au cours de ce projet une librairie a été proposée facilitant la résolution de tels problèmes en offrant des algorithmes et des structures de donnée génériques [7][8][5]. Une méthodologie aussi a été proposé, elle aide l'utilisateur à reformuler si possible son problème sous la forme d'un problème d'extraction de motifs intéressants et l'assiste dans le développement [7][8][5].

2.6.1 Aspects méthodologiques

1. **Reformulation d'un problème dans le cadre**

   Ce cheminement permet de vérifier qu'un problème appartient à la famille des problèmes traités par la librairie iZi.

   Définir l'espace de recherche
   Déterminer le langage qui permet d'exprimer les motifs et une relation d'ordre partiel sur ces motifs pour définir l'espace de recherche.

   Définir le prédicat
   Définir le prédicat revient à exprimer d'une manière formelle ce que veut dire "intéressant", autrement dit il suffit d'écrire la condition qui rend un motif intéressant dans la base de données.

   Identifier la sortie
   Il suffit de caractériser quels motifs seront donnés en sortie en fonction de l'ensemble des motifs intéressants.

   Exhiber une représentation ensembliste
   De nombreux problèmes s'expriment déjà de façon ensembliste donc cette étape est triviale. Mais quand ce n'est pas le cas, cette étape est nécessaire.

2. **Stratégies d'exploration disponibles**

   Une fois le problème est placé dans le cadre théorique, l'utilisateur doit choisir une stra-

tégie d'exploration de l'espace de recherche. Les algorithmes de parcours disponibles dans la librairie iZi sont :

- L'algorithme Apriori qui permet de découvrir la théorie et les bordures.
- L'algorithme ABS [6][5] qui permet de calculer uniquement les deux bordures.

Ces deux algorithmes peuvent traiter des prédicats anti-monotones ou monotones. Le choix de l'algorithme de parcours est laissé à l'utilisateur, cela dépend des caractéristiques du problème et de la sortie qu'il désire obtenir.

2.6.2 La librairie iZi

Une librairie C++ a été proposée [7][8][5], l'idée est d'offrir une boite noire permettant le développement rapide de programmes utilisant un algorithme préalablement choisi.

La librairie comprend des algorithmes et des composants utilisés par ces algorithmes. Ces derniers sont indépendants des problèmes à résoudre, ce sont des boites noires pouvant être utilisées pour résoudre n'importe quel problème. Ainsi, cela mène au développement rapide de programmes sans se soucier de l'implémentation d'algorithmes complexes. Les autres composants sont spécifiques à un problème donné et deviennent des paramètres aux algorithmes. Ainsi l'utilisateur doit les développer comme il peut aussi les réutiliser pour résoudre d'autres problèmes.

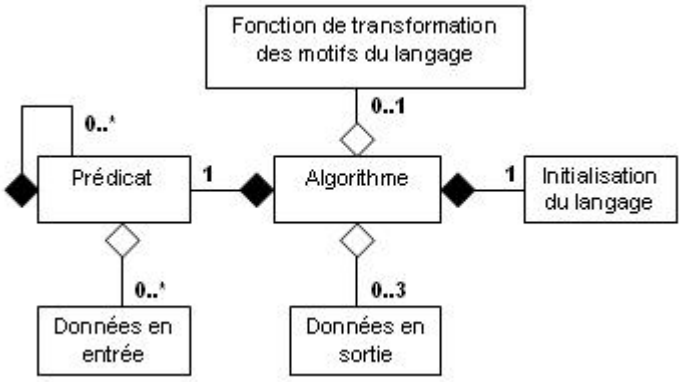

FIG. 2.1 – Diagramme de classe d'un algorithme dans la librairie iZi

La figure 2.1 représente le diagramme de classe d'un algorithme de la librairie iZi. Selon le problème data mining auquel on veut implémenter une solution, l'utilisateur devra développer jusqu'à cinq classes : les données en entrée de l'algorithme, la fonction d'initialisation du langage, le prédicat, les données en sortie de l'algorithme et la fonction de transformation des motifs du

langage.

Utilisation de la librairie iZi

En premier lieu, l'utilisateur doit définir les données sur lesquelles s'effectuera l'extraction, ces données seront utilisées par le prédicat pour sélectionner parmi tous les motifs ceux qui sont intéressants. Puis il faut définir la classe initialisation du langage permettant d'initialiser l'alphabet du langage. L'utilisateur doit ensuite implémenter la classe prédicat, cette classe est utilisée par l'algorithme pour tester si un motif est intéressant. L'utilisateur doit développer ou utiliser des classes pour sauvegarder la solution découverte par l'algorithme. Cette dernière peut être la théorie, la bordure positive et/ou la bordure négative. Si le problème ne s'exprime pas trivialement de façon ensembliste, il est impératif de développer une fonction permettant de transformer chaque élément du langage en un ensemble et prenant en considération la relation d'ordre partiel.
Pour implémenter une solution à un problème donné, il suffit d'instancier une instance de la classe d'un algorithme et de lui passer les composants nécessaires, que nous venons de citer, en paramètres.
Dans la version iZi que nous avons utilisée, des composants pour les problèmes de découverte des clés minimales d'une relation, des dépendances d'inclusion entre deux relations, des motifs fréquents et des motifs essentiels fréquents ont été développés.

La solution iZi facilite effectivement l'implémentation de solutions aux problèmes appartenant au cadre théorique proposé dans [13]. Dans notre cas par exemple, nous avons pu implémenter facilement un programme pour l'extraction de la couverture canonique des dépendances fonctionnelles [17].

2.7 Problème de découverte de la couverture canonique des dépendances fonctionnelles

Au cours de ce projet, nous avons implémenté une solution pour le problème de découverte de la couverture canonique des dépendances fonctionnelles à partir d'une base de données relationnelle. La solution a été conçue suivant la méthodologie iZi et a été implémentée en utilisant

la librairie iZi.

Pour cela, nous avons considéré le problème qui consiste à énumérer les dépendances élémentaires ayant l'attribut B dans la partie droite. Le prédicat de sélection p(r, X), où r est une base de données définie sur le schéma relationnel R, est vrai si $r \models X \longrightarrow B$.

L'ensemble des dépendances fonctionnelles canoniques ayant B en partie droite est $Th_B = \{X \subseteq R \setminus B \mid r \models p(r, X)\}$ et l'ensemble des DF élémentaires canoniques est alors la bordure positive de Th_B.

Algorithme 1 – Extraction de la couverture canonique

INPUT: r : une base de données relationnelle définie sur le schéma relationnel R
OUTPUT: CC : couverture canonique

1: $CC \longleftarrow \emptyset$
2: Récuperer la liste des attributs R de la base de données r.
3: Pour chaque attribut $A_i \in R$ faire
4: Instanciation d'une variable prédicat P "est une dépendance fonctionnelle ayant A_i comme partie droite"
5: Initialiser l'espace de recherche /* $init(r, A_i)$ */
6: $Apriori(init, r, P, BP^+_{A_i})$
7: $CC \longleftarrow CC \cup BP^+_{A_i}$
8: Fin Pour
9: **return** CC

Cet algorithme est itératif, chaque itération permet de fixer la partie droite et de chercher les ensembles d'attributs en dépendance fonctionnelle avec l'attribut fixé. Pour cela, nous avons implémenté deux composants : le prédicat de sélection et la fonction d'initialisation de l'espace de recherche. Deux instances de ces deux composants seront passées comme paramètres à l'algorithme Apriori qui effectuera le parcours de l'espace de recherche.

2.8 Conclusion

Dans ce chapitre, nous avons présenté le cadre théorique défini dans [13], il a été démontré aussi qu'une classe importante de problèmes sont équivalents au problème d'extraction de motifs fréquents [12]. Cela veut dire que les algorithmes de parcours de l'espace de recherche dans le cadre d'extraction des itemsets fréquents peuvent être utilisés aussi pour répondre à d'autres problèmes.

Il sera intéressant de réutiliser ces algorithmes pour faciliter l'implémentation de solutions. Le projet iZi a abordé ce problème et a porté des solutions en proposant une méthodologie et une

librairie facilitant l'implémentation de solutions aux problèmes data mining représentables par des ensembles. Une autre approche a été aussi proposée, dans laquelle l'utilisateur utilisera des primitives pour formuler des requêtes data mining et qui fera l'objet du prochain chapitre.

Chapitre 3

Langages requête data mining

3.1 Introduction

Depuis l'émergence du domaine d'extraction de connaissances à partir des données KDD, des solutions et des techniques sont développées pour résoudre des problèmes sur des données dont le format est spécifique comme des fichiers textes au lieu de véritables bases de données. En 1996, Imielinski et Mannila ont introduit un nouveau concept [10] : bases de données inductive (IDB -Inductive Data Base-) dont l'objectif est d'intégrer les outils d'extraction de connaissances et les systèmes de gestion de base de données [10]. Dans le cadre des IDB, une requête ordinaire est utilisée pour accéder et manipuler les données et une requête inductive peut être utilisée pour générer et manipuler des motifs, d'où la nécessité de définir un langage requête pour les bases de données inductives. Effectivement, des langages déclaratifs, permettant aux utilisateurs d'exprimer des tâches data mining, ont été proposés. Dans ce chapitre nous allons étudier quelques exemples de langages requête data mining comme nous allons établir une étude comparative sur ces derniers.

3.2 Exemples de langages de requête data mining

3.2.1 DMQL -Data Mining Query Language for relational databases-

DMQL est un langage de requête data mining [9]. Il a été conçu dans le projet DBMiner pour l'extraction de différents types de connaissances dans les bases de données relationnelles (règles d'association, règles de classification,...). DMQL a défini des primitives prenant en considération les points suivants :

1. Spécifier l'ensemble des données pertinentes pour la fouille de données,
2. spécifier le type de motifs à extraire,
3. spécifier les seuils qui seront utilisés pour sélectionner les motifs intéressants et filtrer les moins intéressants.

Syntaxe DMQL

DMQL a opté pour une syntaxe semblable à celle de SQL. Comme il est possible aussi d'utiliser les opérateurs relationnels FROM, WHERE et ORDER BY et de spécifier la base de données sur laquelle se déroulera l'extraction ainsi que les attributs pertinents pour l'opération. La syntaxe d'une requête DMQL est illustrée par la grammaire suivante :

$< DMQL >$::=
use database $< database\_name >$
{use hierarchy $< hierarchy\_name >$ for <attribute>}
$<rule\_spec>$
related to $<attr\_or\_agg\_list>$
from <relation(s)>
$[where < condition >]$
[order by $< order\_list >$]
$\{with[< kinds\_of >]threshold =< threshold\_value > [for < attribute(s) >]\}$
Avec :

- **use database**$<database\_name>$: permet de spécifier le nom de la base de données sur laquelle s'effectuera la fouille de données.
- **use hierarchy** $< hierarchy\_name >$ **for** <attribute> : elle est optionnelle, elle permet d'attribuer une hiérarchie à un certain attribut.
- $<rule\_spec>$: permet de spécifier le type de règle à extraire, son format change selon le type de la règle. Les différentes règles que DMQL permet d'extraire ainsi que leur syntaxe sont :
  1. Relation de généralisation (generalized relation) : permet d'obtenir une relation à partir d'un ensemble de données, cette relation pourra par la suite être utilisée pour extraire différents types de règles.
     $< rule\_spec >$::= generalize data $[into < relation\_name >]$

2. Règle caractéristique (caracteristic rule) : c'est une assertion qui caractérise un concept satisfait par tous ou la plupart des éléments d'une classe. Par exemple, les symptômes d'une certaine maladie peut être précisé par une règle caractéristique.
   $< rule\_spec >$::=
   find caracteristic rules [$as < rule\_name >$]
3. Règle discriminante : C'est une assertion qui discrimine un concept d'une classe des autres classes. Par exemple, pour distinguer entre une maladie des autres, la règle discriminante doit spécifier les symptômes discriminant cette maladie des autres.
   $< rule\_spec >$::=
   find discriminant rules [$as < rule\_name >$]
   for $class\_1$ with $< condition\_1 >$
   from$< relation(s)_1 >$
   in contrast to $< class\_2 >$ with $< condition\_2 >$
   from $< relation(s)\_2 >$
   {in contrast to $< class\_i >$ with $< condition\_i >$ from $< relation(s)\_i >$ }
4. Règle de classification : c'est une règle permettant de classifier les données.
   $< rule\_spec >$::= find classification rules [$as < rule\_name >$]
   [according to <attributtes>]
5. Règle d'association : détermine les règles d'association vérifiées dans un ensemble de données.
   $< rule\_spec >$::= find association rules [$as < rule\_name >$]

- **related to** $<attr\_or\_agg\_list>$: détermine une liste des attributs pour la généralisation.
- **from** <relation(s)> [**where** $< condition >$] : Les clauses from et where forment une requête SQL pour collecter l'ensemble des données.
- **with**[$< kinds\_of >$]**threshold** =$< threshold\_value >$ [**for** $< attribute(s) >$] : permet de spécifier les différents types de seuils à utiliser lors de l'extraction des règles.

Pour chaque type de règles, un algorithme d'extraction correspondant sera exécuté avec les paramètres et contraintes spécifiées dans la requête DMQL.

3.2.2 MINE RULE

MINE RULE est un opérateur conçu aux universités de Turin et Milan comme une extension du SQL [15], il permet d'exprimer différents problèmes de fouille de données concernant

l'extraction de règles d'association. Les principales caractéristiques de MINE RULE sont les suivantes :

1. Sélection de l'ensemble des données pertinentes pour la fouille de données.
2. Définition de la structure des règles en spécifiant par exemple la cardinalité de la partie en-tête et corps des règles à extraire.
3. Définition de mesures d'évaluation des règles.
4. Possibilité de travailler à différents niveaux de groupement de données.

Syntaxe MINE RULE

La syntaxe d'une requête MINE RULE est la suivante :

<MineRuleO> : :=MINE RULE <TableName>AS
SELECT DISTINCT <BodyDescr>, <HeadDescr>
[,SUPPORT][,CONFIDENCE]
[WHERE<WhereClause>]
FROM<FromList> [WHERE <WhereClause>]
GROUP BY <Attribute> <AttributeList>
[HAVING]<HavingClause>
CLUSTER BY <Attribute> <AttributeList>
[HAVING]<HavingClause>
EXTRACTING RULES WITH SUPPORT :<real>, CONFIDENCE :<real>

BodyDescr :=
[<CardSpec>]<AttrName><AttrList> AS BODY
/* default cardinality for Body : 1..n */
HeadDescr :=

[<CardSpec>]<AttrName><AttrList> AS HEAD
/* default cardinality for Head : 1..1 */
<CardSpec> :=<Number> ..(<Number> | n)
<AttributeList> :=,<AttributeName>

Avec :

- <FromList> et <WhereClause> dénotent les clauses From et Where du SQL.
- <TableName> et <AttributeName> sont des identificateurs respectivement d'une table et d'un attribut.
- <Number> dénote un nombre positif.
- <real> dénote un nombre réel.

Les règles d'association extraites seront sauvegardées dans une table relationnelle "TableName" dont les attributs sont spécifiés par la clause "SELECT", chaque tuple de cette dernière correspond à une règle extraite. Dans la clause "SELECT" l'utilisateur spécifie la structure des règles à extraire : l'ensemble d'attributs constituant la partie corps (body) et la partie en-tête (head) de la règle, la cardinalité de chaque partie et le support et la confidence. Le mot clé DISTINCT est obligatoire, permet d'éliminer les redondances au niveau du corps et en-tête des règles.

La clause GROUP BY permet de partitionner la relation en groupes, tel que chaque tuple d'un groupe a la même valeur des attributs AttributeList. La clause optionnelle HAVING associée à GROUP BY permet de filtrer les groupes, que les groupes dont tous les tuples vérifient la condition HavingClause sont pris en considération pour l'extraction. La clause optionnelle CLUSTER BY partitionne chaque groupe en sous groupes appelés clusters tel que les tuples appartenant au même cluster ont les mêmes valeurs des attributs de groupement. Le corps (body) et l'en-tête (head) d'une règle sont extraits à partir des clusters. Pour obtenir une règle, chaque paire de clusters (un pour le body et l'autre pour le head) dans le même groupe est considérée. La clause optionnelle HAVING de la clause CLUSTER BY sélectionne les paires de clusters à utiliser pour l'extraction, ces dernières sont celles qui vérifient la condition de la clause HAVING.

Pour chaque groupe et chaque paire de cluster, la clause SELECT extrait toutes les associations possibles et la clause optionnelle WHERE, qui la suit, permet de prendre en considération que les tuples vérifiant la condition lors de l'extraction. Le support d'une règle est le nombre de groupes à partir desquels la règle est extraite, divisé par le nombre total de groupes générés par la clause GROUP BY. La confiance est le nombre de groupes à partir desquels la règle est extraite, divisé par le nombre de groupes qui contiennent le body dans certains clusters. Dans le cas ou le support ou la confiance sont inférieurs aux seuils SUPPORT et CONFIDENCE

spécifiés, la règle est rejetée.

3.2.3 OLE DB for DM

OLE DB for DM ou OLE DB pour data mining a été conçu par Microsoft Corporation [18], c'est une extension de l'API (Application Programming Interface) OLE DB qui permet d'accéder facilement à une source de données relationnelles. Les algorithmes data mining travaillent sur des grands volumes de données comme ils nécessitent aussi la transformation de ces données en format intermédiaire pour l'extraction. OLE DB DM a pour principal but de faciliter la communication entre les algorithmes de Data Mining et les sources de données.

OLE DB pour data mining a introduit aussi un nouveau concept : modèle data mining (DMM -Data Mining Modele-). Le modèle data mining est semblable à une table relationnelle dont les ligne sont des descriptions de données (ensemble de cas). Une fois créé, l'utilisateur peut insérer des données dans le modèle issues de la base, appliquer un algorithme de fouille de données et utiliser le modèle pour la prédiction des valeurs d'un attribut pour les nouvelles données.

Pour la manipulation des DMM, OLE DB DM data mining a définit un langage comme extension de SQL.

Création d'un DMM

OLE DB for DM n'exige pas que les données aient un certain format pour l'extraction, l'utilisateur peut définir son propre modèle qui aura le format souhaitable dans lequel seront insérées les données.

Pour créer un modèle, OLE DB for DM a adopté la même syntaxe de création d'une table relationnelle avec SQL.

$< dm_create > : :=$CREATE MINING MODEL $< identifier >$ ($< col\_def\_list >$) USING $< algorithm >$ [($< algo\_param\_list >$)]

Pour chaque colonne, il est possible de préciser le type de l'attribut, s'il s'agit d'un attribut à prédire, s'il s'agit d'une clé,...etc. OLE DB for DM permet aussi d'imbriquer des tables à l'in-

térieur d'un DMM. Les données en entrée d'un algorithme d'extraction sont souvent obtenues en regroupant et en joignant l'information éclatée dans différentes tables d'une base de données. Si par exemple l'on fait la jointure des tables clients et ventes, il est possible de stocker dans une table imbriquée du modèle tous les produits achetés par un client donné.

OLE DB for DM a introduit un opérateur particulier, PREDICTION JOIN, c'est une extension de la jointure classique de SQL. Une fois que l'algorithme d'extraction a été exécuté, il devient possible de réaliser certaines opérations de jointure entre le DMM et des données ayant un schéma identique à celui du DMM en utilisant PREDICTION JOIN. Ce dernier permet de prédire la valeur de certains attributs des données en entrée en appliquant un DMM donné, à condition que ces attributs aient été indiqués comme attributs cibles lors de la construction du DMM.

3.2.4 DML -Data Mining Logic-

C'est une logique data mining introduite par Toon Calders et Jef Wijsen en 2001 [2]. Elle étend le calcul relationnel tuple avec des variables attribut qui s'étendent sur les attributs et des variables schéma qui s'étendent sur des ensembles d'attributs pour exprimer différentes tâches data mining, comme l'extraction des règles d'association, des dépendances fonctionnelles, dépendances d'inclusion...etc.

Syntaxe DML

Alphabet DML :

- Ensemble de variables attribut : X, Y, Z, X_1, Y_1, Z_1,...
- Ensemble de variables tuple : t, s, t_1, s_1,...
- Ensemble de variables schéma $\mathcal{X}$, $\mathcal{Y}$,... ayant l'arité n pour tout entier n.
- Un ensemble de constantes C.

L'arité d'une variable schéma est notée $U^{(n)}$, cela veut dire que la variable schéma U est un ensemble de n-ary tuples d'attributs. Les variables attribut et tuple sont appelées variables simples.

Formules DML atomiques :

1. Si X et Y sont deux variables attribut, t et s sont deux variables tuple et a une constante alors :

X=Y,

t.X=s.Y,

t.X=a,

sont des formules DML atomiques.

2. Si $\mathcal{X}$ est une variable schéma d'arité n et X_1,...,X_n sont des variables attribut alors $\mathcal{X}(X_1, ..., X_n)$ est une formule DML atomique.

Formules DML :

1. Une formule DML atomique est aussi une formule DML.
2. Si δ_1 et δ_2 sont deux formules DML alors $\neg\delta_1$ et $(\delta_1 \vee \delta_2)$ sont des formules DML.
3. Si δ est une formule DML et X est une variable attribut alors $\exists X(\delta)$ est une formule DML.
4. Si δ est une formule DML et t est une variable tuple alors $\exists t(\delta)$ est une formule DML.

Une formule DML est dite fermée si toutes les occurrences des variables simples sont quantifiées. Une formule DML fermée est dite aussi phrase DML (DML sentence). Les abréviations $\wedge$, $\rightarrow$, $\leftrightarrow$, True, False, $\forall$, $\neq$ sont aussi introduites. D'autres abréviations sont aussi définies :

- $\forall\mathcal{X}(X_1, ..., X_n)(\delta)$ pour $\forall X_1(...(\forall X_n(\mathcal{X}(X_1, ..., X_n) \longrightarrow (\delta)))...)$,
- t=s pour $\forall X(t.X = s.X)$

Requête DML

Une requête DML est une expression de la forme :

$$\{\mathcal{X}_1, ..., \mathcal{X}_m|\delta\}$$

avec δ est une phrase DML (DML sentense) et $\mathcal{X}_1$,....., $\mathcal{X}_m$ sont toutes des variables schéma apparaissant dans δ (m $\geq$1)

Sémantique DML

Un schéma est un ensemble fini non vide d'attributs. Un tuple sur un schéma S est une fonction de S vers l'ensemble de constantes C. Une relation est un ensemble fini de tuples.

La notion de structure DML est définie relativement à un schéma S, c'est une paire (R, Σ) ou R est une relation sur S et Σ est une attribution de variables schéma qui permet d'attribuer

$\Sigma(\mathcal{X}^{(n)}) \subseteq 2^{(S^n)}$ aux variables schéma d'arités n $\mathcal{X}^{(n)}$. Par convention, le schéma de R a été noté $|R|$ [2].

Une interprétation DML est une paire $((R,\Sigma),\sigma)$ ou (R,Σ) est une structure DML et σ est une attribution de variables simples qui permet d'attribuer un tuple sur $|R|$ pour chaque variable tuple t et d'attribuer $\sigma(X) \in |R|$ pour chaque variable attribut X.

La satisfaction d'une formule DML est définie relativement à une interprétation DML $((R,\Sigma),\sigma)$:

$$
\begin{array}{ll}
((R,\Sigma),\sigma) \models X = Y & \text{ssi } \sigma(X) = \sigma(Y) \\
((R,\Sigma),\sigma) \models t.X = s.Y & \text{ssi } \sigma(t)(\sigma(X)) = \sigma(s)(\sigma(X)) \\
((R,\Sigma),\sigma) \models t.X = a & \text{ssi } \sigma(t)(\sigma(X)) = a \\
((R,\Sigma),\sigma) \models \mathcal{X}(X_1,...,X_n) & \text{ssi } (\sigma(X_1),...,\sigma(X_n)) \subset \Sigma(\mathcal{X}) \\
((R,\Sigma),\sigma) \models \neg\delta & \text{ssi } ((R,\Sigma),\sigma) \not\models \delta \\
((R,\Sigma),\sigma) \models \delta_1 \vee \delta_2 & \text{ssi } ((R,\Sigma),\sigma) \models \delta_1 \text{ or } ((R,\Sigma),\sigma) \models \delta_2 \\
((R,\Sigma),\sigma) \models \exists X(\delta) & \text{ssi } ((R,\Sigma),\sigma_{X\longrightarrow A}) \models \delta \text{ pour un A} \in |\ R\ | \\
((R,\Sigma),\sigma) \models \exists t(\delta) & \text{ssi } ((R,\Sigma),\sigma_{t\longrightarrow r}) \models \delta \text{ pour un r} \in R
\end{array}
$$

L'attribution des variables tuple et attribut doit satisfaire la formule DML. La réponse à une requête DML est définie relativement à une relation R. La réponse à la requête DML suivante $\{\mathcal{X}_1,...,\mathcal{X}_m|\delta\}$ est l'ensemble :

$$\{(\Sigma(\mathcal{X}_1),...,\Sigma(\mathcal{X}_m))|(R,\Sigma) \text{ est une structure DML satisfaisant } \delta\}$$

Requêtes DML sur ensemble fermées et sous-ensemble fermées

La propriété d'anti-monotonie est généralisée dans le cadre des requêtes DML, appelée sous-ensemble fermée.

Définition 1 *[2] Soit δ une phrase DML avec la variable schéma $\mathcal{X}$. δ est sous-ensemble fermée (subset-closed) dans $\mathcal{X}$ si et seulement si pour chaque structure DML (R,Σ), pour chaque $T \subseteq \Sigma(\mathcal{X})$, si $(R,\Sigma) \models \delta$ alors $(R,\Sigma_{\mathcal{X}\rightarrow T}) \models \delta$.*

Une phrase DML est sous-ensemble fermée (subset-closed) si et seulement si elle est sous-ensemble fermée dans chaque variable schéma.

Une requête DML $\{\mathcal{X}_1,...,\mathcal{X}_m \mid \delta\}$ est sous-ensemble fermée (dans $\mathcal{X}_i$) si et seulement si δ est sous-ensemble fermée (dans $\mathcal{X}_i, i \in [1..m]$).

Les requêtes et formules DML sur-ensemble fermées (superset-closed) sont définies de la même manière sauf qu'il faut remplacer $T \subseteq \Sigma(\mathcal{X})$ par $T \supseteq \Sigma(\mathcal{X})$.

Les notions sur-ensemble fermée (superset-closed) et sous-ensemble fermée (subset-closed) sont complémentaires, la négation d'une phrase DML sous-ensemble fermée (subset-closed) est sur-ensemble fermée (superset-closed) et vice versa.

Théorème 1 *[2] La propriété sous-ensemble fermée pour une requête DML est indécidable.*

La classe de requêtes DML sous-ensemble fermées n'est pas identifiable, cela veut dire qu'il n'existe pas une caractéristique commune permettant d'identifier ces requêtes.
Mais peut-on trouver une sous classe identifiable de la classe des requêtes sous-ensemble fermées qui couvre sémantiquement une large classe ou entière des requêtes sous-ensemble fermées ? Les requêtes DML positives sont les candidats.

Requêtes DML positives

Définition 2 *[2] Une formule DML δ qui contient la variable schéma $\mathcal{X}$, est positive dans $\mathcal{X}$ (ou bien $\mathcal{X}$-positive) si et seulement si chaque symbole $\mathcal{X}$ est lié avec un nombre pair de négations. Une formule DML δ est positive si et seulement si elle est positive dans chaque variable schéma. Une requête $\{\mathcal{X}_1, ..., \mathcal{X}_m \mid \delta\}$ est positive (dans $\mathcal{X}_i$) si et seulement si δ est positive (dans $\mathcal{X}_i, i \in [1..m]$).*

Lemme 1 *[2] Soit δ une phrase DML. Si δ est $\mathcal{X}$-positive alors δ est $\mathcal{X}$-sur-ensemble fermée ($\mathcal{X}$-superset-closed). Si $\neg\delta$ est $\mathcal{X}$-positive alors δ est $\mathcal{X}$-sous-ensemble fermée ($\mathcal{X}$-subset-closed)*

Le lemme de Lyndon a confirmé que toute formule monotone du premier ordre peut être exprimé positivement. Mais ce dernier a échoué, il a été réfuté par Stolboushkin qui a donné un exemple de formule premier ordre monotone non équivalente à une formule positive.
Il a été démontré aussi dans [2] qu'il existe des requêtes DML sur-ensemble fermées avec une seule variable schéma qui ne peut pas être exprimée positivement.
Donc la classe des requêtes positives DML ne couvre pas entièrement la classe des requêtes sur-ensemble fermée, chaque requête DML positive est sur-ensemble fermée mais non chaque formule sur-ensemble fermée peut être exprimée positivement.

3.2.5 Comparaison

Le tableau ci-dessous illustre une étude comparative sur les langages étudiés précédemment.

TAB. 3.1 – Tableau comparatif

	Spécification explicite du type de motif à extraire	**Syntaxe**	**Définition des spécifications à chaque type de motif**	**Types de motif**	**Forme logique de la requête**	**Algorithme d'exploration**
DMQL	Oui	Extension de SQL	Oui	Relation généralisée, règle caractéristique, règle discriminante, règle de classification et règle d'association	Aucune	non précisé
MINE RULE	Non (un seul type)	Extension de SQL	Oui	Règles d'association	Aucune	non précisé
OLE DB DM	Non (prédiction)	Extension de SQL	Oui	Prédiction	Aucune	non précisé
DML	Non	Extension du calcul relationnel tuple	Requêtes uniformes pour les différents types de motif	Différents types de motif	Monotonie Positivité	Technique levelwise si la requête est monotone/anti-monotone

A l'exception de la logique DML, les langages que nous avons cités permettent d'exprimer qu'un certain type de tâches data mining. Cependant, l'expressivité de DML permet d'exprimer différents types de tâches data mining équivalents à l'extraction de motifs intéressant tels que les dépendances fonctionnelles, les dépendances d'inclusion, clés minimales, clés étrangères, itemsets fréquents,...

Nous avons classé ces langages selon deux approches :

1. Approche consistant à enrichir le langage utilisateur déclaratif de manipulation de bases de données relationnelles SQL, en introduisant des primitives data mining. Dans ce cas une primitive pour chaque type de problème comme dans le cas de DMQL, MINE RULE et OLE DB DM.
2. L'autre approche consiste à définir un nouveau langage déclaratif tel que DML pour ex-

primer différentes taches data mining.

3.3 Conclusion

A la fin de cette étude, nous avons constaté que la plupart de ces langages ne sont qu'une couche syntaxique autour du SQL pour intégrer quelques algorithmes data mining. Ils ne prennent en considération qu'une classe limitée de problèmes data mining et n'offrent qu'un nombre limité de primitives pour exprimer des requêtes. L'autre point, plus important aussi qui n'a pas été pris en considération, est l'optimisation. Ce dernier point a été invoqué dans le cas de MINE RULE [14] en introduisant des relations d'inclusion, de dominance et d'équivalence entre requêtes data mining pour exploiter les résultats de requêtes déjà calculés. D'autres techniques d'optimisation ont été aussi introduites dans le cadre de la fouille de données et que nous allons présenter dans le prochain chapitre.

Chapitre 4

Techniques et algorithmes d'optimisation data mining

4.1 Introduction

Concevoir une solution pour résoudre un problème n'est pas suffisant, il est intéressant de prendre en considération les performances de la solution et plus particulièrement lorsque l'on traite un volume important de données comme le cas de la fouille de données.
Dans ce chapitre, nous allons présenter quelques techniques d'optimisation proposées et quelques algorithmes utilisant ces techniques pour résoudre des problèmes data mining.

4.2 Techniques d'optimisation

4.2.1 Monotonie / Anti-monotonie

La monotonie et l'anti-monotonie sont deux propriétés caractérisant certaines contraintes ou prédicats. Elles permettent un élagage important de l'espace de recherche. Par conséquent, ces deux propriétés réduisent le nombre de tests à effectuer dans l'espace de recherche.

Définition 3 *Monotonie : [13]*
Soit C une contrainte définie sur les éléments de l'espace de recherche d'un problème de fouille de données. C est une contrainte monotone ssi :

$$\forall X, Y, X \subseteq Y, C(X) \Longrightarrow C(Y)$$

Définition 4 *Anti-monotonie : [13]*
Soit C une contrainte définie sur les éléments de l'espace de recherche d'un problème de fouille de données. C est une contrainte anti-monotone ssi :

$$\forall X, Y, X \subseteq Y, C(Y) \Longrightarrow C(X)$$

D'après la définition 3, si un ensemble ne vérifie pas une contrainte monotone alors tous ses sous ensembles ne la vérifient pas aussi. Et dans le cas d'une contrainte anti-monotone C, si un ensemble ne vérifie pas C alors ses sur-ensembles ne la vérifient pas aussi. Ainsi l'anti-monotonie et la monotonie permettent de réduire le nombre de tests à effectuer.
Ces deux propriétés sont complémentaires [13], si une contrainte C est monotone alors sa négation ¬ C est anti-monotone et vice-versa. La conjonction de contraintes monotones est aussi monotone et la conjonction de contraintes anti-monotones est anti-monotone [13].

4.2.2 Contraintes succinctes

Han et al. ont étudié l'optimisation des requêtes d'extraction des règles d'association avec contraintes [16]. L'utilisateur avant d'émettre sa requête concernant l'extraction des règles d'association, peut spécifier des contraintes que les règles doivent satisfaire. Leur challenge est de garantir un niveau de performance en étudiant les caractéristiques des contraintes imposées. Pour cela, ils ont introduit deux propriétés, caractérisant les contraintes, ayant un pouvoir d'élagage important : les contraintes anti-monotones et/ou succinctes. En se basant sur ces deux propriétés, ils ont classé les contraintes en quatre catégories puis proposé un algorithme d'extraction qui réalise un maximum d'élagage pour chaque catégorie de contraintes.

Requêtes d'extraction des règles d'associations avec contraintes

Han et al. ont introduit la notion CAQ - Constrained Association Query - ou bien requête d'extraction de règles d'association avec contraintes pour spécifier les contraintes que doit vérifier la partie antécédente S_1 et la partie conséquence S_2 d'une règle d'association extraite.
Une requête CAQ est définie de la manière suivante : $\{S_1, S_2 \mid C\}$ tel que C est l'ensemble de contraintes que doivent vérifier S_1 et S_2 y compris la fréquence. Ces contraintes peuvent être définies sur une seule variable comme la partie antécédente et/ou conséquence séparément comme elles peuvent lier les deux variables S_1, S_2 conjointement.

Contraintes à une variable

Une contrainte à une variable peut avoir l'une des formes suivantes [16] :

1. Contraintes classe : elle est sous la forme $S \subset A$, avec S est une variable ensemble et A est un attribut. Cette contrainte signifie que S est un ensemble de valeurs du domaine de l'attribut A.
2. Contrainte domaine : ce type de contraintes peut avoir l'une des formes suivantes.
   - $S\theta v$, S est une variable ensemble, v est une constante du domaine de S et θ est l'un des opérateurs booléens suivants : $=, \neq, <, \leq, >, \geq$. Cela signifie que chaque élément de S est en relation θ avec la constante v.
   - $v\theta S$, S et v sont définis comme précédemment et θ est l'un des opérateurs $\in$, $\notin$. Cela signifie que l'élément v appartient ou non à l'ensemble S.
   - $S\theta V$ ou $V\theta S$, S est une variable ensemble, V est un ensemble de constantes du domaine de S et θ est l'un des opérateurs $\subseteq, \not\subseteq, \subset, \not\subset, =, \neq$.
3. Contrainte aggrégat : elle est de la forme $agg(S)\theta v$, avec agg est l'une des fonctions d'aggrégat min, max, sum, $count$, avg et θ est l'un des opérateurs booléens $=, \neq, <, \leq, >, \geq$. Cela signifie que l'aggrégat de l'ensemble des valeurs numériques dans S sont en relation θ avec v.

Contraintes à deux variables

Une contrainte à deux variables peut avoir l'une des formes suivantes [16] :

1. $S_1\theta S_2, S_i$ est une variable ensemble et θ est l'un de $\subseteq, \not\subseteq, \subset, \not\subset$
2. $(S_1 \circ S_2)\theta V$, S_1 et S_2 des variables ensemble, V est un ensemble de constantes ou $\emptyset$, $\circ$ est l'un de $\cap, \cup$ et θ est l'un des $=, \neq, \subseteq, \not\subseteq, \subset, \not\subset$.
3. $agg_1(S_1)\theta agg_2(S_2)$, agg_1 et agg_2 sont deux fonctions d'aggrégat et θ est l'un des opérateurs booléens $=, \neq, <, \leq, >, \geq$.

L'optimisation et les contraintes anti-monotones

L'anti-monotonie est l'une des deux propriétés qu'ont utilisé Han et al. pour l'optimisation des requêtes CAQ. Une fois les contraintes anti-monotones sont identifiées, comment exploiter cette propriété ? L'idée est que l'optimisation avec élagage appliquée à la fréquence est aussi

applicable à toutes les contraintes anti-monotones.
Pour la fréquence, la propriété suivante est utilisée pour l'élagage lors du parcours de l'espace de recherche :
S, tel que $|S| = k$, est fréquent $\Longrightarrow \forall S' \subseteq S$ tel que $|S'| = k - 1$, S' est fréquent.
Lors du traitement d'une requête CAQ, si C_{am} est l'ensemble de toutes les contraintes anti-monotones de C y compris la contrainte de fréquence alors la généralisation de la propriété précédente est utilisée :
S, tel que $|S| = k$, satisfait $C_{am} \Longrightarrow \forall S' \subseteq S$ tel que $|S'| = k - 1$, S' satisfait C_{am}.

L'optimisation et les contraintes succinctes

L'optimisation réalisée par l'anti-monotonie se résume à l'élagage itératif : dans chaque itération de génération et test de l'algorithme Apriori, on peut élaguer des ensembles candidats qui requièrent le calcul de fréquence.
Une autre classe de contraintes a été identifiée, il s'agit des contraintes succinctes [16]. Ce type de contraintes permet l'élagage une fois pour toute et évite le paradigme de génération et test.

Soient les relations suivante : $trans(TID, Itemset)$ et $itemInfo(Item, Type, Price)$. S est une variable ensemble définie sur le domaine de l'attribut Item : $S \subset Item$. C est une contrainte à une variable, $SAT_C(Item)$ est l'ensemble qui contient les ensembles d'items satisfaisant C.

Définition 5 *Contraintes succinctes [16]*

- *$I \subseteq Item$ est un ensemble succinct ssi I peut être exprimé comme $\sigma_p(Item)$ pour un certain prédicat de sélection p.*
- *$SP \subseteq 2^{Item}$ est un ensemble de parties (powerset) succinct s'il existe un nombre fixe d'ensembles succincts $Item_1, ...Item_k \subseteq Item$ tel que SP peut être exprimé en fonction des powersets des $Item_1, ...Item_k$ en utilisant l'union et la différence.*
- *une contrainte C, à une variable, est succincte à condition que $SAT_C(Item)$ soit un powerset succinct.*

Dans le tableau 4.1, Han et al. ont introduit les différentes contraintes 1-variable que l'utilisateur peut imposer sur les règles d'association qu'il désire extraire [16] :

TAB. 4.1 – Caractérisation des contraintes à une variable

Contrainte 1-variable	**Anti-monotone**	**Succincte**
$S\theta v, \theta \in \{=, \leq, \geq\}$	oui	oui
$v \in S$	non	oui
$S \supseteq V$	non	oui
$S \subseteq V$	oui	oui
$S = V$	partiellement	oui
$min(S) \leq v$	non	oui
$min(S) \geq v$	oui	oui
$min(S) = v$	partiellement	oui
$max(S) \leq v$	oui	oui
$max(S) \geq v$	non	oui
$max(S) = v$	partiellement	oui
$cont(S) \leq v$	oui	faiblement
$count(S) \geq v$	non	faiblement
$count(S) = v$	partiellement	faiblement
$sum(S) \leq v$	oui	non
$sum(S) \geq v$	non	non
$sum(S) = v$	partiellement	non
$avg(S)\theta v, \theta \in \{=, \leq, \geq\}$	non	non
Contrainte de fréquence	oui	non

Toutes les contraintes avec fonction d'aggrégat de la forme $agg(S)\theta v$ doivent être écrite comme $agg(S.A)\theta v$ avec A est un attribut numérique [16]. L'exception avec la contrainte avec la fonction count() qui peut être utilisée avec deux manières : $count(S)\theta v$ ou $count(S.A)\theta v$. Cette dernière n'est pas dupliquée dans la table, elle se comporte comme les contraintes avec la fonction sum(). La fonction sum() suppose que les valeurs dans l'ensemble S (temps, largeur, prix,...) sont positives.

Théorème 2 *[16] Pour chaque contrainte C citée dans le tableau, C est succincte si le tableau 4.1 le dit.*

Nous avons défini ci-dessus qu'est ce qu'une contrainte succincte mais comment tenir compte de l'avantage de cette propriété ?
Pour répondre à cette question, Han et al ont introduit une autre notion qui est la fonction de génération membre d'un ensemble (MGF -Member Generating Function-) [16].

Définition 6 *Member Generating Function [16]*

1. *On dit que $SP \subseteq 2^{Item}$ a une MGF s'il y a une fonction, qui peut énumérer tous et que*

les éléments de SP, dont la forme est la suivante : $\{X_1 \cup ... \cup X_n \mid X_i \subseteq \sigma_{p_i}(Item), 1 \leq i \leq n, \exists k \leq n : X_j \neq \emptyset, 1 \leq j \leq k\}$ *pour* $n \geq 1$ *et les prédicats* $p_1, ..., p_n$.

2. *Une contrainte à une variable C est pre-counting prunable à condition que* $SAT_C(Item)$ *ait une MGF.*

Lemme 2 *[16] Si C est succincte alors C est pre-counting prunuable.*

L'utilisateur peut spécifier plusieurs contraintes dans sa requête. Comment calculer alors la MGF dans le cas ou nous avons plusieurs contraintes succinctes ? Pour cela, le lemme ci-dessous illustre comment fusionner deux MGF en une seule MGF.

Lemme 3 *[16] Soient* C_1, C_2 *deux contraintes succinctes avec leurs MGFs respectives :*

$\{S_1 \cup ... \cup S_m \mid S_i \subseteq \sigma_{p_i}(Item), 1 \leq i \leq m, \exists m' \leq m : S_i \neq \emptyset, 1 \leq i \leq m'\}$ *et* $\{T_1 \cup ... \cup T_n \mid T_i \subseteq \sigma_{p_i}(Item), 1 \leq i \leq n, \exists n' \leq n : S_i \neq \emptyset, 1 \leq i \leq n'\}$ *alors :* $\{R_{11} \cup ... \cup R_{mn} \mid R_{ij} \subseteq \sigma_{p_i \bigwedge q_j}(Item), 1 \leq i \leq m, 1 \leq j \leq n, R_{kl} \neq \emptyset, 1 \leq k \leq m', 1 \leq l \leq n'\}$ *est une MGF pour* $C_1 \wedge C_2$.

4.3 Algorithmes d'optimisation

4.3.1 Apriori

L'algorithme Apriori a été proposé par Agrawal [1] dans le cadre d'extraction de règles d'association. Extraire ces règles a un intérêt très important pour étudier le comportement des clients dans un centre commercial. La base de données dans ce type d'application est volumineuse, Apriori alors a été proposé pour une extraction facile et allant à l'échelle. Ce problème tel qu'il est formalisé dans [1] est :

Soit $\mathcal{I} = \{i_1, i_2, ..., i_n\}$ un ensemble d'éléments appelés items. Soit $\mathcal{D}$ un ensemble de transactions, chaque transaction $\mathcal{T}$ est un ensemble d'items telle que $\mathcal{T} \subseteq \mathcal{I}$. Une transaction $\mathcal{T}$ contient X, l'ensemble d'items de $\mathcal{I}$, si $X \subseteq \mathcal{T}$.
Une règle d'association est une implication de la forme $X \Longrightarrow Y$, tel que $X \subseteq \mathcal{I}$, $Y \subseteq \mathcal{I}$ et $X \cap Y = \emptyset$.
La confidence de la règle $X \Longrightarrow Y$ dans $\mathcal{D}$ est c si c% de transactions dans $\mathcal{D}$ qui contiennent X, contiennent aussi Y.

Le support de la règle $X \Longrightarrow Y$ dans $\mathcal{D}$ est s si s% des transactions dans $\mathcal{D}$ contiennent $X \cup Y$. Le problème d'extraction de toutes les règles d'association est décomposé en deux sous problèmes [1] :

1. Le premier consiste à calculer les ensembles d'items (itemset) ayant un support dépassant le support seuil fixé, il s'agit des itemset fréquents.
2. Le second consiste à construire les règles d'association à partir des itemsets fréquents. Pour chaque itemset fréquent I, calculer tous ses sous ensembles non vides. Puis, pour chaque sous ensemble a, la règles suivante $a \Longrightarrow (I - a)$ est générée si sa confidence support(I)/support(a) est égale au minimum à la confidence seuil.

L'algorithme Apriori permet de calculer les itemsets fréquents, il effectue un parcours par niveau et utilise la propriété anti-monotonie pour l'élagage de l'espace de recherche [1]. Pour chaque niveau k, il effectue les deux opérations suivantes :

1. Jointure : permet de générer l'ensemble C_k des candidats de taille k à partir des itemsets fréquents de taille k-1 en effectuant une jointure de l'ensemble L_{k-1} avec lui-même.
2. Elagage : c'est à ce niveau que sera exploitée la propriété anti-monotonie pour l'élagage de l'espace de recherche. Un itemset non fréquent de taille k-1 ne peut être un sous ensemble d'un candidat de taille k.

C_k : Comprend les itemsets candidats de taille k.

L_k : Comprend les itemsets fréquents de taille k.

L'algorithme Apriori

Algorithme 2 – Apriori

INPUT: $min\_support$
OUTPUT: Ans

1: $L_1 = \{$ les itemsets fréquents de taille 1 $\}$
2: **for** $(k = 2; L_{k-1} \neq \emptyset; k++)$ **do**
3: $C_k = apriori - gen(L_{k-1})$
4: **for all** (transaction t $\in D$) **do**
5: $C_t = subset(C_k, t)$ // candidats contenus dans t
6: **for all** (candidat c $\in C_t$) **do**
7: c.count++ ;
8: **end for**
9: **end for**
10: $L_k = \{c \in C_k \mid c.count \geq min\_support\}$
11: **end for**
12: **return** $(Ans = \cup_k L_k)$

C'est un parcours par niveau du bas vers le haut, l'algorithme commence d'abord par chercher les singletons qui sont fréquents L_1, puis itérativement génère et examine des ensembles de plus en plus large.

Pour chaque niveau k, il génère l'ensemble de ses candidats C_k en utilisant l'ensemble L_{k-1} des ensembles fréquent du niveau inférieur puis effectue un élagage pour filtrer les ensembles qui ne peuvent être fréquents en utilisant la propriété d'anti-monotonie. Les ensembles candidats se limitent à ceux ayant tous leurs sous ensembles fréquents. Une fois l'élagage est effectué, l'algorithme vérifie chaque candidat de C_k et ainsi utilise les itemsetss fréquents de taille k dans l'itération suivante. L'algorithme s'arrête lorsque il a parcouru tous les niveaux, ne dispose plus de candidats.

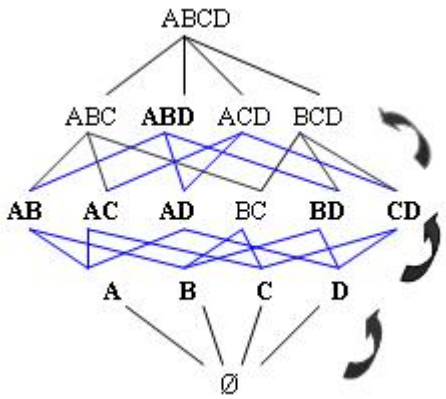

FIG. 4.1 – Exécution d'Apriori

La figure 4.1 illustre le fonctionnement de l'algorithme Apriori sur un treillis de motifs construits à partir de l'ensemble $\{A, B, C, D\}$. Les motifs en gras représentent ceux qui sont fréquents. Après la génération et l'élagage des motifs du niveau 2 on remarque que tous les motifs de taille 2 sont pris en considération pour calculer leur fréquence car tous leurs sous ensembles sont fréquents (propriété d'anti-monotonie). Au niveau 3 par exemple BCD est filtré car ses sous ensembles BC et CD ne sont pas fréquents, la même chose pour ABC.

Une implémentation de l'algorithme Apriori dans le cas d'un prédicat monotone est disponible dans la librairie iZi. Dans ce cas, Apriori effectue un parcours du haut vers le bas, il commence d'abord par vérifier l'ensemble le plus large puis itérativement génère et examine des ensembles plus petits et exploite la propriété de monotonie pour l'élagage de l'espace de recherche. Prenons comme exemple le problème de calcule de l'ensemble des clés dans une relation définie sur le schéma $\{A, B, C, D\}$. Le prédicat être une clé est monotone, si un ensemble d'attribut n'est pas une clé alors tous ses sous ensembles ne le sont pas aussi.

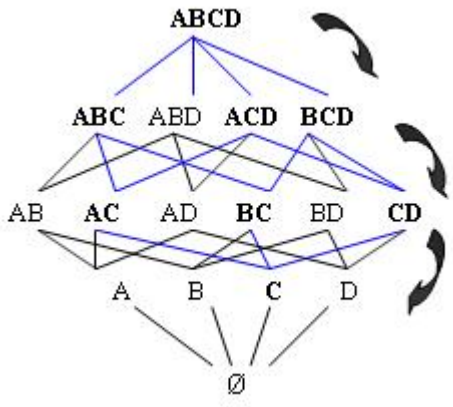

FIG. 4.2 – Exécution d'Apriori avec un prédicat monotone

Sur la figure 4.2 les motifs qui vérifient le prédicat sont mis en gras. En premier lieu l'algorithme génère ABCD, comme ce dernier constitue une clé alors tous ses sous ensembles sont vérifiés. Les motifs AD et BD sont élagués car le super ensemble ABD ne constitue pas une clé.

4.3.2 *Apriori*$^+$

Apriori$^+$ est une extension de l'algorithme Apriori, c'est un algorithme d'extraction des ensembles fréquents avec contraintes, proposé dans [16].
Soit C l'ensemble de ces contraintes y compris la fréquence qui est dénotée par C_{freq}.

L'algorithme *Apriori*$^+$

Algorithme 3 – *Apriori*$^+$

INPUT: C
OUTPUT: Ans
1: C_1= { les ensembles candidats de taille 1 }
2: k=1 ;
3: Ans= $\emptyset$;
4: **while** $C_k \neq \emptyset$ **do**
5: Lire la base de données et construire L_k à partir de C_k
6: Former C_{k+1} à partir de L_k en utilisant C_{freq} ; k++ ;
7: **end while**
8: Pour chaque ensemble S dans un certain L_k ajouter S dans Ans si S vérifie $(C - C_{freq})$.
9: **return** Ans

Apriori$^+$calcule en premier les ensembles fréquents (étapes 1-7) puis pour chaque ensemble fréquent, il vérifie le reste des contraintes. La solution finale comprend tous les ensembles qui vérifient toutes les contraintes dans C.

4.3.3 ABS

ABS (Adaptive Borders Search) [6][5] est un algorithme adaptif pour la découverte des bordures des itemsets fréquents, c'est un algorithme dérivé de l'algorithme ZigZag pour la découverte des dépendances d'inclusion dans une base de données. ABS suit une approche hybride, il change de stratégie au cours de l'exploration de l'espace de recherche. Ce changement de stratégie se fait dynamiquement en fonction des données.

ABS utilise deux stratégies d'exploration de l'espace de recherche, le parcours par niveau et la dualisation. L'objectif était d'exploiter Apriori et de changer de stratégie lorsque Apriori est mis en difficulté. La connaissance extraite ainsi sera utilisée pour débuter une autre stratégie se basant sur la dualisation entre la bordure positive et la bordure négative, chaque bordure est mise à jour et utilisée pour calculer l'autre bordure.

La première phase de l'algorithme consiste à utiliser Apriori pour explorer l'espace de recherche par niveau, des motifs plus petits aux motifs plus grands, jusqu'à un niveau k fixé dynamiquement à partir des données. Cette phase permet ainsi de calculer les motifs de la bordure positive ayant une taille strictement inférieure à k ainsi que les motifs de la bordure négative ayant une taille inférieure ou égale à k. L'idée derrière l'utilisation de cette technique de parcours est justifiée du fait qu'en pratique le nombre de motifs non fréquents de taille petite peut être élevé. Dans ce cas, la technique de parcours par niveau permet d'élaguer une grande partie l'espace de recherche. L'autre avantage aussi est que l'utilisation de cette technique est très efficace lorsque les motifs fréquents sont de petite taille.

La particularité d'ABS est son aspect adaptif qui consiste à déterminer à quel niveau le parcours par niveau doit être arrêté ou bien à quel moment la dualisation débutera. Aucun saut n'est autorisé avant un certain niveau car en pratique le parcours par niveau est très intéressant dans les premiers niveaux.
Considérons F_i l'ensemble des itemsets fréquents et C_i l'ensemble des itemsets candidats obtenus jusqu'à là par l'algorithme Apriori. Le ratio $\mid F_i \mid / \mid C_i \mid$ représente la proportion des de motifs fréquents générés. Si dans un certain niveau cette probabilité est proche de un, cela veut dire que l'espace de recherche à étudier comprend principalement des motifs fréquents. Ce cas de figure n'est pas favorable à une approche par niveau qui utilise l'anti-monotonie pour l'élagage de l'espace de recherche. Et les stratégies de sauts sont plus intéressantes, des expérimentations l'ont montrés dans [5]. En utilisant ainsi la propriété d'anti-monotonie, il est possible de déduire la bordure positive des motifs fréquents.

Une fois Apriori est arrêté, la dualisation est utilisée. Cette stratégie permet d'effectuer des sauts dans l'espace de recherche et de passer d'une bordure à une autre. ABS utilisera après les résultats déjà calculés pour estimer la bordure positive. Il effectue des dualisation de la bordure négative à la bordure positive et obtient ainsi les plus grands motifs non invalidés par un motif de la bordure négative déjà découvert. Les motifs obtenus ainsi seront soit fréquents et appartiennent alors à la bordure positive finale, soit ils seront non fréquents et l'algorithme doit porter une correction. Pour corriger l'erreur, ABS effectue aussi des dualisations pour estimer la bordure négative en fonction de la bordure positive en cours. Les candidats générés seront les plus

petits motifs non inclus dans un motif de la bordure positive et ces nouveaux motifs découverts seront utilisés pour corriger la bordure positive optimiste lors de la dualisation suivante.

L'algorithme ABS

Algorithme 4 – ABS

INPUT: une base de données binaire r, un entier minsup
OUTPUT: $Bd^+(FI)$ et $Bd^-(FI)$

1: $F_1 = \{A \in R \mid sup(A) \geq minsup\}$
2: $C_2 = AprioriGen(F_1)$
3: i = 2 ; $Bd_1^- = R - F_1$; $Bd_0^+ = \emptyset$
4: **while** $C_i \neq \emptyset$ **do**
5: $F_i = \{X \in C_i \mid sup(X) \geq minsup\}$
6: $Bd_i^- = Bd_{i-1}^- \cup (C_i - F_i)$
7: $Bd_{i-1}^+ = Bd_{i-2}^+ \cup \{X \in F_{i-1} \mid \forall Y \in F_i, X \not\subseteq Y\}$
8: **if** $IsDualizationRelevant(i, \mid Bd_i^- \mid, \mid F_i \mid, \mid C_i \mid)$ = TRUE **then**
9: $Bd_i^+ = \{X \in GenPosBorder(Bd_i^-) \mid\mid X \mid \geq minsup\}$
10: **while** $Bd_i^+ \neq Bd_{i-1}^+$ **do**
11: $Bd_i^- = \{X \in GenNegBorder(Bd_i^+) \mid\mid X \mid \leq minsup\}$
12: $Bd_{i+1}^+ = \{X \in GenPosBorder(Bd_i^-) \mid\mid X \mid \geq minsup\}$
13: i = i+1
14: **end while**
15: **return** Bd_i^+ and Bd_{i-1}^- and exit
16: **end if**
17: $C_{i+1} = AprioriGen(F_i)$
18: i = i+1
19: **end while**
20: $Bd_{i-1}^+ = Bd_{i-2}^+ \cup F_{i-1}$
21: **return** Bd_{i-1}^+ and Bd_{i-1}^-

L'algorithme ABS calcule la bordure positive et la bordure négative, il effectue un parcours par niveau. A chaque itération, ABS vérifie si Apriori doit être arrêté en utilisant la fonction *IsDualizationRelevant*. Une fois l'approche par niveau est arrêtée, les deux bordures seront progressivement mises à jour en effectuant des dualisations comme expliqué précédemment, Bd_i^+ (resp. Bd_i^-) est le sous ensemble de toute la bordure positive (resp. la bordure négative) découverts à l'itération i. On a : $\forall i < j, Bd_i^+ \subseteq Bd_j^+$ et $Bd_i^- \subseteq Bd_j^-$ et les motifs de Bd_i^+ sont obtenus à partir des motifs de Bd_i^- et les motifs de Bd_{i+1}^- sont obtenus à partir des motifs de Bd_i^+.
Les deux fonctions suivantes *GenPosBorder* et *GenNegBorder* calculent respectivement les bordures optimistes positives et négatives. L'algorithme s'arrête une fois les motifs de la bordure optimiste positive courants sont tous fréquents. Dans le cas où aucune dualisation n'est effectuée, ABS se réduit à l'algorithme APRIORI.

4.3.4 CAP

En tenant compte des contraintes succinctes et/ou anti-monotones, Han et al. ont classé les contraintes en quatre catégories [16] :

- Contraintes anti-monotones et succinctes.
- Contraintes succinctes non anti-monotones.
- Contraintes anti-monotones non succinctes.
- Contraintes non anti-monotones non succinctes.

En se basant sur les deux propriétés, ils ont également proposé une stratégie d'évaluation pour chaque catégorie [16].

1. **Contraintes anti-monotones et succinctes**

   Quand une contrainte C est anti-monotone alors sa MGF est de la forme suivante :

   $$\{S \mid S \subseteq \sigma_p(Item), S \neq \emptyset\}$$

   Dans ce cas l'ensemble C_1 des candidats de l'algorithme Apriori peut être simplement remplacé par :

   $$C_1^c = \{e \mid e \in C_1, e \in \sigma_p(Item)\}$$

   Comme C est une contrainte anti-monotone alors les ensembles, contenant un élément n'appartenant pas à C_1^c, ne sont pas pris en considération. Nous avons donc la stratégie suivante pour les contraintes succinctes et anti-monotones.

   Stratégie 1 *:*

   – *Remplacer C_1 dans l'algorithme Apriori par C_1^c comme défini ci-dessus.*

2. **Contraintes succinctes non anti-monotones**

   Dans ce cas, la contrainte n'est pas anti-monotone, donc on ne peut pas garantir que tous les ensembles satisfaisants la contrainte doivent être des sous ensembles de C_1^c. Néanmoins, on peut utiliser la structure donnée par la MGF de la contrainte. La stratégie décrit dans [16] est celle dans le cas ou la MGF est de la forme :

   $$\{S_1 \cup S_2 \mid S_1 \subseteq \sigma_{p_1}(Item), S_1 \neq \emptyset, S_2 \subseteq \sigma_{p_2}(Item)\}$$

Stratégie 2 *:*

- *Définir* $C_1^c = \sigma_{p_1}(Item)$ *et* $C_1^{\neg c} = \sigma_{p_2}(Item)$*. Définir aussi les ensembles correspondants des ensembles fréquents de taille 1 :* $L_1^c = \{e \mid e \in C_1^c, freq(e)\}$*et* $L_1^{\neg c} = \{e \mid e \in C_1^{\neg c}, freq(e)\}$
- *Définir* $C_2 = \{\{e, f\} \mid e \in L_1^c, f \in (L_1^c \cup L_1^{\neg c})\}$*.* L_2 *est l'ensembles des ensembles fréquents dans* C_2 *:* $L_2 = \{S \mid S \in C_2 \wedge freq(S)\}$*.*
- *L'ensemble* C_{k+1} *peut être obtenu à partir de* C_k *de la même manière que dans l'algorithme Apriori mais avec une seule modification. Dans le cas classique, un ensemble S est dans* C_{k+1} *si tous ses sous ensembles de taille k sont dans* L_k *:*

$$\forall S', S' \subseteq S \text{ et } |S| = k \Rightarrow S' \in L_k$$

Dans le cas de contraintes succinctes non anti-monotones, le principe est de calculer la fréquence que des sous ensembles S' ayant des éléments communs avec L_1^c*. Les sous ensembles qui sont disjoints avec* L_1^c *sont exclus ne nécessitent pas la vérification de la fréquence. Ceux sont des ensembles qui ne contiennent que des éléments de* $L_1^{\neg c}$*.*
Lors de la génération des candidats, la modification suivante est utilisée : Un ensemble S est dans C_{k+1}*, si pour tous ses sous ensembles :*

$$\forall S', S' \subseteq S, |S'| = k, S' \cap L_1^c \neq \emptyset \Rightarrow S' \in L_k$$

et

$$L_{k+1} = \{S \mid S \in C_{k+1}, freq(S)\}$$

3. **Contraintes anti-monotones non succinctes**

Dans cette catégorie, une contrainte C est non succincte, n'admet pas de MGF qui pourra être utilisé pour éviter génération et test. Comme C est anti-monotone donc lors de la génération des ensembles candidats pour un certain niveau, la satisfaction de la contrainte C est d'abord testée avant de calculer la fréquence. Les ensembles ne vérifiant pas la contrainte C ne demandent pas le calcul de fréquence.

Stratégie 3 *:*

- *Définir* C_k *comme dans l'algorithme Apriori. Ne pas calculer la fréquence d'un ensemble S si ce dernier ne vérifie pas C : tester la satisfaction de la contrainte avant de calculer la fréquence.*

4. **Contraintes non anti-monotones non succinctes**

   Pour traiter une contrainte C qui n'est ni succincte ni anti-monotone, le principe est d'essayer d'inférer des contraintes faibles qui peuvent être anti-monotones et/ou succinctes. Une fois trouvées, l'une des stratégies citées ci-dessus peur être utilisé. Dans ce cas, après avoir calculé les ensembles fréquents, une itération finale consistant à vérifier la satisfaction des ensembles fréquents pour la contrainte C.

   Stratégie 4 *:*

   - *Inférer une contrainte faible C' à partir de la contrainte C. Selon C', qu'elle soit succincte et/ou anti-monotone, utiliser l'une des trois stratégies pour la génération des ensembles fréquents.*
   - *Une fois les ensembles fréquents sont générés, il faut vérifier leur satisfaction pour la contrainte C.*

5. **Traitement dans le cas de plusieurs contraintes** Cette stratégie consiste à illustrer comment traiter une requête CAQ avec plusieurs contraintes, chacune appartient à l'une des catégories décrites précédemment.
   Soit C l'ensemble des contraintes dans une CAQ, y compris la contrainte de fréquence, et soient C_{sam} (respectivement C_{suc}, C_{am}, C_{none}) qui dénotent les contraintes de C qui sont succinctes et anti-monotones (respectivement succinctes mais non anti-monotones, anti-monotones mais non succinctes et ni anti-monotones ni succinctes). Les contraintes faibles inférées à partir des contraintes de C_{none} sont supposées incluses dans C_{suc}, C_{am}, C_{sam}.

   Stratégie de traitement dans le cas de plusieurs contraintes
   - Combiner les MGFs de toutes les contraintes dans C_{sam} comme montré dans le lemme 3. Appliquer la stratégie 1 avec la MGF obtenue.
   - Combiner les MGF de toutes les contraintes dans C_{suc}. Appliquer la stratégie 2 avec la MGF obtenue
   - Pour toutes les contraintes dans C_{am}, suivre la stratégie 3.
   - Pour chaque contrainte dans C_{none}, inférer les contraintes faibles et appliquer la stratégie 4.

   L'algorithme CAP (Constrained APriori) ci-dessous intègre la stratégie de traitement dans

le cas de plusieurs contraintes et montre comment les différentes stratégies sont appliquées dans Apriori.

L'algorithme CAP

Algorithme 5 – CAP

INPUT: $C_{sam}, C_{suc}, C_{am}, C_{none}$
OUTPUT: Ans
1: **if** $C_{sam} \cup C_{suc} \cup C_{none} \neq \emptyset$ **then**
2: Calculer C_1 comme indiqué dans les stratégies 1, 2 et 4
3: **end if**
4: **if** $C_{suc} \neq \emptyset$ **then**
5: Former L_1 comme indiqué dans la stratégie 2
6: Former C_2 comme indiqué dans la stratégie 2
7: $K = 2$
8: **end if**
9: **while** $C_k \neq \emptyset$ **do**
10: Former L_k à partir de C_k
11: Former C_{k+1} à partir de L_k en se basant sur la stratégie 2 si $C_{suc} \neq \emptyset$ et sur la stratégie 3 pour les contraintes dans C_{am}
12: **end while**
13: **if** $C_{none} = \emptyset$ **then**
14: $Ans = \bigcup L_k$
15: **else**
16: Pour chaque élément S dans un certain L_k, ajouter S à Ans ssi S satisfait C_{none}
17: **end if**

4.4 Tableau récapitulatif

Dans le tableau ci-dessous, nous avons établi une comparaison entre les différents algorithmes que nous avons étudiés dans ce chapitre, en illustrant les différentes techniques d'optimisation qu'ils utilisent.

Apriori permet de calculer les ensembles fréquents et utilise l'anti-monotonie pour effectuer un filtrage de l'espace des candidats générés à chaque itération (niveau). L'algorithme ABS exploite aussi cette propriété pour le calcul des ensembles fréquents, il effectue dans un premier temps un parcours par niveau.
Dans leurs versions implémentées dans la librairie iZi, Apriori et ABS sont généralisés d'une part pour effectuer aussi un parcours du haut vers le bas dans le cas des prédicats monotones et

d'une autre part pour traiter n'importe quel prédicat monotone ou anti-monotone.

Apriori$^+$ extension d'Apriori, donc utilise l'anti-monotonie pour le calcul des motifs fréquents, puis sur le résultat obtenu vérifie le reste des contraintes mais sans tenir compte des propriétés de ces contraintes.
L'algorithme CAP va plus loin qu'*Apriori*$^+$, en plus de l'anti-monotonie, CAP a étudié une autre classe de contraintes qui est la classe des contraintes succinctes.

TAB. 4.2 – Tableau récapitulatif

	Monotonie	Anti-monotonie	Succinctness
Apriori	x	x	
ABS	x	x	
Apriori$^+$		x	
CAP		x	x

4.5 Conclusion

L'optimisation dans le cadre des problèmes de fouille de données est une tâche très importante. Nous avons cité, dans ce chapitre, quelques techniques et algorithmes garantissant un certain niveau de performance. Mais certaines techniques et algorithmes restent spécifiques et ont été proposés pour répondre à un problème bien particulier. Prenons par exemple l'algorithme CAP, cet algorithme a été proposé pour l'évaluation des contraintes anti-monotones et/ou succinctes dans le cas d'extraction de règles d'association. Or que d'autres problèmes data mining peuvent bénéficier de ces techniques et les mêmes contraintes peuvent être utilisés pour exprimer d'autres problèmes de fouille de données.

Chapitre 5

Optimisation des requêtes data mining pour IZI

5.1 Introduction

Le nouveau concept des bases de données inductives a créé un nouveau champ d'étude comme dans le domaine des bases de données relationnelles où il a fallu des années de recherche pour répondre aux problèmes d'évaluation et d'optimisation de requêtes. Nous avons étudié dans le chapitre 4 quelques langages déclaratifs data mining mais ces derniers restent trop spécifiques à des problèmes bien particuliers avec un nombre limité de primitives. Un point important qui n'a pas été pris en considération est l'optimisation des requêtes, ces langages ne représentent qu'une couche syntaxique autour des algorithmes de fouille de données. L'autre point que nous avons constaté aussi est que les différentes solutions, proposées dans le cadre de la recherche en data mining, sont spécifiques et non génériques. Le projet iZi, abordé dans le chapitre 2, répond à ce problème et offre un outil générique.

Dans ce travail, nous nous sommes focalisé sur une sous-classe des problèmes data mining, c'est la classe des problèmes d'extraction de motifs intéressants représentables par des ensembles. D'une manière similaire aux bases de données relationnelles, nous proposons une approche pour l'optimisation des requêtes data mining pour cette classe de problèmes ; nous donnerons les détails de cette approche et nous justifierons aussi le choix de la logique que nous avons utilisée pour exprimer ces requêtes qui est la logique DML. Nous expliquerons également comment nous avons procédé pour enrichir DML. Ce choix a été fait après une étude bibliographique et pour

répondre à une fonction objective qui est l'optimisation des requêtes data mining. Nous verrons par la suite comment exploiter les techniques que nous avons étudiées pour répondre à notre fonction objective qui concerne la classe des problèmes considérés et comment la librairie iZi a été exploitée pour répondre aux requêtes utilisateur.

5.2 Approche d'optimisation pour IZI

L'idée principale de notre approche est de suivre le même processus de traitement des requêtes SQL en introduisant des techniques d'optimisation, cela pour répondre à des requêtes data mining pour les problèmes d'extraction de motifs intéressants et qui appartiennent au cadre théorique [13].

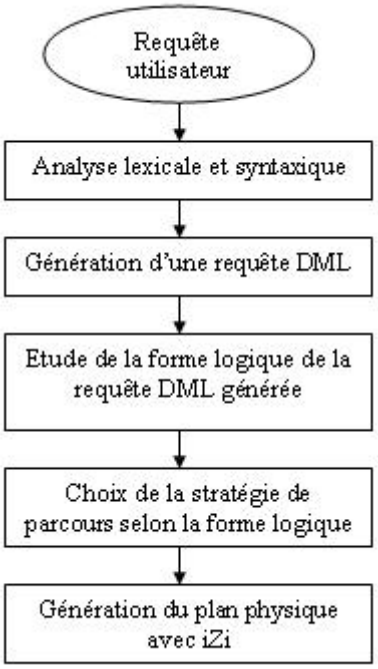

FIG. 5.1 – Processus d'évaluation des requêtes Data Mining

L'évaluation des requêtes est illustrée par la figure 5.1. Après l'analyse lexicale et syntaxique d'une requête utilisateur data mining, une requête DML sera générée. Il sera alors intéressant de définir une couche syntaxique ou bien un langage utilisateur, au dessus de DML, avec laquelle des requêtes utilisateur seront formulées et de développer un mapper permettant de transformer ces requêtes en requêtes DML.

Une fois la requête DML est générée, cette dernière sera analysée pour étudier ses formes logiques. Selon les formes logiques de la requête, un plan physique sera généré en utilisant les structures de données et les algorithmes de la librairie iZi. L'algorithme d'exploration d'espace de recherche sera choisi en fonction des formes logiques de la requête.

Cette approche est déclarative, l'utilisateur se limite à formuler sa requête data mining en exprimant ses besoins ainsi l'utilisateur ne s'occupera pas du code de bas niveau et des détails d'optimisation.
Nous allons voir ci-dessous les détails de cette approche. Nous définirons une couche syntaxique permettant à l'utilisateur de formuler ses requêtes. Puis, nous verrons comment la logique DML telle qu'elle est définie dans [2] permet d'exprimer la classe des problèmes data mining étudiée. Nous verrons également comment étendre DML avec des fonctions d'aggrégat et aussi avec une classe importante de contraintes, dites succinctes. En dernier, nous expliquerons comment étudier les formes logiques d'une requête DML et comment choisir la stratégie d'exploration pour répondre à la requête.

5.2.1 Logique DML

En étudiant la logique DML nous avons constaté que cette dernière peut être utilisée pour exprimer les différentes tâches data mining et principalement celles qui rentrent dans le cadre de la bibliothèque iZi. Deux propriétés sur lesquelles iZi s'est basé : la représentation ensembliste et la monotonie/ anti-monotonie. Nous allons voir alors ci-dessous comment DML prend en considération ces deux propriétés.

Représentation ensembliste

Soient r une relation définie sur le schéma R, le langage L constitué des parties de R ou bien $\mathcal{P}(R)$ dits motifs, $\leq$ la relation d'ordre partiel sur les éléments de L et le prédicat P(X, r) permettant d'évaluer si X est un motif intéressant dans r.

La représentation ensembliste est vérifiée s'il existe un isomorphisme f de (L, $\leq$) vers ($\mathcal{P}(R)$, $\subseteq$), ayant pour propriété de conserver l'ordre des éléments de R :

$$X \leq Y \Longleftrightarrow f(X) \subseteq f(Y)$$

La logique DML exprime aussi les problèmes ayant la représentation ensembliste. Rappelons nous que la réponse à la requête DML $\{\mathcal{X}_1, ..., \mathcal{X}_m \mid \delta\}$ est l'ensemble :

$$\{\Sigma(\mathcal{X}_1), ..., \Sigma(\mathcal{X}_m) \mid (R, \Sigma) \text{ est une structure DML satisfaisant } \delta\}$$

La représentation ensembliste apparaît bien dans l'attribution des variables schéma, Σ attribue $\Sigma(\mathcal{X}^{(n)}) \subseteq 2^{S^n}$ (S est le schéma sur lequel la relation R est définie) à chaque variable schéma $\mathcal{X}^{(n)}$ dont l'arité est n.
Cela veut dire que Σ attribue un sous ensemble d'attributs pour chaque variable schéma $\mathcal{X}^{(n)}$ d'arité n apparaissant dans la requête DML. L'espace de recherche alors est un treillis de parties $\mathcal{P}(S^n)$ pour chaque variable schéma dans la requête DML.

Monotonie/anti-monotonie

La propriété d'anti-monotonie (respectivement monotonie) utilisée dans l'algorithme Apriori est généralisée dans la logique DML, cela correspond aux requêtes sous-ensemble fermées (subset closed) (respectivement requêtes sur-ensemble fermées (superset closed)). Cette classe de requête DML est identifiable, la classe des requêtes positives couvre une classe importante de requêtes sur-ensemble fermées (superset closed).

D'après ce que nous venons de voir, DML prend bien en considération les contraintes sur lesquelles se base iZi en ce qui concerne la représentation ensembliste et la propriété de monotonie/anti-monotonie qui offre un niveau d'optimisation important grâce à la technique de parcours level-wise. Donc les algorithmes disponibles dans la librairie iZi peuvent être utilisée dans le cadre de la logique DML. Par conséquent cela encourage son utilisation dans le cadre iZi pour exprimer différentes tâches data mining.

Nous constatons aussi que la logique DML telle qu'elle a été définie n'introduit pas les fonctions d'aggrégat. Comment peut-on alors introduire ces fonctions tout en assurant un niveau de performance ?

DML et les fonctions d'aggrégat

Comme nous l'avons constaté, la logique DML, telle qu'elle a été définie, n'intègre pas les fonctions d'agrégat qui permettront à l'utilisateur de spécifier des contraintes que doivent vérifier les motifs extraits.

Prenons par exemple le problème d'extraction d'itemsets fréquents à partir d'un ensemble de transactions, ce problème consiste à extraire tous les ensembles de produits achetés simultanément et ayant un support dépassant un certain seuil défini par l'utilisateur.

Supposons qu'on veut définir une contrainte que doit vérifier ces ensembles fréquents, par exemple le prix total des produits achetés simultanément $\mathcal{X}$ ne dépasse pas une certaine valeur.
La fonction d'aggrégat $sum(\mathcal{X}.prix)$, permet d'exprimer cette contrainte de la même manière avec SQL. Ce type de contraintes a été étudié dans [16] et leurs propriétés ont été aussi étudiées pour l'optimisation.

Fonctions d'aggrégat

Comme nous l'avons vu dans le chapitre précédent, Han et al. ont étudié deux classes de contraintes dans le cadre d'extraction de règles d'association : les contraintes anti-monotones et / ou succinctes [16]. Ils ont aussi proposé des stratégies qui intègrent l'évaluation de ces contraintes avec le calcul de fréquence [16].

Une classe de ces contraintes utilise des fonctions d'aggrégat, nous avons choisi d'intégrer ces mêmes fonctions dans les requêtes DML pour exprimer des contraintes que doit vérifier les variables schéma.

Les contraintes avec fonctions d'aggrégat introduites dans [16] sont définies ainsi :
$agg(S)\theta v$ avec agg est l'une des fonctions suivantes : min, max, sum, $count$, avg. θ est l'un des opérateurs booléens suivants : $=, \neq, <, \leq, >, \geq$ et S est un ensemble de valeurs numériques.

Ces fonctions prennent S comme argument et S est une variable ensembliste. Dans le cadre de DML, S correspondra à une variable schéma.

Comme les problèmes traités dans notre cas sont des problèmes ayant la représentation ensembliste, ces fonctions seront utilisées alors pour définir des contraintes que doit vérifier les variables schéma.

Ces fonctions ont l'avantage d'enrichir DML pour définir des contraintes, comme ces dernières ont aussi des propriétés importantes pour l'optimisation. Les contraintes anti-monotones et / ou succinctes ont un pouvoir d'élagage de l'espace de recherche important.
Et selon l'utilisation de ces fonctions pour exprimer une contrainte, on pourra utiliser le tableau des caractéristiques des contraintes à une variable [16] pour déduire les caractéristiques d'une contrainte.

Contraintes ensemblistes

Les contraintes avec les opérateurs ensemblistes peuvent être aussi utilisés. Elles facilitent l'expression de contraintes sur les variables schéma comme elles ont des propriétés importantes (l'anti-monotonie et la succinctness) pour l'optimisation des requêtes. Nous allons voir ci-dessous un exemple de requête exprimée avec deux manières, une fois en utilisant DML comme définie dans [2] et l'autre en utilisant une contrainte de la table 4.1.

Prenons la requête qui consiste à extraire les itemsets fréquents. Le prédicat " être fréquent" est anti-monotone :

$\{\mathcal{X} \mid \exists t_1, ..., t_n(\bigwedge_{1 \leq i < j \leq n} t_i \neq t_j \wedge \forall \mathcal{X}(Y)(t_1.Y = 1 \wedge ... \wedge t_n.Y = 1))\}$
Cette requête est sous-ensemble fermée (subset-closed). Supposons maintenant qu'on veut avoir les itemsets fréquent contenant l'élément " article1 ", la requête sera :

$\{\mathcal{X} \mid \exists t_1, ..., t_n(\bigwedge_{1 \leq i < j \leq n} t_i \neq t_j \wedge \forall \mathcal{X}(Y)(t_1.Y = 1 \wedge ... \wedge t_n.Y = 1)) \wedge \exists \mathcal{X}(A)(A = "article1")\}$

En appliquant les abréviations introduite dans on [2] obtient :

$\{\mathcal{X} \mid \exists t_1, ..., t_n(\bigwedge_{1 \leq i < j \leq n} t_i \neq t_j \wedge \forall Y(\neg \mathcal{X}(Y) \vee (t_1.Y = 1 \wedge ... \wedge t_n.Y = 1))) \wedge \exists A(\mathcal{X}(A) \wedge A = "article1")\}$

La formule entière n'est ni monotone ni anti-monotone, la première partie de la conjonction est sub-set fermé (sa négation est fermée) donc anti-monotone et la deuxième partie de la conjonction est positive donc monotone. Cette dernière signifie que l'un des éléments de la variable schéma doit être article1, autrement dit "$article1" \in \mathcal{X}$. D'après la table 4.1, cette contrainte est succincte. En utilisant cette contrainte, la requête s'exprimera ainsi :

$$\{\mathcal{X} \mid \exists t_1, ..., t_n(\bigwedge_{1 \leq i < j \leq n} t_i \neq t_j \wedge \forall \mathcal{X}(Y)(t_1.Y = 1 \wedge ... \wedge t_n.Y = 1)) \wedge "article1" \in \mathcal{X}\}$$

Dans ce cas la requête est une conjonction d'une formule sous-ensemble fermée (subset-closed) et d'une autre succincte. En exprimant la requête avec les contraintes de [16] est plus intéressant, car cela nous a permis d'identifier d'une part une propriété pour l'optimisation importante et d'une autre part de simplifier la formulation de requêtes.

L'autre avantage est dans l'implémentation, le composant prédicat est très important et il n'est pas intéressant de le redéfinir pour chaque problème data mining exprimant des contraintes sur la variable schéma, il sera plus intéressant de le réutiliser. Dans ce cas par exemple, le prédicat être fréquent pourra être utilisé avec d'autres contraintes.

Pour cela, nous avons porté des modifications au niveau syntaxique et niveau sémantique :

1. **Niveau syntaxique :**

   Un ensemble de symboles fonctions est ajouté à l'alphabet DML comme f, g, h,... Nous avons introduit deux types de fonctions :
   - Fonctions d'aggrégat ayant comme paramètre une variable schéma.
   - Fonctions d'aggrégat ayant comme paramètre un attribut du schéma relationnel sur lequel est définie la relation sur laquelle s'effectue l'extraction.

   Pour le premier type de fonctions d'aggrégat, celles ayant comme paramètre une variable schéma, consiste à exprimer des contraintes que doit vérifier une variable schéma. Elles sont définies ainsi :

   $f : \mathcal{P}(I) \longrightarrow D$
   $\mathcal{X} = X_1, X_2, ..., X_n \longmapsto f(\mathcal{X})$

Tel que $\mathcal{P}(I)$ est l'ensemble des parties d'un ensemble d'éléments I et $\mathcal{D}$ est un ensemble numérique.

Comme exemple de fonctions nous avons la fonction count() [16] qui représente la cardinalité d'un ensemble.

L'autre classe de fonctions d'aggrégat consiste à définir aussi une contrainte que doit vérifier la variable schéma mais elle prend comme paramètre un attribut du schéma relationnel $schema(R)$ sur lequel est définie une relation R sur laquelle s'effectuera l'extraction. Elle est définie ainsi :

$g : schema(R) \longrightarrow D$
$\mathcal{X}.A \longmapsto g(\mathcal{X}.A)$

tel que $A \in schema(R)$ et D est le domaine de cet attribut.
Cela signifie que l'ensemble de valeurs de l'attribut A des différents éléments de la variable schéma $\mathcal{X}$ vérifient la fonction d'aggrégat g. Comme exemple de fonctions nous citons min() et max().
Comme exemple, prenons le problème qui consiste à chercher les ensembles des articles tel que le prix minimum de chaque ensemble est v. Soit S la variable ensemble représentant l'ensemble de ces articles, la contrainte sera alors $min(S.prix) = v$. Cette contrainte est définie sur la variable schéma S qui représente un sous ensemble d'articles et elle prend comme paramètre un attribut d'un schéma relationnel. Le résultat de cette contrainte est tous les ensembles d'articles tel que le prix minimum des articles dans chaque ensemble est égal à v.

A ce niveau aussi, nous allons voir comment étendre les requêtes DML en utilisant ces contraintes.
Une requête DML se constitue des variables schéma et d'une formule DML. Cette dernière représente le prédicat que doit vérifier les variables schéma. En utilisant les fonctions d'agrégat et les opérateurs ensemblistes, on pourra exprimer aussi d'autres contraintes que

doit vérifier les variables schéma. Ainsi une requête DML pourra avoir la forme suivante :

$$\{\mathcal{X}_1, ..., \mathcal{X}_n \mid \delta \wedge C\}$$

Tel que C est une conjonction de contraintes, définies en utilisant des fonctions d'aggrégat et opérateurs ensemblistes, et de δ qui est une formule DML.

Nous avons donc :

$$C = \wedge_{i=1}^{k} C_i$$

et $\forall i, C_i = ag(\mathcal{X})\theta v$ ou $C_i = ag(\mathcal{X}.A)\theta v$ sont des formules DML tel que v est une constante et $\theta \in \{<, \leq, >, \geq, =, \neq\}$

2. **Niveau sémantique :**

   Nous avons porté aussi des modifications au niveau de la sémantique DML. L'évaluation de l'ensemble des contraintes C et d'une formule DML exprimés dans une requête DML s'effectuera avec la même interprétation, nous avons donc :

   $$((\Sigma, R), \sigma) \models \delta \wedge C \text{ ssi } ((\Sigma, R), \sigma) \models \delta \wedge ((\Sigma, R), \sigma) \models C$$

   La valeur qu'aura une fonction d'aggrégat une fois calculée sur une variable schéma dépendra de la valeur attribuée à la variable schéma et de la relation sur laquelle s'effectuera l'évaluation de la requête. Comme nous avons deux types de fonctions, nous aurons alors :

   $$((R, \Sigma), \sigma) \models f(\mathcal{X})\theta v \text{ ssi } f(\Sigma(\mathcal{X}))\theta v$$

   ou bien :

   $$((R, \Sigma), \sigma) \models f(\mathcal{X}.A)\theta v \text{ ssi } f(X_1.A, X_2.A, ..., X_n.A)\theta v, \forall i, X_i \in \Sigma(\mathcal{X})$$

5.2.2 Langage utilisateur

Le langage de requêtes data mining utilisateur que nous définissons est déclaratif, il doit permettre à l'utilisateur de formuler ses requêtes data mining et de spécifier différentes contraintes sur les motifs à extraire. Le langage doit être aussi indépendant de tout problème, il doit être capable d'exprimer une classe importante des problèmes d'extraction de motifs intéressants ayant la représentation ensembliste.

Le langage doit permettre à l'utilisateur de spécifier les différents paramètres nécessaires pour l'exécution d'un algorithme de la librairie iZi :

1. **Les variables schéma**

   La réponse à une requête data mining de la classe des problèmes considérés est l'ensemble des valeurs que peut prendre la ou les variables schéma. L'utilisateur doit spécifier alors dans sa requête les variables schéma, cela permet d'identifier ces variables. Ce dernier est important car c'est sur ces variables que seront exprimées des contraintes.

2. **L'espace de recherche**

   L'utilisateur peut spécifier l'espace de recherche dans lequel s'effectuera la recherche des motifs intéressants.

3. **Le prédicat**

   L'utilisateur doit spécifier le prédicat qui vérifie si un motif est intéressant ou pas. Ce prédicat sera défini sur les variables schéma spécifiées précédemment.

4. **La source des données**

   L'utilisateur doit spécifier dans sa requête la source de données sur laquelle s'effectuera l'extraction.

5. **La sortie**

   Une fois un algorithme est exécuté, le résultat sera sauvegardé dans l'endroit spécifié par l'utilisateur.

Pour définir le langage nous avons précédé comme SQL. Nous avons défini une clause pour chaque point précisé ci-dessus.

Select $\mathcal{X}_1, \mathcal{X}_2, ..., \mathcal{X}_n$
From source
With $\mathcal{X}_1$ **In**... , $\mathcal{X}_2$ **In**..., ... ,$\mathcal{X}_n$ **In**...
Where Pred$(\mathcal{X}_1, \mathcal{X}_2, ..., \mathcal{X}_n)$
To sortie

Nous allons présenter quelques exemples de requêtes data mining, exprimées avec le langage défini ci-dessus, pour certains problèmes :

Select $\mathcal{X}$
From transaction
With $\mathcal{X}$ **In** $\mathcal{P}$(transaction)

Where $freq(\mathcal{X}, support\_min)$ and $sum(\mathcal{X}.prix) \leq 1200$
To $result\_1$

Cette requête consiste à extraire les itemsets fréquents, tel que $support\_min$ est le support minimum, dans des transactions enregistrées de telle manière que la somme du prix total de tous les articles ne dépasse pas 1200 unités prix d'achat. L'espace de recherche est l'ensemble des parties des articles figurant dans les transactions.

Select $\mathcal{X}$
From relation
With $\mathcal{X}$ **In** $\mathcal{P}$(relation)
Where $Key(\mathcal{X})$
To $result\_2$

Cette requête consiste à extraire toutes les clés vérifiées dans une base de données relationnelle, le prédicat de sélection $key(\mathcal{X})$ vérifie si $\mathcal{X}$ est une clé dans la relation "relation". L'espace de recherche est l'ensemble des parties du schéma de la base de données relationnelle.

Select $\mathcal{X}$
From $student\_relation$
With $\mathcal{X}$ **In** $\mathcal{P}(student\_relation \setminus "name")$
Where $FD(\mathcal{X}, "name")$
To $result\_3$

Cette requête permet d'extraire les dépendances fonctionnelles canoniques à partir de la relation $student\_relation$, le prédicat de sélection $FD(\mathcal{X}, "name")$ indique que le prédicat consiste à vérifier si un ensemble d'attributs $\mathcal{X}$ est en dépendance fonctionnelle avec l'attribut "name". L'espace de recherche est l'ensemble des parties du schéma de la base de données relationnelle sans l'attribut "name".

Select $\mathcal{X}^{(2)}$
From $relation\_1, relation\_2$

With $\mathcal{X}^{(2)}$ **In** $\mathcal{P}(relation\_1 \times relation\_2)$

Where $ID(\mathcal{X}^{(2)})$

To $result\_4$

La requête suivante permet d'extraire les dépendances d'inclusion à partir des deux relations $relation\_1$ et $relation\_2$. L'espace de recherche comprend des ensembles de couples (variable schéma d'arité 2) dont le premier élément est un attribut du schéma de la relation $relation\_1$ et le deuxième élément est un attribut de la relation $relation\_2$. Le prédicat vérifiera l'inclusion des attributs pour chaque couple d'attributs.

5.2.3 Détermination de la forme logique d'une requête

Dans cette section nous allons étudier la forme logique d'une requête DML. Pour cela nous avons les deux cas de figure :

1. Une requête DML comme définie dans [2] est sous la forme : $\{\mathcal{X}_1, ..., \mathcal{X}_m | \delta\}$, la formule DML δ constitue le prédicat qui déterminera si un motif est intéressant ou pas. Nous avons vu dans ce cas comment identifier sa forme logique (positivité) ainsi l'algorithme Apriori ou ABS pourra être utilisé pour le parcours de l'espace de recherche.
2. Dans le cas où la requête DML est constituée aussi de contraintes, avec fonctions d'aggrégat ou opérateurs ensemblistes, alors la requête a la forme suivante : $\{\mathcal{X}_1, ..., \mathcal{X}_n \mid \delta \wedge C\}$. Dans ce cas, le prédicat est une conjonction d'une formule DML et de contraintes ou bien que des contraintes. On étudie la forme logique de la formule DML comme expliqué ci-dessus et la forme des contraintes comme indiquée dans le tableau 4.1. Le prédicat peut être sous forme de :
   - Conjonction de contraintes : l'algorithme CAP est appliqué mais sans la contrainte de fréquence.
   - Conjonction de contraintes et formule DML : si la formule DML est anti-monotone alors l'algorithme CAP peut être utilisé mais en remplaçant le prédicat de fréquence qui est anti-monotone par la formule DML anti-monotone.

5.2.4 Génération d'un plan physique

Une fois la forme logique d'une requête DML est étudiée et l'algorithme de parcours est choisi en fonction de cette forme, un plan physique sera généré. L'idée est d'utiliser les structures de

données et algorithmes génériques de la librairie iZi.
Pour commencer, il faut instancier une instance de la classe de l'algorithme choisi pour l'évaluation de la requête. Puis paramétrer ce dernier en lui passant les arguments nécessaires pour l'exécution. Comme préciser dans la figure 2.1, jusqu'à cinq argument à passer pour l'instance algorithme : données en entrée, données en sortie, initialisation du langage, fonction de transformation des motifs du langage et le prédicat de sélection. Une fois le plan est généré, ce dernier sera compilé puis exécuté et les résultats d'exécution constitueront une réponse à la requête utilisateur émise.
Le code source généré pour répondre à la requête en question et l'optimisation appliquée sont obtenus d'une manière transparente à l'utilisateur. Ce dernier ne s'occupe qu'à déclarer ses besoins en spécifiant les données sur lesquelles s'effectuera l'extraction et les contraintes de sélection.

5.3 Conclusion

Dans ce chapitre nous avons proposé une approche d'optimisation des requêtes data mining pour la classe de problèmes d'extraction de motifs intéressants ayant la représentation ensembliste.
Pour cela nous avons proposé un prototype d'une couche syntaxique que pourra utiliser un utilisateur pour formuler sa requête. Nous avons démontré aussi comment la logique DML pourra être utilisée comme couche intermédiaire pour étudier les formes logiques des requêtes data mining et ainsi de choisir l'algorithme d'évaluation qui intègre des techniques d'optimisation associée à la forme logique. Nous avons enrichi DML en introduisant une classe importante de contraintes ayant des propriétés intéressantes pour l'optimisation. Un plan physique sera par la suite généré en utilisant des structures de données et algorithmes génériques de la librairie iZi.
Cette approche est déclarative, l'utilisateur ne s'occupe ni des détails d'optimisation ni d'implémentation. Elle permet aussi de répondre à n'importe quel problème appartenant à la classe des problèmes considérés en utilisant des algorithmes génériques intégrant des techniques d'optimisation.

Conclusion et perspectives

Dans ce travail, nous avons étudié un domaine de recherche récent et intéressant, qui est l'une des dix disciplines émergeantes du 21^{ime} siècle qui est le data mining.

Le data mining constitue une discipline proposant des méthodes et techniques d'extraction de connaissances à partir d'un grand volume de données. Les connaissances extraites seront ainsi utilisées dans différentes applications liées à différents domaines comme le marketing, l'astronomie, les laboratoires pharmaceutiques...etc.

Dans ce mémoire, nous avons proposé un langage requêtes pour une classe de problèmes d'extraction de motifs intéressants dits représentables par des ensembles. La base de données sur laquelle s'effectue l'extraction est importante en nombre de lignes et de colonnes. L'espace de recherche est aussi important et un parcours séquentiel de tous ses candidats n'est pas intéressant, il est nécessaire alors d'introduire des techniques d'optimisation permettant de réduire le nombre de tests à effectuer et ainsi les accès mémoire. L'approche d'optimisation proposée utilise la logique DML pour étudier les propriétés des requêtes. Une fois les propriétés des requêtes sont étudiées, une technique est choisie et les structures de données et algorithmes d'exploration de l'espace de recherches génériques de la librairie iZi sont utilisés pour générer des plans physiques aux requêtes émises par l'utilisateur. Nous avons étendu la logique DML avec une classe importante de contraintes (avec fonctions d'aggrégat et opérateurs ensemblistes), ayant une propriété intéressante permettant l'élagage de l'espace de recherche, dites succinctes.

Au cours de ce travail, nous avons proposé également un algorithme pour l'extraction de la couverture canonique des dépendances fonctionnelles. L'algorithme est développé conformément à la méthodologie proposée dans le projet iZi et implémenté en utilisant la bibliothèque iZi. A travers ce travail, nous avons constaté la facilité et la générécité de la librairie iZi, cette dernière

offre différentes classes facilitant l'implémentation de solutions à la classe des problèmes considérés. Ce travail a fait l'objet d'un papier, intitulé " inférence des dépendances fonctionnelles", soumis et présenté à ISOR'08 -International Symposium On Operational Research-, novembre 2008, Alger, Algérie.

Le prédicat de sélection de motifs exprimé dans une requête est sous forme d'une conjonction. Pour cela, nous avons trois cas : prédicat monotone, anti-monotone ou une conjonction de contraintes anti-monotones et/ou et succinctes sur une seule variable schéma. Comme perspectives, nous envisageons d'une part d'introduire d'autres techniques prenant en considération le cas où le prédicat est une conjonction de contraintes monotones et/ou succinctes et d'une autre part d'introduire des contraintes avec fonction d'aggrégat mettant en jeu plus d'une variable schéma.

Bibliographie

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93 : Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, New York, NY, USA, 1993. ACM.

[2] T. Calders and J. Wijsen. On monotone data mining languages. In *Proc. Eighth Int. Workshop on Databases and Programming Languages, DBPL 2001*, pages 119–132. Springer, 2001.

[3] Usama Fayyad, Gregory Piatetsky-shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17 :37–54, 1996.

[4] Usama Fayyad, Gregory Piatetsky-shapiro, and Padhraic Smyth. Knowledge discovery and data mining : Towards a unifying framework. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining*, pages 82–88. AAAI Press, 1996.

[5] Frédéric Flouvat. *Vers des solutions adaptatives et génériques pour l'extraction de motifs intéressants dans les données*. PhD thesis, 2006.

[6] Frédéric Flouvat, Fabien De Marchi, and Jean-Marc Petit. ABS : adaptative borders search of frequent itemsets. In Roberto J. Bayardo Jr., Bart Goethals, and Mohammed Javeed Zaki, editors, *2nd workshop of frequent itemsets mining implementation (FIMI'04)*, volume 126 of *CEUR Workshop Proceedings*, pages 1–8. CEUR-WS.org, 2004.

[7] Frédéric Flouvat, Fabien De Marchi, and Jean-Marc Petit. Vers le prototypage rapide de programmes de fouille de données. In Dominique Laurent, editor, *Bases de Données Avancées*, BDA, pages 1–18, 2006.

[8] Frédéric Flouvat, Fabien De Marchi, and Jean-Marc Petit. Towards easy prototyping of pattern mining problems. In L. Saïs, editor, *Colloque sur l'Optimisation et les Systèmes d'Information (COSI'07)*, pages 611–623, 2007.

[9] Jiawei Han, Yongjian Fu, Wei Wang, Krzysztof Koperski, and Osmar Zaiane. Dmql : A data mining query language for relational databases. pages 27–33, 1996.

[10] Tomasz Imielinski and Heikki Mannila. A database perspective on knowledge discovery. *Commun. ACM*, 39(11) :58–64, 1996.

[11] René Lefébure and Gilles Venturi. *Data mining Gestion de la relation client Personnalisation de sites web*. Eyrolles edition, 2001.

[12] Heikki Mannila and Hannu Toivonen. Multiple uses of frequent sets and condensed representations. In *In Proc. KDD Int. Conf. Knowledge Discovery in Databases*, pages 189–194, 1996.

[13] Heikki Mannila and Hannu Toivonen. Levelwise search and borders of theories in knowledgediscovery. *Data Min. Knowl. Discov.*, 1(3) :241–258, 1997.

[14] Rosa Meo. Optimization of a language for data mining. In *SAC '03 : Proceedings of the 2003 ACM symposium on Applied computing*, pages 437–444, New York, NY, USA, 2003. ACM.

[15] Rosa Meo, Giuseppe Psaila, and Stefano Ceri. A new sql-like operator for mining association rules. In *VLDB*, pages 122–133, 1996.

[16] Raymond T. Ng, Laks V. S. Lakshmanan, Jiawei Han, and Alex Pang. Exploratory mining and pruning optimizations of constrained association rules. In *SIGMOD Conference*, pages 13–24, 1998.

[17] Jean-Marc Petit, Abderaouf Kouhoul, and Ryme Chelouah. Inférence des dépendances fonctionnelles. In *International Symposium On Operational Research (ISOR'08)*, 2008.

[18] Zhaohui Tang, Jamie Maclennan, and Peter Pyungchul Kim. Building data mining solutions with ole db for dm and xml for analysis. *SIGMOD Rec.*, 34(2) :80–85, 2005.

[19] Stéphane Tufféry. *Data mining et scoring, Base de données et gestion de la relation client.* Dunod edition, 2002.

Annexe A

Inférence des dépendances fonctionnelles

International Symposium on Operational Research, ISOR'08
Algiers, Algeria : November 2-6, 2008

Inférence des dépendances fonctionnelles

Jean-Marques PETIT [1] **Abderaouf KOUHOUL** [2]
Ryme CHELOUAH [2]

[1] Laboratoire LIRIS, UMR CNRS 5205,
Université Lyon I INSA Lyon, 69 621 Villeurbanne, France
jmpetit@liris.cnrs.fr

[2] Université A.Mira de Bejaia, Département d'Informatique,
Bejaia, Algérie
{kouhoul_abderaouf,ryme_chelouah}@yahoo.fr

Abstract: The task of discovering of functional dependencies within great data bases is one of the data mining problems of discovering interesting patterns. This problem has been a subject of several research works[7][8][6]. In this paper, we have proposed an algorithm for extracting the canonical closure of functional dependencies. The algorithm was designed according to the iZi project methodology and it was developed under the iZi library.

Keywords: Data mining, functional dependencies, canonical closure, positif bordure.

Résumé : La découverte des dépendances fonctionnelles dans les bases de données relationnelles est l'un des problèmes data mining d'extraction de motifs interessants. Ce problème a fait l'objet de plusieurs travaux de recherche [7][8][6]. Dans ce papier, nous proposons un algorithme d'extraction de la couverture canonique des dépendances fonctionnelles. L'algorithme est développé conformément à la méthodologie proposée dans le projet iZi et implémenté en utilisant la bibliothèque iZi.

Mots clés : Data mining, dépendance fonctionnelle, couverture canonique, bordure positive.

---

1 Introduction

Les dépendances fonctionnelles sont des contraintes d'intégrité très importantes; elles représentent une implication entre deux ensembles d'attributs. La couverture des dépendances fonctionnelles peut être élaborée lors de la phase conception. Cependant, l'évolution temporelle de la taille de la base de données peut soulever le problème de la détermination de l'ensemble des dépendances fonctionnelles satisfaites. Ce problème a fait l'objet de plusieurs travaux de recherche [7][8][6]. Dans cet article, nous avons proposé un algorithme pour l'inférence des dépendances fonctionnelles de la couverture canonique. L'implémentation de cet algorithme a été réalisée en utilisant la bibliothèque iZi.

2 Problème de découverte de la couverture canonique des dépendances fonctionnelles

Soit r une relation définie sur le schéma relationnel R. Soient X, Y deux sous ensembles d'attributs de R.
On dit que X et Y sont en dépendance fonctionnelle, notée $X \longrightarrow Y$ si et seulement si :

$$\forall u, v \in r, u \neq v, u[X] = v[X] \Longrightarrow u[Y] = v[Y]$$

La dépendance fonctionnelle $X \longrightarrow Y$ est dite canonique si et seulement si Y se réduit en un seul attribut.

Un ensemble F de dépendances fonctionnelles est dit minimal si et seulement si :

$$\forall f \in F, (F - f) \neq F$$

Le problème d'inférence des dépendances fonctionnelles consiste à la recherche de toutes les dépendances fonctionnelles élémentaires canoniques satisfaites dans r.

On note par CC la couverture canonique des dépendances fonctionnelles de la relation r si et seulement si :

$\forall f \in CC, r \models f$ et f est une dépendance fonctionnelle canonique

Problématique :
Soit à découvrir les dépendances fonctionnelles ayant comme partie droite l'attribut B.

Ce problème peut se formuler en un problème ayant une représentation ensembliste suivant le cadre théorique défini par Mannila de la manière suivante [1]:

- Le langage $\mathcal{L}$ est l'ensemble des parties du schéma $R \setminus B$ ou bien $\mathcal{P}(R \setminus B)$.

- Le prédicat de sélection p(r, X) est vrai si et seulement si $r \models X \longrightarrow B$, avec $X \subseteq R \setminus B$.

- La relation d'ordre partiel est l'inclusion $\subseteq$ et le prédicat p est monotone par rapport à l'inclusion :

  Soient $X, Y, Z \subseteq R$,
  Si $X \subseteq Y$ et $X \longrightarrow Z$ alors $Y \longrightarrow Z$
  ou bien :
  $X \subseteq Y$, $r \models p(r, X) \Longrightarrow r \models p(r, Y)$

On note Th_B l'ensemble des dépendances fonctionnelles canoniques ayant l'attribut B en partie droite.

$$Th_B = \{X \subset R \setminus B \mid r \models P(r, X)\}$$

L'ensemble des dépendances fonctionnelles élémentaires canoniques est la bordure positive de la théorie Th_B.

Donc la couverture canonique est :

$$CC = \cup_{A_i \in R} Bd^+_{A_i}$$

3 Algorithme proposé

Notre algorithme a été conçu conformément à la méthodologie proposée dans le projet iZi [2][3]. L'algorithme permet l'inférence de la couverture canonique des dépendances fonctionnelles satisfaites dans une base de données relationnelle.

Algorithme :

ENTREE : Une base de données relationnelle r définie sur le schéma relationnel R
$BP^+_{A_i}$ /* Bordure positive relative à l'attribut A_i */
SORTIE : la couverture canonique CC
DEBUT
$CC \longleftarrow \oslash$
Récuperer la liste des attributs R de la base de données r.
Pour chaque attribut $A_i \in R$ faire

Instanciation d'une variable prédicat P "est une dépendance fonctionnelle ayant A_i comme partie droite"

Initialiser l'espace de recherche /* init(r, A_i) */

Apriori(init, r, P, $BP^+_{A_i}$)

$CC \longleftarrow CC \cup BP^+_{A_i}$

Fin pour
FIN.

L'algorithme proposé est itératif, chaque itération consiste à fixer la partie droite de la dépendance fonctionnelle par un attribut (couverture canonique), initialiser l'espace de recherche sans tenir compte de l'attribut fixé en partie droite puis parcourir l'espace de recherche en utilisant Apriori et tester les candidats générés par ce dernier. Pour cela, nous avons implémenté aussi un composant important qui est le prédicat. C'est ce composant qui permet d'évaluer si une dépendance est vérifiée dans la base de données. En sortie, l'algorithme calcule la bordure positive qui correspond à l'ensemble des dépendances fonctionnelles canoniques élémentaires.

4 Implémentation

4.1 Projet iZi

Le projet iZi s'est concentré aux problèmes d'extraction de motifs intéressants et plus particulièrement aux problèmes représentables par des ensembles dont l'objectif est double [2][3] :

- Rendre plus facile les implémentations de fouille de données en proposant une boite noire de structures de données et d'algorithmes (Bibliothèque iZi).
- Guider l'utilisateur lors de l'implémentation.

Au cours de ce projet, une librairie en C++ appelée iZi a été proposée pour la résolution des problèmes d'extraction de motifs intéressants [2][3]. L'idée est d'offrir une boite noire permettant le développement rapide de programmes utilisant un algorithme préalablement choisi. La librairie comprend cinq composants. Le composant ALGORITHM représente les implémentations des différents algorithmes disponibles dans la librairie iZi et qui peuvent être utilisés pour résoudre différents problèmes de fouille de données. Les autres composants (prédicat,...) sont spécifiques à un problème donné et deviennent des paramètres au composant ALGORITHM. Ainsi l'utilisateur doit les développer comme il peut aussi les réutiliser pour résoudre d'autres problèmes.
L'autre objectif de ce projet est de guider l'utilisateur, lors de l'implémentation, à utiliser la librairie proposée. Pour cela, une méthodologie a été proposée aussi [2][3], consistant à aider l'utilisateur à reformuler si possible son problème sous la forme d'un problème d'extraction de motifs intéressants et l'assiste aussi dans le développement des différents composants. La méthodologiè iZi se résume en :

4.1.1 Définir l'espace de recherche :

L'utilisateur à ce niveau doit d'abord déterminer le langage qui permet d'exprimer les motifs de l'espace de recherche et une relation d'ordre partiel sur ces motifs.

4.1.2 Définir le prédicat :

Définir le prédicat revient à exprimer d'une manière formelle ce que veut dire "intéressant".

4.1.3 Identifier la sortie :

Il suffit de caractériser quels motifs seront donnés en sortie en fonction de l'ensemble des motifs intéressants.

4.1.4 Exhiber une représentation ensembliste :

De nombreux problèmes s'expriment déjà de façon ensembliste donc cette étape est triviale. Mais quand c'est pas le cas, cette étape est nécessaire.

4.2 Expérimentation

Pour l'experimentation de l'algorithme propsé nous avons utilisé deux jeux de données ayant le format CSV.

Lors de l'expérimentation, nous avons expérimenter l'algorithme sur les jeux de test suivants :

Test 1 :
a;b;c;d;e;f;g
5;4;1;1;1;1;1
2;2;2;2;2;2;2
3;3;5;3;3;3;3
4;4;4;4;4;4;4
5;5;5;5;5;5;1

Les résultat d'éxecution de l'algorithme proposé sur ce jeu de données est :

$Bd^+ = \{bc \longrightarrow a, d \longrightarrow a, e \longrightarrow a, f \longrightarrow a, g \longrightarrow a, cg \longrightarrow b, ca \longrightarrow b, d \longrightarrow b,$
$e \longrightarrow b, f \longrightarrow b, bg \longrightarrow c, ba \longrightarrow c, d \longrightarrow c, e \longrightarrow c, f \longrightarrow c, bc \longrightarrow d, bg \longrightarrow d,$
$ba \longrightarrow d, cg \longrightarrow d, ca \longrightarrow d, e \longrightarrow d, f \longrightarrow d, bc \longrightarrow e, bg \longrightarrow e, ba \longrightarrow e,$
$cg \longrightarrow e, ca \longrightarrow e, d \longrightarrow e, f \longrightarrow e, bc \longrightarrow f, bg \longrightarrow f, ba \longrightarrow f, cg \longrightarrow f,$
$ca \longrightarrow f, d \longrightarrow f, e \longrightarrow f, bc \longrightarrow g, d \longrightarrow g, e \longrightarrow g, f \longrightarrow g, a \longrightarrow g\}$

Test 2 :
a;b;c;d;e
5;4;1;6;1
2;2;2;2;2
3;3;5;3;9
4;4;4;4;4
5;3;9;6;4

Les résultat d'éxecution de l'algorithme proposé sur ce jeu de données est :

$Bd^+ = \{c \longrightarrow a, d \longrightarrow a, be \longrightarrow a, c \longrightarrow b, de \longrightarrow b, ea \longrightarrow b, bd \longrightarrow c,$
$be \longrightarrow c, ba \longrightarrow c, de \longrightarrow c, ea \longrightarrow c, c \longrightarrow d, a \longrightarrow d, be \longrightarrow d,$
$c \longrightarrow e, bd \longrightarrow e, ba \longrightarrow e\}$

5 Conclusion

Dans ce papier nous avons proposé un algorithme pour l'inférence des dépendances fonctionnelles dans une base de données relationnelles. Notre algorithme est conçu en se basant sur la méthodologie iZi et est implémenté en utilisant la bibliothèque iZi. L'algorithme de parcours de l'espace de recherche utilisé est Apriori. Ce qui nous permet d'avoir en sortie la bordure positive donc l'ensemble des dépendances fonctionnelles élémentaires canoniques. Cependant, on peut utiliser un autre algorithme de parcours disponible dans la bibliothèque iZi qui est ABS -Adaptive Borders Search- [4][5]. En perspectives, nous envisageons d'étudier la complexité de l'algorithme proposé et le comparer avec d'autres implémentations.

References

[1] Heikki Mannila and Hannu Toivonen. Levelwise search and boders of theories in knowledge discovery. Data Min. Knowl. Discov, 1(3) :241– 258, 1997.

[2] Frédéric Flouvat, Fabien De Marchi, and Jean-Marc Petit. Vers le prototypage rapide de programmes de fouille de données. In Dominique Laurent, editor, DBA, 2006.

[3] Frédéric Flouvat, Fabien De Marchi, and Jean-Marc Petit. Towards easy prototyping of pattern mining problems. COSI'07, pages 611– 623, 2007.

[4] Frédéric Flouvat. Vers des solutions adaptatives et génériques pour l'extraction de motifs intéressants dans les données. PhD thesis, 2006.

[5] Frédéric Flouvat, Fabien De Marchi, and Jean-Marc Petit. Abs: Adaptive borders search of frequent itemsets. In FIMI, 2004.

[6] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. Efficient discovery of functional and approximate dependencies using partitions. In Proceedings of the Fourteenth International Conference on Data Engineering, February 23– 27, 1998, Orlando, Florida, USA, pages 392– 401. IEEE Computer Society, 1998.

[7] Yahiko Kambayashi, Werner Winiwarter, and Masatoshi Arikawa, editors. Data Warehousing and Knowledge Discovery, Third International Conference, DaWaK 2001, Munich, Germany, September 5– 7, 2001, Proceedings, volume 2114 of Lecture Notes in Computer Science. Springer, 2001.

[8] Peter A. Flach and Iztok Savnik. Database dependency discovery : A machine learning approach. AI Commun, 12(3): 139– 160, 1999.

Résumé

Le développement de méthodes et de techniques pour les problèmes data mining et le succès des bases de données grâce à la simplicité des langages déclaratifs comme SQL et des techniques d'optimisation, ont emmené à la proposition de langages requêtes pour exprimer différentes tâches data mining. Ces derniers s'avèrent très spécifiques, c'est une couche syntaxique autour d'un algorithme data mining.

Dans ce mémoire, nous avons proposé une approche déclarative pour répondre à des requêtes data mining. Nous avons proposé un langage déclaratif pour exprimer des requêtes pour une classe particulière de problèmes d'extraction de motifs, dits "Représentables par des ensembles". Nous avons utilisé la logique DML pour exprimer les requêtes utilisateur et étudier leurs formes logiques. La logique DML permet de d'identifier deux propriétés importantes pour l'optimisation des requêtes : monotonie / anti-monotonie, ce sont les deux propriétés qu'exploitent les algorithmes génériques de la librairie iZi. Nous avons étendu également DML avec une classe importante de contraintes, dites succinctes ayant un pouvoir d'élagage de l'espace de recherche important. Une fois les formes logiques sont étudiées, un algorithme d'évaluation de la librairie iZi est choisi et un plan physique est généré pour répondre à une requête data mining.

Mots clé : fouille de données, langage requête, optimisation, représentation ensembliste, monotonie/anti-monotonie, librairie iZi.

Abstract

The development of methods and techniques for data mining problems and the success of database, because of the simplicity of declarative languages like SQL and optimization techniques, brought to proposing query languages to express different data mining tasks. These are very specific ; it is a syntactic layer around a data mining algorithm.

In this work, we proposed a declarative approach to answer data mining queries. We proposed a declarative language to express queries for a particular class of discovering interesting patterns problems known as representable by sets. We used the DML logic to express the user queries and to study their logical forms. DML allows us to study two important properties for queries optimization : monotony / anti-monotony, which are used by the generic algorithms of the iZi

library. We have also extended the DML logic with an important class of constraint, said succinct, and having an important power of pruning of the research space. Once the logical forms are studied, an evaluation algorithm of the iZi library is chosen and a physical plan is generated to answer a data mining query.

Key words : data mining, query language, optimization, set representation, monotony/anti-monotony, iZi library.