N° d'ordre 02 ISAL 0090 Année 2002

THÈSE

présentée

DEVANT L'INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE LYON

pour obtenir

LE GRADE DE DOCTEUR

SPÉCIALITÉ INFORMATIQUE

FORMATION DOCTORALE : DISIC
ÉCOLE DOCTORALE : EDIIS

PAR

Baptiste Jeudy

(D.E.A. de l'ENS de Lyon - Université Lyon I - INSA de Lyon)

OPTIMISATION DE REQUÊTES INDUCTIVES : APPLICATION À L'EXTRACTION SOUS CONTRAINTES DE RÈGLES D'ASSOCIATION

Soutenue le : 13 décembre 2002 **devant la Commission d'Examen**

Jury

Dominique Laurent, Professeur, université de Tours,	rapporteur
Michèle Sebag, CR CNRS, HdR, université Paris Sud Orsay,	rapporteur
Lionel Brunie, Professeur, INSA de Lyon,	co-directeur
Jean-François Boulicaut, Maître de conférence, HdR, INSA de Lyon,	co-directeur
François Jacquenet, Professeur, université de Saint Étienne,	examinateur
Stefan Kramer, Assistant Professeur, université de Freiburg (Allemagne),	examinateur

Cette thèse a été préparée au Laboratoire d'Ingénierie des Systèmes d'Information (LISI) de l'INSA de Lyon

Table des matières

Résumé **7**

Résumé en anglais **9**

Introduction **11**

1 Bases de données inductives **19**
1.1 Motivations . . . 19
1.1.1 Itemsets et règles d'association . . . 19
1.1.2 Processus d'extraction de connaissances . . . 22
1.1.3 Problèmes posés par la gestion du processus d'ECD . . . 24
1.1.4 L'apport des bases de données inductives . . . 25
1.1.5 Exemples de langages existants . . . 25
1.2 Formalisation des bases de données inductives . . . 27
1.3 Formalisation des requêtes inductives . . . 27
1.3.1 Calcul de théories . . . 27
1.3.2 Requête étendue . . . 28
1.3.3 Fonctions d'évaluation . . . 29
1.3.4 Contraintes syntaxiques . . . 30
1.3.5 Contraintes d'optimisation . . . 30
1.3.6 Contraintes complexes . . . 31
1.4 Évaluation des requêtes inductives . . . 31
1.4.1 Les contraintes anti-monotones et la frontière . . . 31
1.4.2 L'algorithme Guess and correct . . . 34
1.4.3 Stratégies avancées . . . 37
1.5 Questions ouvertes . . . 40
1.5.1 Les primitives et les solveurs . . . 40
1.5.2 Vues et matérialisations . . . 41
1.5.3 Optimisation de séquences de requêtes . . . 42
1.6 Introduction de la suite du mémoire par rapport à ces problèmes . . . 43

2 Extraction sous contraintes **45**
2.1 Le cas particulier de la contrainte de fréquence minimale . . . 45
2.1.1 Généralisation au cas d'une contrainte anti-monotone quelconque . . . 48
2.1.2 Complexité d'Apriori . . . 49
2.2 Calcul direct de la frontière . . . 49
2.3 Contraintes monotones . . . 51
2.4 État de l'art sur l'utilisation des contraintes . . . 52
2.4.1 Extraction d'itemsets . . . 53

2.4.2 Extraction de séquences . . . 57
2.4.3 Fragments . . . 59
2.4.4 Requêtes datalog . . . 59
2.4.5 Autres types de motifs . . . 61
2.5 Discussion sur l'état de l'art . . . 61
2.5.1 Génération des candidats . . . 61
2.5.2 Critère d'arrêt de l'algorithme . . . 64
2.5.3 Élagage . . . 65
2.5.4 Conclusion sur l'état de l'art . . . 67
2.6 Extraction d'itemsets sous contraintes anti-monotone et monotone . . . 68
2.6.1 Caractérisation par les frontières . . . 68
2.6.2 Algorithme générique . . . 70
2.6.3 Calcul de la frontière inférieure . . . 73
2.6.4 Utiliser les contraintes monotones sans élaguer . . . 74
2.6.5 Disjonctions . . . 74
2.6.6 Contraintes faiblement anti-monotones . . . 75
2.7 Différentes stratégies d'évaluation de requêtes inductives . . . 76
2.7.1 Des contraintes de règles aux contraintes d'itemsets . . . 76
2.7.2 Pousser les contraintes anti-monotones . . . 76
2.7.3 Utiliser seulement les contraintes syntaxiques . . . 77
2.7.4 Utilisation d'autres contraintes . . . 78
2.7.5 Conclusion . . . 79

3 Extraction de représentations condensées sous contraintes 81
3.1 Diverses formes de représentations condensées . . . 81
3.1.1 Itemsets fermés . . . 81
3.1.2 Algorithmes d'extraction des itemsets fermés . . . 83
3.1.3 Représentations condensées utilisant les fermés . . . 90
3.1.4 Représentations condensées utilisant les libres . . . 91
3.1.5 Règles non redondantes . . . 93
3.1.6 Les itemsets δ-libres . . . 94
3.1.7 Les itemsets $\vee$-*libres* . . . 96
3.1.8 Discussion . . . 97
3.2 Extraction des representations condensées sous contraintes . . . 98
3.2.1 Cas des contraintes anti-monotones . . . 98
3.2.2 Cas des contraintes monotones . . . 100
3.3 Nouvelles stratégies d'évaluation de requêtes inductives . . . 105
3.4 Extraction de règles avec négations . . . 106
3.4.1 Itemsets généralisés et règles avec négations . . . 106
3.4.2 Les bases de données binaires . . . 107
3.4.3 Différentes stratégies . . . 108
3.4.4 Résultats expérimentaux . . . 109
3.4.5 Étude expérimentale de l'erreur en fonction de δ . . . 110
3.5 Application à l'étude de données d'expression de gènes . . . 111
3.5.1 Un peu de biologie . . . 112
3.5.2 Données brutes . . . 112
3.5.3 Pré-traitements . . . 113
3.5.4 Extraction dans le contexte booléen . . . 114
3.5.5 Post-traitement et interprétation des résultats . . . 115

3.5.6 Perspectives 116
3.6 Conclusion 117

4 Optimisation d'une séquence de requêtes 119
4.1 Introduction 119
4.2 Comparaison de deux techniques d'évaluation d'une séquence de requêtes . . 119
4.3 Travaux existants 121
4.3.1 Discussion 122
4.4 Description de notre approche 123
4.5 Algorithme iCoCo 123
4.5.1 Contenu du cache 128
4.6 Expérimentations 129
4.6.1 Efficacité de l'utilisation du cache 130
4.6.2 Taille du cache 130
4.7 Nouvelles stratégies d'évaluation de requêtes inductives 132

Conclusion 135

A Intégration de l'élagage de certains libres dans Close 141

Remerciements 153

Résumé

L'utilisation de plus en plus généralisée de l'informatique permet de récolter de plus en plus de données de manière automatique, que ce soit en sciences (biologie, astronomie, etc.) ou dans le commerce (en particulier sur internet). L'analyse de telles quantités de données est problématique. Les techniques d'extraction de connaissances dans les données (ECD) ont été conçues pour répondre à ce besoin.

Dans cette thèse, nous avons utilisé la notion de base de données inductive comme cadre pour notre travail. Une base de données inductive est une généralisation des bases de données classiques dans lesquelles sont stockées non seulement des données mais aussi des propriétés apprises sur ces données. On peut alors voir le processus d'ECD comme l'interrogation d'une base de données inductive. Nous avons particulièrement étudié l'optimisation des requêtes inductives portant sur l'extraction des règles d'association et des itemsets. Dans ce cas, l'utilisateur peut préciser les règles ou les itemsets qui l'intéressent en utilisant des contraintes. Ces dernières peuvent, par exemple, porter sur la fréquence ou imposer des restrictions syntaxiques sur les itemsets ou les règles à extraire.

Nous avons proposé différentes stratégies d'évaluation des requêtes d'extraction de règles ou d'itemsets en utilisant efficacement les contraintes (en particulier les contraintes dites monotones et anti-monotones). Nous avons également étudié l'apport des représentations condensées dans l'optimisation de l'évaluation de ces requêtes et nos expériences montrent que l'utilisation simultanée des contraintes et des représentations condensées donne de très bons résultats. Nous avons aussi montré comment utiliser les représentations condensées comme caches pour l'optimisation de séquences de requêtes. Ici encore, les résultats sont bons et l'utilisation des représentations condensées permet d'obtenir des caches remarquablement petits.

Résumé en anglais *Abstract*

The increasingly generalized use of computers makes it possible to collect more and more data in an automatic way, e.g. in sciences (biology, astronomy, etc.) or in the trade (Internet). The analysis of such quantities of data is problematic. Knowledge Discovery in Databases (KDD) techniques were conceived to meet this need.

In this thesis, we used the concept of inductive databases as a framework for our work. An inductive database is a generalization of the traditional databases in which the user can query not only the data but also properties learned on the data. One can then see the whole KDD process as the interrogation of an inductive database.

In this thesis, we particularly studied the optimization of inductive queries relating to the extraction of association rules and itemsets. In this case, the user can specify the rules or the itemsets of interest by using constraints. These constraints can, e.g., specify a frequency threshold or impose syntactic restrictions on the itemsets or the rules.

We propose various strategies for the evaluation of rules and itemsets extraction queries by effectively using the constraints (in particular constraints known as monotonic and anti-monotonic). We studied the use of condensed representations in the optimization of the evaluation of these requests and our experiments show that the simultaneous use of the constraints and the condensed representations gives very good results. We also show how to use condensed representations as a cache for optimization of sequences of queries. Here still, the results are good and the use of the condensed representations makes it possible to obtain much smaller caches than with previous techniques.

Introduction

Contexte. L'utilisation de plus en plus généralisée de l'informatique permet de récolter de plus en plus de données de manière automatique. Ainsi, en sciences expérimentales, les mesures, autrefois faites à la main, sont recueillies par des appareils qui les enregistrent directement sous forme informatique. Ce n'est pas le seul domaine touché. Du fait de l'informatisation rapide des administrations, des entreprises, du commerce, des télécommunications, la quantité de données disponibles sous forme informatique augmente très rapidement. Cependant, l'analyse et l'exploitation de ces données reste très difficile. Dans les sciences expérimentales, il n'est pas rare de consacrer plusieurs années à l'analyse des résultats d'une expérience. Il suffit de songer aux données issues de recensements, d'enquêtes épidémiologiques, d'observations spatiales, du décryptage du génome, etc. Dans le cas des données récoltées automatiquement issues du commerce (en particulier électronique), des télécommunications, ... on ne sait pas toujours comment les exploiter une fois qu'elles ont remplit leur rôle principal (par exemple la facturation). Pourtant, les propriétaires de ce type de données hésitent souvent à les détruire pour au moins deux raisons. D'une part le coût de stockage et donc de la conservation de ces données a très fortement baissé et, d'autre part, ils supposent que ces données contiennent peut être des connaissances d'une grande valeur (par ex. sur le comportement de leurs clients, sur la survenue d'erreurs dans un réseau de communication...). C'est en fait le postulat principal qui motive l'extraction de connaissances dans les données (ECD) : ces masses de données contiennent sûrement des connaissances d'une grande valeur commerciale ou scientifique. Il faut noter que ce postulat n'a évidemment pas été "démontré". Même expérimentalement, il n'est pas sûr qu'il soit toujours vérifié. Rien n'indique qu'il y ait autre chose dans les données que des phénomènes triviaux (i.e., connus par tout expert du domaine) et du bruit statistique.

Processus d'extraction de connaissances. Une fois ce postulat admis, la question se pose de savoir comment des connaissances peuvent être extraites de ces données. Cela ne peut pas être fait directement par un opérateur humain. En effet, le cerveau humain n'est pas adapté pour traiter de telles masses de données. Par contre, seul un expert humain est capable d'évaluer si le résultat d'une analyse des données apporte vraiment une nouvelle connaissances. Cette évaluation fait en effet intervenir tellement de paramètres subjectifs (connaissances du domaine, contexte économique ou scientifique ...) qu'il n'est pas possible de la faire automatiquement.

Le processus d'extraction de connaissances ne se limite donc pas à une extraction automatique, il comporte plusieurs étapes pendant lesquelles l'expert humain a un rôle important. Il faut tout d'abord récupérer les données qui peuvent être issues de plusieurs sources différentes et les mettre dans un format commun pour pouvoir les fusionner. Il faut ensuite pré-traiter les données, par exemple résoudre le problème posé par les valeurs manquantes ou aberrantes, et sélectionner les données sur lesquelles vont être appliqué l'algorithme d'ex-

traction de connaissances proprement dit. Ensuite, il faut post-traiter les résultats de cet algorithme et les interpréter. Ce post-traitement peut consister à sélectionner les résultats les plus prometteurs, à les trier ou à vérifier leur pertinence à l'aide d'outils statistiques.

Dans chacune de ces étapes, l'expert doit faire des choix et évaluer les résultats obtenus en fonction de ses objectifs. Il peut alors décider de passer à l'étape suivante ou de recommencer une ou plusieurs des étapes précédentes en utilisant une technique différente. Ce processus est donc naturellement itératif et interactif.

Différentes techniques d'extraction de connaissances. On peut regrouper les différentes techniques d'ECD en deux catégories : l'extraction de motifs locaux et la construction de modèles des données. Les premières techniques visent à extraire des propriétés concernant des sous ensembles des données alors que les techniques de construction de modèles sont globales et cherchent à mettre en évidence des propriétés de l'ensemble des données.

Dans la figure 1, nous donnons un premier exemple de motif local très étudié introduit en 1993 [AIS93], les *règles d'association*. Les règles d'association sont extraites dans des matrices booléennes (ou bases de données binaires). Chaque règle est formée d'un antécédent ou corps (dans la figure, corps de la première règle est constitué des attributs `A` et `B`) et d'un conséquent ou tête (`C` pour la première règle). Une indication de l'intérêt de ces règles est donnée par des *fonctions d'évaluation*, par exemple la fréquence et la confiance. La fréquence d'une règle indique le nombre de lignes dans lesquelles tous les attributs de l'antécédent et du conséquent de la règle prennent la valeur 1. La confiance indique la probabilité conditionnelle (dans les données) que tous les attributs du conséquent d'une règle prennent la valeur 1 dans une ligne lorsque tous les attributs de l'antécédent prennent la valeur 1.

	A	B	C	D	E
1	1	1	1	1	1
2	1	1	1	1	0
3	1	0	1	1	0
4	1	1	0	0	1
5	0	0	1	1	0
6	0	0	1	0	1

1. Lorsque `A` et `B` sont vrais simultanément, alors `C` est vrai 2 fois sur 3 (fréquence 2, confiance 66%).
2. Lorsque `B` est vrai alors `A` est toujours vrai (fréquence 3, confiance 100%).

FIG. 1 – Une base de données binaire et deux règles d'association

Les règles d'association peuvent être utilisées dans de nombreux contextes. La matrice peut, par exemple, représenter des données de vente. Dans ce cas, chaque colonne correspond à un produit et chaque ligne à une transaction. Les cellules contenant la valeur "1" indiquent les produits présents dans chaque transaction. La fréquence et la confiance des règles fournissent alors des indications sur les produits qui tendent, ou non, à être achetés simultanément. Ces règles peuvent ensuite être utilisées dans un but commercial. Par exemple, si une règle indique que lorsqu'un produit `A` à forte valeur ajoutée est fréquemment acheté simultanément à un produit `B` bon marché, on peut imaginer de faire une offre promotionnelle sur le produit `B` pour doper les ventes du produit `A`. Les lignes peuvent aussi désigner des cellules dans différentes conditions biologiques et les colonnes des gènes. Dans la matrice, la valeur "1" indique alors que le gène est surexprimé dans la cellule concernée. Dans ce cas, les règles (par exemple, "Si les gènes A et B sont simultanément surexprimés dans une cellule, alors, dans 70% des cas, le gène D est également surexprimé") permettent de découvrir des relations entre l'expression des gènes et ainsi de mieux comprendre le fonctionnement cellulaire et la fonction des différents gènes.

Dans le cas d'une base de données de transactions, une date peut être associée à chaque transaction. Si on se restreint aux règles d'association, l'information relative aux dates n'est pas exploitée. Plus généralement, lorsque la base de données contient des indications temporelles ou spatiales, on peut utiliser les motifs séquentiels [MTV95] qui sont d'autres types de motifs locaux. Dans ce cas, on recherche des motifs du type : "si les produits `A` et `C` sont achetés le même jour par un même client alors il y a 60% de chance pour qu'il achète aussi le produit `B` dans la semaine qui suit"

Une base de données décrivant des molécules peut comporter, pour chaque molécule, le graphe représentant les liaisons entre les différents atomes qui la composent. On peut alors extraire les sous graphes fréquents dans une base de données de molécules actives sur le virus du SIDA et infréquents dans une base de données de molécules inactives afin de tenter de caractériser des structures chimiques communes aux molécules actives [KdRH01].

La génération de modèles globaux des données donne également lieu à de nombreux travaux. On peut citer par exemple le partitionnement (clustering) ou la construction d'arbres de décision. Dans cas du partitionnement, l'objectif est de regrouper les différents objets contenus dans la base de données en groupes (ou clusters) de sorte que les objets d'un même groupe soient aussi homogènes que possible et que deux groupes différents contiennent des objets suffisamment différents. Les arbres de décision sont des classifieurs, c'est-à-dire qu'ils permettent de prédire une des caractéristiques des objets en fonction des autres caractéristiques. Cela peut être utilisé, par exemple, pour déterminer quel est le traitement le plus adapté à un patient atteint d'une certaine pathologie en fonction de son âge, de son poids, etc. Pour ces deux types de modèles, on utilise des fonctions objectives pour mesurer la qualité de celui-ci, l'objectif étant de trouver le modèle qui maximise cette fonction. Cependant, il n'est généralement pas possible de trouver l'optimum de la fonction objective. On utilise donc des techniques heuristiques pour s'en approcher. Le livre de Han et Kamber [HK00] donne un bon état de l'art sur les différents algorithmes d'extraction de motifs ou de construction de modèles.

Formalisation : bases de données inductives. Les bases de données inductives ont été introduites en ECD en 1996 par Imielinski et Mannila [IM96] et une formalisation simple en a été donnée dans [BKM99]. De façon informelle, une base de donnée inductive est une extension des bases de données classiques qui contient, en plus des données, des collections de motifs définies en intention sur ces données. Un utilisateur peut donc interroger les données comme dans une base de données classique, mais il peut aussi utiliser des requêtes de sélection sur les motifs (par exemple, "quelles sont les règles à forte confiance concernant le gène A ?"). Une telle requête est appelée une requête inductive. Pour l'utilisateur tout ce passe comme si tous les motifs étaient disponibles, c'est au solveur de décider comment les motifs demandés dans la requête inductive doivent être extraits ou, par exemple, s'il est possible de les trouver dans une collection de motifs déjà matérialisée.

Plus formellement, étant donné une base de données bd, un langage de motifs $\mathcal{L}$ et un prédicat de sélection p, le résultat d'une requête inductive est l'ensemble de tous les motifs $\varphi \in \mathcal{L}$ qui satisfont le prédicat p dans la base de données bd. Dans le cas des règles d'association, $\mathcal{L}$ est l'ensemble de toutes les règles et le prédicat p peut, par exemple, imposer des seuils de fréquence et de confiance minimaux dans la base de donnée binaire bd. Ce cadre convient bien à l'extraction de motifs mais il peut aussi être utilisé pour la construction de modèles : dans le cas des arbres de décision, $\mathcal{L}$ est l'ensemble des arbres possibles et p est le prédicat qui est vrai sur l'arbre qui maximise la fonction objective. On constate d'ailleurs que des auteurs commencent à s'intéresser à l'extraction de modèles sous contraintes de manière

générale [GR00] (c'est-à-dire en utilisant des contraintes autres que juste la maximisation de la fonction objective).

Les requêtes inductives ainsi modélisées ne permettent cependant pas de prendre en compte un point important : l'utilisateur ne veut pas seulement obtenir la liste des motifs φ satisfaisant une contrainte mais également la valeur d'une fonction d'évaluation pour ces motifs dans la base de donnée (par exemple la fréquence et la confiance dans la base de donnée pour les règles d'association). Nous avons donc proposé des requête inductives étendues. Une telle requête est composée de la base de donnée bd, du langage de motifs $\mathcal{L}$, d'une fonction d'évaluation f sur les motifs et d'un prédicat de sélection q. Le résultat de cette requête étendue est l'ensemble des paires $(\varphi, f(\varphi, bd))$ telles que φ satisfait le prédicat p dans la base de donnée bd. Si nous prenons par exemple la requête "quelles sont les règles de confiance supérieure à 90% concernant le gène A ?", alors le prédicat p sera vrai sur les règles dont la confiance est supérieure à 90% et qui contiennent l'attribut A. Le résultat de cette requête sera la liste de règles qui satisfont ce prédicat p ainsi que les confiances et les fréquences des règles (si on suppose que la fonction d'évaluation f a deux composantes $f = (\mathsf{freq}, \mathsf{conf})$, la fréquence freq et la confiance conf).

Quelques problèmes identifiés. Nos travaux s'inscrivent dans le cadre du projet européen cInQ —*consortium on discovering knowledge with Inductive Queries*— financé par la branche *Future and Emerging Technologies* (FET) du programme *Information Society Technologies* de l'union européenne (IST-2000-26469). Ce projet, qui a débuté en mai 2001 pour une durée de 3 ans, s'est donné comme objectif de définir et de formaliser de la manière la plus précise possible les bases de données inductives.

Un certain nombre de problèmes ouverts ont déjà été identifiés dans le cadre de ce projet. Nous allons ici évoquer ceux sur lesquels portent nos travaux.

Définition des primitives. Lorsque l'utilisateur veut formuler une requête inductive, il doit pouvoir simplement définir les différentes composantes de celle-ci, c'est-à-dire la base de données bd sur laquelle se fait l'extraction, le langage des motifs $\mathcal{L}$, la fonction d'évaluation f et enfin le prédicat de sélection p sur les motifs. Enfin, l'utilisateur doit pouvoir faire des liens entre les motifs et les données : par exemple, il doit pouvoir simplement définir les données qui violent un ensemble de motifs (pour éventuellement faire une nouvelle extraction sur ces données particulières). Par exemple, après avoir découvert la règle "lorsque le gène A est surexprimé dans une cellule, alors dans 80% des cas le gène C est également sur-exprimé", l'utilisateur peut vouloir demander la liste des cellules dans lesquelles le gène A est surexprimé mais pas le gène C, autrement dit, les cellules qui violent la règle. Il faut donc proposer à l'utilisateur un certain nombre de primitives et un moyen de les combiner, i.e. un langage, pour qu'il puisse définir chacune de ces composantes.

Cela implique qu'il faut choisir quel va être le degré de liberté qui va être laissé à l'utilisateur. On peut par exemple se limiter à certains motifs particuliers, comme les règles d'association ou les itemsets, avec des fonctions d'évaluation figées. Évidement, ce problème de la définition des primitives est très lié à la conception d'un solveur pour les requêtes inductives : plus il y a de primitives et plus le solveur doit être générique. Généralement, les solveurs actuels sont très spécialisés pour un certain type de motifs, de fonctions d'évaluation et de prédicats de sélection ce qui permet de développer des optimisations ad-hoc.

Optimisation de l'évaluation des requetes inductives. L'optimisation des requêtes inductive est évidement un problème central pour les bases de données inductives. Il faut tout d'abord souligner la complexité de cette évaluation.

D'une manière générale, l'extraction des motifs de $\mathcal{L}$ qui satisfont le prédicat p est très coûteuse voire même impossible. Cela est dû à plusieurs facteurs. D'une part, le langage $\mathcal{L}$ est souvent très grand. Par exemple, dans le cas des règles d'association, s'il y a n attributs, le nombre de règles possibles est de l'ordre de 3^n. Dans une base de données réelle, le nombre d'attributs peut facilement atteindre plusieurs centaines ou plusieurs milliers. Le nombre de règles différentes est alors tellement élevé que le calcul de la fréquence et de la confiance de toutes les règles est impossible quelle que soit la puissance de calcul disponible (pour 50 attributs, il y a déjà plus de $7 \cdot 10^{23}$ règles). D'autre part, l'évaluation du prédicat de sélection peut être coûteuse. Cela est particulièrement vrai dans le cas (très fréquent) ou l'évaluation du prédicat implique le calcul d'une fonction d'évaluation. Il faut alors souvent faire un parcours de la base de donnée pour calculer la valeur de cette fonction sur un motif. C'est le cas, par exemple, lorsqu'il faut calculer la fréquence d'une règle d'association. En pratique, cela signifie qu'il n'est généralement pas possible de répondre à une requête du type : "lister toutes les règles d'association et leur confiance"

Pour permettre ce calcul, on se restreint donc le plus souvent aux règles d'association a priori les plus "intéressantes", c'est-à-dire celles dont la fréquence et la confiance sont plus élevées que des seuils définis par l'utilisateur. Ces règles sont appelées les règles *valides*. Évidemment, plus ces seuils sont élevés et moins il y a de règles valides. On peut trouver une justification statistique à ces seuils : d'une part, une règle ayant une faible fréquence risque de ne pas décrire une "régularité" des données mais plutôt être issue du bruit statistique, d'autre part, une règle ayant une faible confiance ne donne pas une bonne "description" des données. Cependant, il y a une autre raison plus pragmatique : sans ces seuils, l'extraction est impossible car le nombre de règles est trop important. D'ailleurs, ces seuils sont souvent fixés par des considérations de temps de calcul et/ou de ressources mémoire (on prend les seuils les plus bas qui permettent une extraction en un temps raisonnable) et non pas par des considération statistiques.

D'une manière plus générale, il faut étudier comment utiliser le prédicat de sélection p pendant l'extraction pour ne pas avoir à explorer tous les motifs du langage $\mathcal{L}$. Ce problème est crucial car il n'est généralement pas possible de tester le prédicat sur tous les motifs.

Voyons maintenant rapidement comment cela peut être fait dans le cas des règles d'association. Pour générer les règles d'association valides, on utilise traditionnellement une étape intermédiaire, l'extraction des itemsets fréquents. Un itemset est un ensemble d'attributs (par exemple, $\{\mathtt{A}, \mathtt{C}, \mathtt{D}\}$) et la fréquence d'un itemset est le nombre de lignes dans la base de données où tous les attributs de l'itemset prennent la valeur 1 (par exemple, dans la figure 1, la fréquence de $\{\mathtt{A}, \mathtt{C}, \mathtt{D}\}$ est 3). Une fois que l'on dispose des itemsets fréquents et de leur fréquence, il est relativement aisé de générer les règles d'association. Nous n'entrerons pas dans les détails ici, mais cette génération peut se faire très efficacement. L'étape la plus coûteuse est l'extraction des itemsets fréquents et c'est cette dernière qui doit être optimisée. Pour cela, on utilise une propriété importante de la fréquence : elle est décroissante par rapport à l'inclusion. En effet, si les attributs d'un itemset S sont vrais dans une ligne de la base de données, alors, a fortiori, les attributs d'un sous ensemble T de S sont aussi vrais dans cette même ligne. Par conséquent, les attributs de T sont vrais dans toutes les lignes où ceux de S le sont. Cela signifie exactement que la fréquence de S est plus faible que celle de T. Cette propriété est très utile. En effet, si la fréquence d'un itemset T est plus faible que le seuil fixé par l'utilisateur, alors la fréquence de tous les sur ensembles de T est aussi inférieure

à ce seuil. Cela signifie qu'il n'est pas nécessaire de considérer les itemsets qui sont des sur ensembles d'itemsets non fréquents. Cette technique est utilisée dans tous les algorithmes et permet d'extraire des itemsets et des règles d'association même lorsque le nombre d'attributs est grand, pour peut que le nombre d'itemsets fréquents ne soit pas trop élevé.

Pour pouvoir utiliser le prédicat efficacement, il est donc nécessaire d'étudier la relation entre le prédicat et l'ordre définit sur le langage $\mathcal{L}$. De nombreux travaux sont consacrés à ce problème aussi bien pour des motifs et des prédicats spécifique que dans le cas général (voir par exemple [TR97]).

Enfin, comme nous l'avons signalé, le processus d'ECD nécessite souvent plusieurs étapes. Cela signifie qu'il ne faut pas se limiter à l'étude de l'optimisation d'une seule requête inductive. Le problème de l'optimisation d'une séquence de requêtes se pose également.

Utilisation de représentations condensées. Les représentations condensées ont été introduites en 1996 par Mannila et Toivonen [MT96b]. De façon informelle, une représentation condensée est une représentation alternative des motifs qui permet de calculer la fonction d'évaluation de manière plus efficace qu'avec un accès aux données. Depuis cet article, plusieurs types de représentations condensées ont été proposées. L'utilisation des représentations condensées présente deux intérêt principaux. D'une part, elles peuvent être extraites plus efficacement que les motifs tout en permettant ensuite de régénérer les motifs et la valeur des fonctions d'évaluation sur ceux-ci. D'autre part, elles contiennent la même information que les collections de motifs tout en étant plus succinctes.

Dans le cas des itemsets et des règles d'association, une des étapes les plus coûteuse des algorithmes d'extraction concerne le calcul de la fréquence des itemsets. Pour calculer la fréquence d'un itemset S, il faut en effet parcourir toute la base de données *bd* et incrémenter la fréquence à chaque fois que l'on trouve un tuple dans lequel tous les attributs de S sont présents. Lorsque le nombre de tuples dans la base de données est très grand, ce calcul devient très coûteux. L'idée des représentations condensées est alors de ne faire ce calcul de fréquence que pour certains itemsets bien choisis. C'est cette collection d'itemsets particuliers qui est appelée une représentation condensée. Ensuite, la fréquence des autres itemsets est inférée à partir de cette représentation condensée sans accéder aux données. Nous donnons maintenant un exemple de la manière dont la fréquence de certains itemsets peut être inférée. Dans notre base de données de la figure 1, on constate que les fréquences de $\{\mathtt{B}\}$ et de $\{\mathtt{A},\mathtt{B}\}$ sont égales. Cela signifie qu'à chaque fois qu'une ligne contient l'attribut `B`, alors elle contient aussi l'attribut `A`. Ainsi, la fréquence de l'itemset $X \cup \{\mathtt{A},\mathtt{B}\}$ est égale à la fréquence de $X \cup \{\mathtt{B}\}$ quel que soit l'itemset X. Par exemple, les fréquences de $\{\mathtt{A},\mathtt{B},\mathtt{C}\}$ et de $\{\mathtt{B},\mathtt{C}\}$ sont égales. Il n'est donc pas nécessaire de calculer la fréquence des itemsets de la forme $X \cup \{\mathtt{A},\mathtt{B}\}$. Cette méthode d'inférence de la fréquence est très simple mais elle permet, dans les jeux de données corrélés, d'améliorer les performances de l'extraction de manière très significative.

Contributions. Dans le cadre du projet européen cInQ, nous nous sommes particulièrement intéressés à l'optimisation de l'évaluation de requêtes inductives. Nous avons tenté de présenter des algorithmes les plus généraux possible afin de pouvoir traiter de nombreux types de contraintes. Nous avons ensuite mis l'accent sur l'utilisation conjointe des contraintes et des représentations condensées tant pour l'optimisation d'une requête unique que pour une séquence de requêtes. Plus précisément, ce mémoire se décompose de la manière suivante :

Dans le premier chapitre, nous présentons le cadre des bases de données inductives. C'est ce cadre qui permettra ensuite de positionner nos travaux par rapport aux objectifs du projet

européen cInQ. Nous commençons par étudier le processus d'extraction de connaissances et nous donnons quelques indications sur la manière dont ce processus pourrait être intégré dans le cadre des bases de données inductives. Ensuite, nous présentons une formalisation des requêtes inductives et l'algorithme générique Guess and correct pour l'évaluation des requêtes utilisant un type de contraintes bien particulier, les contraintes anti-monotones. Nous finissons par évoquer les problèmes ouverts liés aux bases de données inductives et la manière dont nos travaux se positionnent par rapport à ces problèmes.

Le chapitre 2 concerne l'extraction d'itemsets sous contraintes. Après un état de l'art sur les diverses techniques utilisées pour tirer parti des contraintes lors de l'extraction de motifs, nous présentons un algorithme générique d'extraction d'itemsets satisfaisant une contrainte qui est la conjonction d'une contrainte anti-monotone et d'une contrainte monotone. Notre principale motivation et d'étendre le cadre générique de Mannila et Toivonen d'extraction en présence de contraintes anti-monotones pour y inclure les contraintes monotones. Cet algorithme a été publié dans [BJ00].

Comme nous l'avons vu plus haut, l'utilisation des représentations condensée semble très prometteuse. Dans le chapitre 3, nous présentons une technique nouvelle d'extraction de représentations condensées sous contraintes. Pour cela, nous utilisons l'algorithme générique du chapitre 2 avec de nouvelles contraintes liées aux représentations condensées. L'intégration de l'extraction sous contraintes et des représentations condensées pose des problèmes nouveaux que nous résolvons en introduisant les itemsets δ-libres contextuels [BJ01b]. Nous présentons ensuite les nouvelles stratégies d'évaluation des requêtes inductives qui peuvent être conçues en utilisant l'extraction de représentations condensées sous contraintes [JB02a, BJ01a]. Nous finissons par une évaluation expérimentale de notre approche qui montre que cette technique est supérieure à des stratégies qui n'utilisent que les représentations condensées ou que les contraintes. Nous nous attaquons dans cette expérience au problème difficile de l'extraction des itemsets généralisés qui sont utilisées pour la génération des règles avec négations [BBJ00]. Enfin, Nous y présentons aussi une utilisation fructueuse de nos techniques sur des données d'expression de gènes faite en collaboration avec des biologistes [BBJ$^+$02].

Le chapitre 4 présente l'utilisation originale des représentations condensées pour l'optimisation d'une séquence de requêtes [JB02c]. Pour cela, nous utilisons les représentations condensées comme cache pour stocker les résultats des requêtes précédentes. Nos expériences montrent que cette technique est efficace et que la taille de notre cache est considérablement plus faible que les caches qui étaient utilisés auparavant.

Nous finissons par une conclusion dans laquelle nous replaçons nos contribution dans le cadre des bases de données inductives et dégageons quelques perspectives.

Chapitre 1

Bases de données inductives

La théorie des bases de données inductives a été introduite en 1993 par Imielinski et Mannila [IM96] et a ensuite été développée, par exemple, dans [BKM98, BKM99]. Elles sont au cœur du projet européen cInQ (IST-2000-26469) [ec] dans lequel s'inscrivent nos recherches. Les bases de données inductives sont présentées et étudiées en particulier dans le rapport D1 [cc02b] de ce projet. Malgré ces recherches, il reste encore de nombreux problèmes ouverts et il n'y a pas encore de réel consensus sur ce qu'est une base de données inductive. L'objectif de ce chapitre et donc de présenter un point de vue sur les bases de données inductives pour servir de cadre à nos recherches.

1.1 Motivations

Un processus d'extraction de connaissances comporte plusieurs grandes étapes : pré-traitement, extraction, et post-traitement. Ces différentes étapes vont être détaillées dans cette section et illustrées sur les processus d'extraction de règles d'association et d'itemsets que nous introduisons maintenant.

1.1.1 Itemsets et règles d'association

Nous utiliserons la notation $T \subseteq S$ pour indiquer que T est un sous ensemble de S et $T \subset S$ pour indiquer que T est un sous ensemble strict ($T \neq S$) de S. Le cardinal d'un ensemble S est noté $|S|$ et $T \setminus S$ est l'ensemble des éléments de T qui ne sont pas dans S.

Étant donné un ensemble S muni d'un ordre $\leq$ (pas nécessairement total), on définit l'ensemble des éléments maximaux et l'ensemble des éléments minimaux de S de la manière suivante :

$$\max_{\leq}(S) = \{a \in S \mid \nexists b(b \in S \wedge a < b)\},$$
$$\min_{\leq}(S) = \{a \in S \mid \nexists b(b \in S \wedge b < a)\}.$$

Donnons ici la définition formelle des itemsets et des règles d'association. L'extraction des itemsets et des règles d'association se fait dans des bases de données binaires. La figure 1.1 donne un exemple d'une base de données binaire.

Définition 1 (base de données binaire) *Le schéma d'une base de données binaire est* $\langle Tid, \mathsf{Items}\rangle$. *L'ensemble* Items *est un ensemble de n attributs* $\mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots$. *L'attribut* Tid *est de*

type entier et est la clef de la relation. Une base de données binaire instance de ce schéma est donc un ensemble de lignes ou de transactions où chaque ligne est composée d'un identificateur unique et d'un sous ensemble de Items*. Une transaction d'identificateur i est notée t_i, le sous ensemble de* Items *correspondant à cette transaction est noté t_i.*item*.*

$bd_1 =$

Tid	Items
1	ABCDE
2	ABCD
3	ACD
4	ABE
5	CD
6	CE

Itemset	Support	Fréquence absolue
A	$\{t_1, t_2, t_3, t_4\}$	4
B	$\{t_1, t_2, t_4\}$	3
AB	$\{t_1, t_2, t_4\}$	3
ABC	$\{t_1, t_2\}$	2
BD	$\{t_1, t_2\}$	2
CDE	$\{t_1\}$	1

FIG. 1.1 – Base de données binaire bd_1 et supports et fréquences de quelques itemsets.

Définition 2 (Itemset, règle d'association) *Un* itemset *S est un sous ensemble de* Items $= \{\mathtt{A}, \mathtt{B}, \mathtt{C}, ...\}$*. L'ensemble des itemsets est noté 2^{Items}. Une* règle d'association *R est un motif $X \Rightarrow Y$ où X et Y sont des sous ensembles de* Items *tels que $Y \neq \emptyset$ et $X \cap Y = \emptyset$. X est appelé le corps, l'antécédent ou la partie gauche de la règle et Y la tête, le conséquent ou la partie droite.*

L'ensemble des itemsets 2^{Items} ordonné par l'inclusion ensembliste est naturellement muni d'une structure de treillis. Nous l'appellerons donc parfois le treillis des itemsets. Le plus petit élément de ce treillis est l'ensemble vide et son plus grand élément est Items (voir la figure 1.2).

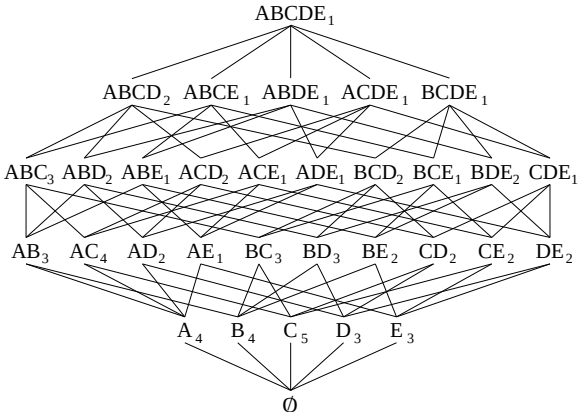


FIG. 1.2 – Le treillis des itemsets, la fréquence de chaque itemset dans bd_1 est indiquée en indice.

Définition 3 (support, fréquence, confiance) *Soit bd une base de données binaire de schéma $\langle Tid, Items\rangle$. Soit S un itemset ($S \subseteq$* Items*), une transaction t* supporte *S si $S \subseteq t$.*items*. Le* support *de S dans bd, noté $\mathsf{sup}(S, bd)$, est l'ensemble des transactions de bd qui supportent S.*

$$\mathsf{sup}(S, bd) = \{t \in bd \mid S \subseteq t.\mathsf{items}\}\,.$$

La fréquence absolue *de S dans bd est définie comme le cardinal de $\mathsf{sup}(S, bd)$,*

$$\mathsf{freq}_a(S, bd) = |\mathsf{sup}(S, bd)|\,.$$

La fréquence relative de S est la proportion des lignes de bd qui supportent S,

$$\mathsf{freq}_r(S, bd) = \frac{|\mathsf{sup}(S, bd)|}{|bd|}.$$

Soit $R = X \Rightarrow Y$ une règle d'association, le support et la fréquence de R sont définis comme le support et la fréquence de $X \cup Y$. La confiance de R dans bd est

$$\mathsf{conf}(X \Rightarrow Y, bd) = \frac{\mathsf{freq}(X \Rightarrow Y, bd)}{\mathsf{freq}(X, bd)}.$$

Une règle d'association est dite exacte *si elle a une confiance de 1. Un itemset ou une règle d'association sont dits γ-fréquents si leur fréquence est supérieure à γ.*

Quelques itemsets, leurs supports et leurs fréquences sont donnés dans la figure 1.1 et la figure 1.2 donne les fréquences de tous les itemsets dans la base de données binaire bd_1. Dans la définition de la confiance d'une règle, on peut utiliser indifféremment la fréquence absolue ou relative (il faut évidemment utiliser les mêmes fréquences au numérateur et au dénominateur). Lorsqu'il n'y aura pas d'ambiguïté possible, les indices a et r seront omis. De même, on notera $\mathsf{sup}(S)$ et $\mathsf{freq}(S)$ à la place de $\mathsf{sup}(S, bd)$ et $\mathsf{freq}(S, bd)$.

Étant donnés deux seuils γ et θ, les deux tâches d'extraction les plus classiques sont d'une part l'extraction de tous les itemsets γ-fréquents et d'autre part l'extraction de toutes les règles γ-fréquentes dont la confiance dépasse θ. Nous les définissons maintenant en terme d'extraction sous contraintes.

Définition 4 (Contraintes de fréquence et confiance minimales) *La contrainte de fréquence minimale $\mathcal{C}_{\gamma\text{-}\mathsf{freq}}$ est une fonction de $\mathcal{L} \times \mathcal{B}$ dans $\{vrai, faux\}$. $\mathcal{C}_{\gamma\text{-}\mathsf{freq}}(S, bd)$ est vraie si l'itemset S est γ-fréquent dans bd, autrement dit :*

$$\mathcal{C}_{\gamma\text{-}\mathsf{freq}}(S, bd) = (\mathsf{freq}(S, bd) \geq \gamma).$$

Si R est une règle d'association, nous définissons de même une contrainte de fréquence minimale et une contrainte de confiance minimale par :

$$\mathcal{C}_{\gamma\text{-}\mathsf{freq}}(R, bd) = (\mathsf{freq}(R, bd) \geq \gamma),$$
$$\mathcal{C}_{\theta\text{-}\mathsf{conf}}(R, bd) = (\mathsf{conf}(R, bd) \geq \theta).$$

Lorsque bd est clairement défini par le contexte, nous noterons $\mathcal{C}_{\gamma\text{-}\mathsf{freq}}(R)$ et $\mathcal{C}_{\theta\text{-}\mathsf{conf}}(R)$.

L'ensemble des itemsets γ-fréquents est donc exactement l'ensemble des itemsets qui satisfont la contrainte $\mathcal{C}_{\gamma\text{-}\mathsf{freq}}$. De même, l'ensemble des règles γ-fréquentes de confiance supérieure à θ est exactement l'ensemble des règles qui satisfont la conjonction $\mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\theta\text{-}\mathsf{conf}}$. Nous dirons qu'une règle est *valide* par rapport à des seuils γ et θ de fréquence et de confiance si elle vérifie la contrainte $\mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\theta\text{-}\mathsf{conf}}$. En fait, nous utiliserons le terme *valide* à chaque fois que nous voudrons désigner des motifs qui satisfont une contrainte donnée.

Introduisons la propriété fondamentale de la fréquence qui est à la base de l'efficacité des algorithmes d'extraction des motifs fréquents.

Proposition 1 *La fréquence est une fonction décroissante par rapport à l'inclusion ensembliste. i.e., soit bd une base de données binaire, S et T deux itemsets, alors*

$$S \subseteq T \Rightarrow \mathsf{freq}(S, bd) \geq \mathsf{freq}(T, bd).$$

1.1.2 Processus d'extraction de connaissances

Durant ces dernières années, de nombreux progrès ont été faits dans la conception d'algorithmes d'extraction de motifs ou de construction de modèles des données. Les motifs et les modèles sont de très nombreux types : règles d'association, dépendances fonctionnelles ou d'inclusion, concepts, clusters, épisodes, requêtes datalog, arbres de décision etc. La variété des domaines d'application est elle aussi très grande : analyse de données médicales ou biologiques, rétro-conception de bases de données, classification de documents, analyse de logs de connexions sur un serveur, analyse d'erreurs dans des réseaux de télécommunication, détection de fraudes ou d'événements exceptionnels, etc.

Cependant, l'ECD ne se limite pas à l'extraction de motifs ou à la construction de modèles. Pour espérer trouver quelque chose dans les données, il faut en effet se lancer dans un processus qui est plutôt lent et qui demande beaucoup de compétences. Ce processus d'extraction de connaissances comporte plusieurs étapes que nous allons illustrer sur un exemple tiré d'une collaboration avec des biologistes [BBJ+02]. Nous commençons par quelques rappels de biologie.

Une cellule, à tout moment de sa vie, produit un grand nombre de protéines. La "formule" de chaque protéine est stockée dans l'ADN du noyau sous forme d'un gène[1]. Lorsque la cellule doit fabriquer un certain type de protéines, une copie de la formule portée par l'ADN est faite sur un ARN messager. Celui-ci sort du noyau et est ensuite utilisé pour produire des protéines. Le nombre d'ARN messagers correspondant à une certaine protéine est donc une mesure de la quantité de ce type de protéine que la cellule est en train de synthétiser. Ce nombre est appelé le *niveau d'expression* du gène codant pour cette protéine. Maintenant que le décodage de l'ADN est fini pour plusieurs espèces animales et végétales, il reste encore un formidable défi pour la biologie moderne : comprendre à quoi sert chaque gène et chaque protéine et comprendre comment la production de chaque protéine (i.e., le niveau d'expression des gènes) est réglé en fonction du type de la cellule, de son environnement et de son état. La mesure et l'étude du niveau d'expression des gènes de différentes cellules dans différentes conditions est un des moyens à la disposition des biologistes pour tenter de répondre à ces questions. Cependant, le grand nombre de gènes (estimé à 30000 pour l'espèce humaine) ne permet pas de faire ces analyses "à la main". Les biologistes ont besoin d'outils informatiques pour les assister dans cette tâche.

Les données brutes que nous avons utilisées sont constituées de plusieurs fichiers. Chacun d'eux contient les résultats d'une expérience de mesure de l'expression des gènes faite sur une culture de cellules. Ces expériences consistent à prélever un grand nombre d'ARN messagers dans les cellules et de leur assigner une "signature" composée d'un mot d'une dizaine de lettres sur l'alphabet A,C,T,U. Cette signature doit en théorie permettre d'identifier le gène exprimé par chaque ARN via des bases de données de gènes. Un fichier contient donc une liste de signatures et, pour chaque signature, le nombre de copies de l'ARN correspondant compté dans les ARN prélevés. Ainsi, il est possible de connaître, à l'instant où les prélèvements sont faits, quel est le niveau d'expression de chaque gène sous la forme d'un entier.

La préparation des données ou le pré-traitement Tout d'abord, une phase de pré-traitement doit être faite sur ces données avant qu'elles puissent être réellement exploitées. Ceci inclus la récupération des données (éventuellement à partir de sources différentes), leur fusion, le traitement éventuel des valeurs manquantes ou erronées, et la mise en forme dans un format exploitable par les algorithmes.

[1] La réalité est bien sûr plus complexe ...

Dans le cas de notre expérience, il a d'abord fallut récupérer les données sur internet. Ces données se trouvaient sous forme de fichiers textes qui ont dû être transformés pour être stockés dans une base de données. Ensuite, les signatures ont été triées. En effet, certaines signatures étaient ambiguës (plusieurs gènes correspondaient à une même signature) ou inconnues (aucun gène n'avait de signature correspondante). Les données ont également été normalisées car le nombre d'ARN prélevés dans chaque expérience était extrêmement variable (de quelques milliers à environ cent mille) et certaines expériences où le nombre de prélèvements était insuffisant n'ont pas été prises en compte. Nous disposions donc à ce moment, pour chaque paire (expérience, gène) d'une valeur normalisée représentant le niveau d'expression de ce gène dans l'expérience correspondante.

Cependant, cette valeur n'était pas binaire, et nos algorithmes d'extraction de règles et d'itemsets ne fonctionnent qu'avec des données binaires. Il a donc fallut binariser ces valeurs. Il y a évidemment de très nombreuses manières d'effectuer cette binarisation. On peut par exemple prendre l'ensemble des valeurs correspondant à un gène, calculer leur moyenne et transformer la valeur en "1" dans le cas où cette valeur est supérieure à la moyenne et "0" sinon. Dans ce cas, une valeur de "1" signifie "le gène est sur-exprimé" et une valeur de "0", "le gène n'est pas sur-exprimé". On peut aussi se demander s'il ne vaut pas mieux prendre la médiane au lieu de la moyenne, ou choisir de faire trois classes pour chaque gène : une classe "sous exprimé", une classe "moyennement exprimé" et une classe "sur-exprimé". Finalement, cette étape de pré-traitement est longue, complexe, et nécessite de faire de nombreux choix. De plus, Il est difficile de déterminer dans quelle mesure ces choix ont une influence sur le résultat des extractions.

Phase d'extraction des motifs Cette phase concerne l'utilisation d'un algorithme d'extraction de motifs sur les données préparées pendant l'étape précédente. Dans cette phase, l'utilisateur doit choisir le type de motifs qu'il souhaite extraire (itemsets, règles d'association, clusters, etc.), définir les contraintes sur ces motifs et fixer les paramètres des algorithmes. Cette définition des contraintes est importante car souvent l'ensemble des motifs possibles est tellement grand qu'il n'est pas calculable, il faut donc le restreindre. Dans le cas des règles ou des itemsets, il s'agit souvent de fixer des seuils minimums de fréquence et de confiance. Seuls les itemsets ou les règles dont la fréquence et la confiance excèdent ces seuils sont extraits.

Dans nos expériences d'extraction sur des données d'expression de gènes, l'extraction des itemsets fréquents n'était pas faisable avec des seuils de fréquences intéressants pour les biologistes. Nous avons donc utilisé un algorithme qui extrait un sous ensemble des itemsets fréquents, les itemsets libres fréquents (que nous étudierons en détail dans le chapitre 3). L'algorithme calcule aussi la fréquence et la fermeture de chacun de ces itemsets. La fermeture d'un itemset peut être définie comme l'intersection de toutes les lignes de la base de données qui contiennent l'itemset. Si, par exemple, un itemset libre contient les gènes A et B et sa fermeture les gènes A, B et C, cela peut être interprété par : "lorsque les gènes A et B sont surexprimés simultanément dans une cellule alors le gène C est également surexprimé dans cette cellule". Un tel motif permet aux biologistes de découvrir une relation forte entre les gènes A et B d'une part et le gène C d'autre part.

Phase de post-traitement des résultats On distingue habituellement deux grand types de techniques pour faire le post-traitement : les techniques indépendantes de l'application et celles qui sont spécifiques à une application donnée. On peut placer dans la première catégorie toutes les techniques qui visent à classer les motifs en fonction de la valeur d'une fonction d'évaluation. Les fonctions d'évaluation utilisables pour les itemsets ou les règles

d'association sont très nombreuses. Il y a bien sûr la fréquence et la confiance, mais nous en verrons d'autres. Nous verrons aussi des techniques qui permettent de limiter la redondance qui est souvent très importante dans les collections d'itemsets et de règles.

Dans nos expériences d'extraction sur des données d'expression de gènes, les biologistes ont utilisé une technique de couleur codage pour visualiser les itemsets. Chaque couleur correspond à une fonction biologique des gènes. Cette technique a permis de s'apercevoir que les gènes qui apparaissaient dans les itemsets et dans leurs fermetures avaient très souvent des fonctions biologiques proches. Cette homogénéité dans les fonctions des gènes était très intéressante d'un point de vue biologique.

Phase d'interprétation Les algorithmes d'extraction de motifs ou de construction de modèles permettent découvrir des propriétés des données. Néanmoins, ces propriétés ne peuvent pas être considérées comme de nouvelles connaissances tant qu'elles n'ont pas été interprétées et validées par un expert humain. En effet, les propriétés mises à jour par les algorithmes peuvent être inintéressantes, incomplètes voire même provenir d'erreurs dans les données. Par exemple, dans nos expériences d'extraction sur des données d'expression de gènes, les biologistes ont découvert, en visualisant les itemsets avec la technique de couleur codage, qu'un gène apparaissait souvent associé avec d'autres gènes qui partageaient entre-eux la même fonction biologique. Cependant ce gène n'était pas indiqué comme ayant la même fonction. En fait, il est apparu qu'une erreur d'attribution de fonction avait été faite dans la construction de la base de donnée.

1.1.3 Problèmes posés par la gestion du processus d'ECD

L'ECD est un processus exploratoire. Cela signifie qu'il est intrinsèquement iteratif. En effet, il est très peu probable de trouver des motifs intéressant dès la première extraction. Il faut souvent faire des essais avant de trouver les bonnes valeurs pour les paramètres ou les bon algorithmes d'extraction. De plus, lors de la phase d'interprétation, il n'est pas rare de devoir relancer de nouvelles extractions pour tenter de comprendre ou d'affiner les résultats des extractions précédentes.

Or, toutes les étapes du processus d'ECD sont actuellement souvent faites de manière adhoc. Pour le pré-traitement des données, il n'est pas rare d'utiliser des scripts même lorsque les données sont stockées dans une base de données. En effet, les langages de bases de données ou les langages utilisés dans les outils d'ECD ne permettent généralement pas d'effectuer les pré-traitements classiques comme la binarisation des valeurs continues ou l'échantillonnage [BBMM02].

La situation est identique dans le traitement des motifs extraits. Les langages spécifiques pour l'ECD sont encore très souvent limités à la phase d'extraction proprement dite. Il est parfois possible de concevoir un schéma de base de données relationnelle pour les stocker, mais même dans ce cas, les langages comme SQL ne sont pas non plus adaptés pour le post-traitement (voir de même [BBMM02]).

Dans notre cas, les traitements (binarisation, fusion des données, codage utilisant les couleurs, etc.) ont été faits en utilisant des outils différents : tableur, scripts, SGBD. Tous ces traitements ont occupé une stagiaire à temps plein pendant 3 mois. En comparaison, le temps des extractions proprement dites était de l'ordre de quelques dizaines de minutes...

1.1.4 L'apport des bases de données inductives

Chaque étape du processus d'extraction est actuellement gérée par des outils différents. Le pré-traitement et le post-traitement peuvent parfois être faits en utilisant des outils classiques sur les bases de données et les entrepôts de données (techniques OLAP, utilisation de langages de type SQL), des outils statistiques ou des scripts quand ces outils ne conviennent pas.

La motivation des bases de données inductives est de proposer un cadre "base de données" dans lequel pourrait se dérouler l'intégralité de ce processus d'extraction. Une tâche d'extraction, une étape de pré-traitement ou de post-traitement pourraient être décrites par des requêtes sur une base de données inductive. Celle-ci contiendrait non seulement les données mais aussi les différentes collections de motifs définies en intention. L'utilisateur pourrait donc faire des requêtes non seulement sur les données, comme dans une base de donnée classique, mais aussi sur les collections de motifs ou à la fois sur les données et sur les motifs. Évidemment, les collections de motifs ne peuvent pas être complètement matérialisées du fait du nombre des motifs. Cependant, du point de vue de l'utilisateur, tout se passe comme si tous les motifs étaient disponibles. La collection des motifs doit donc être gérée comme une vue. L'utilisateur peut créer de nouvelles vues sur ces collections en sélectionnant les motifs qui l'intéressent. Ces vues ne doivent être matérialisées que lorsque l'utilisateur le demande explicitement. Dans la suite, nous appellerons ces requêtes qui portent sur les collections de motifs *requêtes inductives*. Bien évidemment, toutes les vues définies par l'utilisateur ne pourront pas toujours être matérialisées. Il faut pour cela que ces vues soient suffisamment petites et qu'il soit possible de les calculer efficacement. Il y a donc un important problème d'optimisation.

Une caractéristique importante d'une base de données inductive est de pouvoir sélectionner des données relativement à certains motifs. On peut, par exemple, vouloir sélectionner les données vérifiant un ensemble de règles ou au contraire les données qui sont des exceptions pour un ensemble de règles. Si on a par exemple découvert une règle indiquant que le port de la ceinture de sécurité améliore les chances de survie lors d'un accident, on peut s'intéresser au sous ensemble des situations pour lequel le port de la ceinture aggrave les blessures. On pourra alors lancer une nouvelle extraction sur ces situations exceptionnelles afin de mieux les comprendre et les caractériser.

1.1.5 Exemples de langages existants

Il existe déjà plusieurs langages dédiés à la spécification de requêtes inductives. On peut citer MSQL [IV99], MINE RULE [MPC97] qui sont des extensions de SQL pour les règles d'association, DMQL [HK00] qui permet d'utiliser plusieurs types de motifs et de modèles ou RDM [dR00] pour les requêtes datalog (qui généralisent les règles d'association).

Nous donnons ici un exemple d'une requête MINE RULE pour extraire des règles d'association dans une base de données relationnelle. Le lecteur intéressé par la spécification complète du langage peut se référer à [MPC96, MPC97]

Exemple 1 *Considérons l'exemple de requête* `MINE RULE` *de la figure 1.4 sur la base de la table 1.3. Cette base concerne l'enregistrement de diverses transactions réalisées par trois clients à différentes dates. Le résultat de cette requête est la collection de toutes les règles valides dans la base spécifiée. La clause* `FROM` *définit les données qui vont constituer les transactions. Toutes les possibilités du langage SQL peuvent être utilisées. La clause* `GROUP BY` *dit que les enregistrements de la table* `purchase` *sont groupés par transactions (un groupement alternatif serait, e.g., de grouper par client). On ne sélectionne que les transactions*

réalisées après le 8 novembre. La clause `SELECT` *fixe le format des règles recherchées (par des contraintes syntaxiques). Ainsi, dans l'exemple, le corps et la tête sont des ensembles de produits (*`products`*) de taille au moins égale à 1 et sans borne supérieure. Cette requête définit également d'autres contraintes sur les règles : les règles doivent être fréquentes (avec un seuil de fréquence de 0.2), avoir une confiance supérieure à 0.7, et doivent satisfaire les autres contraintes exprimées dans la clause* `SELECT` *:* $\mathcal{C}_a(X \Rightarrow Y) \equiv \forall y \in Y, y.price > 100$ *et* $\mathcal{C}_b(X \Rightarrow Y) \equiv |(X \cup Y) \cap \{\mathtt{A}, \mathtt{B}\}| \leq 1$. $\mathcal{C}_a$ *impose que tous les produits de la tête de règle aient un prix supérieur à 100 et* $\mathcal{C}_b$ *impose que la règle contienne au plus un produit de l'ensemble* $\{\mathtt{A}, \mathtt{B}\}$.

Tid.	Cid.	prod.	date	prix
1	1	C	8 Nov.	5
1	1	D	8 Nov.	5
2	2	A	9 Nov.	5
2	2	B	9 Nov.	5
2	2	C	9 Nov.	5
2	2	D	9 Nov.	5
3	1	B	9 Nov.	5
3	1	C	9 Nov.	5
4	3	A	10 Nov.	5
4	3	C	10 Nov.	5
⋮	⋮	⋮	⋮	⋮

FIG. 1.3 – Extrait de la table `achat`.

```
MINE RULE résultat AS
SELECT DISTINCT 1..n produit AS BODY,
   1..n produit AS HEAD, SUPPORT, CONFIDENCE
   WHERE HEAD.prix> 100 AND
   |(HEAD ∪ BODY) ∩ {A,B}|≤ 1
FROM achat WHERE date > 8 Nov.
GROUP BY transaction
EXTRACTING RULES WITH SUPPORT : 0.2, CONFIDENCE : 0.7
```

FIG. 1.4 – Une requête `MINE RULE` sur la table `achat`.

Le langage MINE-RULE est, comme son nom le suggère, conçu pour spécifier des tâches d'extraction de règles d'association. Comme le montre l'exemple précédent, ce langage permet de définir les contraintes et les données sur lesquelles les règles sont extraites. Par contre, il n'est pas possible d'utiliser d'autres fonctions d'évaluation que la fréquence et la confiance. De plus, ce langage ne fourni pas de moyen simple de faire un croisement entre les données et les règles extraites. Il faut pour cela utiliser des requêtes SQL ce qui est malaisé [BBMM02]. Finalement, MINE RULE, tout comme MSQL, reste très lié à l'algorithme d'extraction sous-jacent. Le langage DMQL permet, quant à lui, l'extraction de différents types de motifs. Seulement, ici encore, il ne sert qu'à choisir l'algorithme à utiliser et à fixer ses paramètres. Le langage RDM est lui beaucoup plus riche et permet à l'utilisateur de préciser le langage de motif, les fonctions d'évaluation et les contraintes. Cependant, il n'existe pas d'implémentation efficace de ce langage. Finalement, les langages existants sont généralement soit trop

fortement liés à un algorithme d'extraction particulier, ce qui limite leur généralité, soit trop complexes pour permettre une évaluation efficace des requêtes avec les techniques actuelles. Ceci montre bien la nécessité de formaliser l'extraction de connaissances. C'est ce qui est tenté avec les bases de données inductives.

1.2 Formalisation des bases de données inductives

Une base de données inductive est constituée de deux parties : une partie contenant les données et une autre concernant les motifs.

Définition 5 *Le* schéma d'une base de données inductive *est un* k+1*-uplet* $\mathcal{I} = (\mathcal{B}, (\mathcal{L}_1, e_1, \mathcal{D}_1), ..., (\mathcal{L}_k, e_k, \mathcal{D}_k))$*, où* $\mathcal{B}$ *est un schéma de base de données, pour* $1 \leq i \leq k$*,* $\mathcal{L}_i$ *est une collection de motifs et* e_i *est une* fonction d'évaluation *qui définit la sémantique des motifs en leur associant des valeurs dans le domaine* $\mathcal{D}_i$*. Cette fonction associe à chaque paire* (θ, bd) *un élément de* $\mathcal{D}_i$*, où bd est une instance de* $\mathcal{B}$ *et* $\theta \in \mathcal{L}_i$ *est un motif. Une instance de ce schéma, i.e., une base de données inductive* $(bd, s_1, ..., s_k)$ *de schéma* $\mathcal{I}$ *est une instance bd de schéma* $\mathcal{B}$ *associée à* $k-1$ *collections de motifs* $s_i \subseteq \mathcal{L}_i$*.*

Exemple 2 *Le schéma d'une base de données binaire contenant des itemsets et des règles d'association sur des données binaires pourrait être* $(\mathsf{Items}, (\mathcal{L}_1, e_1, \mathcal{D}_1), (\mathcal{L}_2, e_2, \mathcal{D}_2))$*. Dans ce schéma,* $\mathcal{B} = \langle Tid, \mathsf{Items} \rangle$ *est le schéma d'une base de données binaire,* $\mathcal{L}_1 = 2^{\mathsf{Items}}$ *est le langage des itemsets,* $e_1 = \mathsf{freq}_r$*,* $\mathcal{D}_1 = [0, 1]$*,* $\mathcal{L}_2 = \mathcal{R}_{\mathsf{Items}}$ *est le langage des règles d'association,* $\mathcal{D}_2 = [0, 1]^2$*, et* $e_2 = (\mathsf{freq}, \mathsf{conf})$*, où* freq *et* conf *sont la fréquence et la confiance d'une règle (définition 3). On pourrait utiliser d'autres fonctions d'évaluation sur les règles comme la conviction à la place ou en plus de la confiance. La figure 1.5 présente une instance de ce schéma de base de données inductive.*

$i =$

Itemset	Fréquence
ACD	0.5
AC	0.5
∅	1
ABCD	0.33

$r =$

Corps	Tête	Fréquence	Confiance
AD	C	0.5	1
A	CD	0.5	0.75
∅	A	0.67	0.67

FIG. 1.5 – Exemple d'une instance (bd_1, i, r) d'une base de données inductive de schéma $(\mathsf{Items}, (\mathcal{L}_1, e_1, \mathcal{D}_1), (\mathcal{L}_2, e_2, \mathcal{D}_2))$. La base de données binaire bd_1 est définie dans la figure 1.1 et le schéma dans l'exemple 2.

Dans la suite, nous nous focalisons sur les requêtes d'extraction de motifs, des requêtes appelées requêtes inductives.

1.3 Formalisation des requêtes inductives

1.3.1 Calcul de théories

Nous utilisons le cadre introduit par Mannila et Toivonen [MT97] pour spécifier une tâche d'extraction dans une base de données, i.e., le résultat d'une requête inductive.

Considérons une base de données bd de schéma $\mathcal{B}$, un langage $\mathcal{L}$ de motifs et un prédicat de sélection p , i.e., une fonction de $\mathcal{L} \times \mathcal{B}$ dans $\{vrai, faux\}$. La tâche d'extraction consiste à calculer la théorie de bd relativement à $\mathcal{L}$ et p définie par

$$\mathsf{Th}(bd, \mathcal{L}, p) = \{\varphi \in \mathcal{L} \mid p(\varphi, bd) = vrai\} .$$

Définition 6 (requête inductive, résultat d'une requête inductive)
Une requête inductive σ est un triplet $(bd, \mathcal{L}, p)$. Le résultat d'une requête inductive est définie par la théorie

$$\mathsf{Th}(bd, \mathcal{L}, p) = \mathsf{Th}(\sigma) = \{\varphi \in \mathcal{L} \mid p(\varphi, bd) = vrai\} .$$

Le prédicat p est une combinaison booléenne arbitraire de contraintes servant à définir les motifs considérés comme intéressants par l'utilisateur.

Définition 7 (contrainte) *Une* contrainte *$\mathcal{C}$ est une fonction de $\mathcal{L}$ dans $\{vrai, faux\}$. On dira qu'un motif φ* satisfait *une contrainte si $\mathcal{C}(\varphi) = vrai$. Un tel motif sera dit* valide *par rapport à la contrainte $\mathcal{C}$.*

Nous avons déjà rencontré les contraintes de fréquence et de confiance minimum pour les itemsets et les règles d'association.

Exemple 3 *Considérons à nouveau la base de données binaire bd_1 définie dans la figure 1.1. Soit la requête $\sigma_1 = (bd_1, 2^{\mathsf{Items}}, p_1)$ où $p_1(S, bd) = \mathcal{C}_{0.5\text{-}\mathsf{freq}}(S, bd_1)$. Le résultat de la requête est*

$$\mathsf{Th}(\sigma_1) = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{AB}, \mathtt{AC}, \mathtt{AD}, \mathtt{CD}, \mathtt{ACD}\} .$$

1.3.2 Requête étendue

Nous avons vu dans la section précédente comment spécifier une requête inductive pour extraire des motifs valides par rapport à une contrainte.

Cependant, lorsqu'un utilisateur demande l'extraction de motifs, il veut souvent aussi connaître la valeur d'une fonction d'évaluation pour les différents motifs de manière à pouvoir sélectionner les plus intéressants. Par exemple, pour des itemsets ou des règles, la fréquence et la confiance sont considérées comme des mesures de qualité. On doit donc proposer des requêtes inductives étendues qui permettent à l'utilisateur de spécifier qu'il veut aussi la valeur d'une ou plusieurs fonctions d'évaluation pour les motifs.

Définition 8 (requête inductive étendue, résultat d'une requête étendue) *Dans ce cadre, une requête inductive étendue σ est un quintuplet $(bd, \mathcal{L}, p, e, \mathcal{D})$ où e est une fonction d'évaluation à valeurs dans $\mathcal{D}$. Le résultat d'une requête inductive étendue est définie par la collection*

$$\mathsf{Res}(bd, \mathcal{L}, p, e, \mathcal{D}) = \mathsf{Res}(\sigma) = \{(\varphi, e(\varphi, bd)) \in \mathcal{L} \times \mathcal{D} \mid p(\varphi, bd) = vrai\}$$

et la théorie de σ est définie par

$$\mathsf{Th}(bd, \mathcal{L}, p) = \mathsf{Th}(\sigma) = \{\varphi \in \mathcal{L} \mid p(\varphi, bd) = vrai\} .$$

Exemple 4 *Le résultat de la requête étendue* $\sigma'_1 = (bd_1, 2^{\mathsf{Items}}, p_1, \mathsf{freq}, [0,1])$*, où* p_1 *et* bd_1 *sont les mêmes que dans l'exemple 3, est :*

$$\mathsf{Res}(\sigma'_1) = \{(\mathtt{A}, 0.67), (\mathtt{B}, 0.5), (\mathtt{C}, 0.83), (\mathtt{D}, 0.67), (\mathtt{AB}, 0.5), (\mathtt{AC}, 0.5), (\mathtt{AD}, 0.5), (\mathtt{CD}, 0.67), (\mathtt{ACD}, 0.5)\}\,.$$

On peut remarquer qu'on peut exprimer une requête inductive étendue comme une requête inductive simple. Il suffit de poser $\mathcal{L}' = 2^{\mathsf{Items}} \times \mathcal{D}$ et de définir un nouveau prédicat par $p'((\varphi, f), bd) = (p(\varphi, bd) \wedge f = e(\varphi, bd))$.

Cependant, il est plus simple de faire la différence dans les notations entre ces deux types de requêtes car lorsque nous étudierons les stratégies d'évaluation dans la section 1.4, nous utiliserons des propriétés de $\mathcal{L}$ et p et qui ne sont pas partagées par $\mathcal{L}'$ et p'.

1.3.3 Fonctions d'évaluation

La notion de qualité des motifs est évidemment subjective et dépend de l'utilisation qu'on veut faire de ceux-ci. Il est donc naturel que de nombreux auteurs aient proposé des fonctions d'évaluation variées pour mesurer cette qualité. Les plus utilisées sont la fréquence et la confiance que nous avons déjà présentées, mais il en existe de nombreuses autres comme la conviction [BMS97, BAG00], la J-mesure [SG92], l'intérêt [BMUT97], la valeur de dépendance (*dependance value*) [Meo00] etc.

Un point commun entre les mesures les plus utilisées est qu'elles sont toutes calculables sur une règle $X \Rightarrow Y$ dès lors qu'on connaît la fréquence des itemsets sous ensembles de $X \cup Y$. L'extraction des itemsets et de leur fréquence apparaît donc comme une tâche d'extraction particulièrement importante.

Avec la fréquence (ou avec n'importe quelle autre fonction d'évaluation), nous pouvons définir des contraintes primitives de fréquence minimum freq et de fréquence maximum $\mathcal{C}_{\mu\text{-}\mathsf{freqmax}}$ (très utiles lorsqu'on veut extraire des motifs fréquents dans un jeu de données et infréquents dans un autre dans le but de découvrir des motifs discriminants entre les deux jeux de données [KdRH01, dRK01]).

Il existe d'autres fonctions d'évaluation qui ne sont pas à valeur numérique. Il s'agit des opérateurs de fermeture.

Définition 9 (fermeture, fermé, libre) *La* fermeture *d'un itemset* S *(notée* $\mathsf{ferm}(S)$*) est le plus grand sur ensemble de* S *qui a la même fréquence que* S*. Un itemset* S *est* fermé *ou* clos *si il est égal à sa propre fermeture. Un itemset est libre s'il n'est pas inclus dans la fermeture de l'un de ses sous ensembles stricts.*

La fermeture d'un itemset S contient les items qui sont impliqués logiquement par S (l'implication ne signifie pas qu'il existe un lien de causalité entre les items [BMS97], mais seulement une co-occurrence dans les données). En effet, comme S et $\mathsf{ferm}(S)$ ont la même fréquence, la règle $S \Rightarrow (\mathsf{ferm}(S) \setminus S)$ a une confiance de 1. La collection des itemsets fermés fréquents est une *représentation condensée* des itemsets fréquents. Nous introduirons ce concept de représentation condensée dans la section 1.4.3 et nous les étudierons plus en détail dans le chapitre 3.

La fermeture peut être généralisée de la manière suivante

Définition 10 (δ-fermeture, δ-libre [BBR00]) *Soit δ un entier positif. La* δ-fermeture *d'un itemset S (notée $\mathsf{ferm}_\delta(S)$) est définie par :*

$$\mathsf{ferm}_\delta(S) = \{I \in \mathsf{Items} \mid \mathsf{freq}_a(S) - \mathsf{freq}_a(S \cup \{I\}) \leq \delta\}$$

(freq_a est la fréquence absolue). Un itemset est δ-libre s'il n'est pas inclus dans la δ-fermeture de l'un de ses sous ensembles stricts.

La δ-fermeture d'un itemset peut aussi être expliquée au moyen des règles. Un item i appartient à la δ-fermeture d'un itemset S si la règle $S \Rightarrow i$ a au plus δ exceptions dans les données. On constate donc que si δ vaut 0, on retrouve la fermeture.

Exemple 5 *Dans la base de données binaire de la figure 1.1,* $\mathsf{ferm}(\mathtt{AD}) = \mathtt{ACD}$, $\mathsf{ferm}(\mathtt{D}) = \mathtt{CD}$, $\mathsf{ferm}_1(\mathtt{D}) = \mathtt{ACD}$ *et* $\mathsf{ferm}_2(\mathtt{D}) = \mathtt{ABCD}$. *Les itemsets* A, CD, ACD *sont fermés.*

Ces opérateurs de fermeture permettent également de définir des contraintes primitives :
- un itemset S satisfait $\mathcal{C}_{\mathsf{clos}}$ si il est fermé ;
- S satisfait $\mathcal{C}_{\mathsf{libre}}$ si il est libre ;
- S satisfait $\mathcal{C}_{\delta\text{-}\mathsf{libre}}$ si il est δ-libre ;

1.3.4 Contraintes syntaxiques

Définition 11 (contrainte syntaxique) *Soit $\mathcal{C}$ une contrainte, $\mathcal{C}$ est une* contrainte syntaxique *si $\mathsf{Th}(bd, 2^{\mathsf{Items}}, \mathcal{C})$ ne dépend pas de bd.*

Si S est un itemset et $\mathcal{C}_s$ une contrainte syntaxique, $\mathcal{C}_s(S, bd)$ ne dépend pas de bd. Par exemple, la contrainte $\mathcal{C}(S) = \mathtt{A} \in S$ est une contrainte syntaxique.

Ces contraintes sont un outil simple permettant à l'utilisateur de filtrer les motifs extraits. Ce filtrage est souvent indispensable car les motifs extraits peuvent être extrêmement nombreux et donc difficilement exploitables. Ainsi, il permet à un utilisateur de se focaliser sur la partie des motifs qui sont utiles pour son problème. Il peut, par exemple, spécifier qu'il n'est intéressé que par des règles d'association dont la tête est un attribut de classe.

1.3.5 Contraintes d'optimisation

Une contrainte d'optimisation est une contrainte qui indique que l'on s'intéresse aux motifs qui sont optimaux par rapport à la valeur d'une fonction d'évaluation.

On peut considérer que beaucoup d'algorithmes en apprentissage sont des algorithmes d'extraction où l'on cherche un modèle des données satisfaisant une contrainte d'optimisation. C'est le cas par exemple pour le clustering où on cherche à minimiser la similarité intra-classe et à maximiser la dissimilarité inter-classe, mais aussi pour les arbres de décision etc. Il faut remarquer que dans la plupart des cas, trouver le motif optimal est infaisable, d'où l'utilisation d'approches heuristiques.

Par contre, ce type de contrainte est assez rarement traité (à l'exception notable de [BA99]) pour les règles d'association et les itemsets autrement que par post-traitement.

1.3.6 Contraintes complexes

Nous pouvons maintenant définir des contraintes complexes comme des combinaisons (à l'aide d'opérateurs booléens) des contraintes primitives que nous avons vues dans les sections précédentes. L'évaluation d'une requête inductive consiste à découvrir tous les motifs qui satisfont une contrainte complexe quelconque. Un certain nombre de contraintes complexes ont été étudiées : l'extraction de fermés fréquents [PBTL99b, ZH02, PHM00] (contrainte $\mathcal{C}_{\mathsf{clos}} \wedge \mathcal{C}_{\gamma\text{-}\mathsf{freq}}$), de libres ou de δ-libres fréquents [BB00, BBR00] (contraintes $\mathcal{C}_{\mathsf{libre}} \wedge \mathcal{C}_{\gamma\text{-}\mathsf{freq}}$, $\mathcal{C}_{\delta\text{-}\mathsf{libre}} \wedge \mathcal{C}_{\gamma\text{-}\mathsf{freq}}$), de motifs fréquents dans une base de données et infréquents dans une autre [dRK01] (contrainte $\mathcal{C}_{\gamma\text{-}\mathsf{freq}}(bd_1) \wedge \mathcal{C}_{\mu\text{-}\mathsf{freqmax}}(bd_2)$).

1.4 Évaluation des requêtes inductives

Après avoir défini les bases de données inductives et les requêtes inductives, nous allons nous intéresser au problème de l'évaluation des requêtes inductives, i.e. l'extraction de motifs. Les extractions de motifs sont généralement très coûteuses pour plusieurs raisons.

- Tout d'abord, la taille du langage de motifs $\mathcal{L}$ peut être très grande, voire infinie (séquences avec répétitions). Dans le cas des itemsets, la taille de $\mathcal{L}$ vaut 2^n où n est le nombre d'items différents.
- Ensuite le coût de l'évaluation du prédicat de sélection p est souvent important. En effet, son calcul nécessite souvent un parcours de la base de données afin de calculer la valeur d'une ou plusieurs fonctions d'évaluation. Dans le cas des itemsets, le prédicat de sélection inclut généralement la contrainte de fréquence minimale. Il faut donc calculer celle-ci. Or il n'est pas rare que la table utilisée pour ce calcul contienne plusieurs centaines de milliers voire plusieurs millions de lignes.

Ces raisons font qu'il n'est pas possible d'utiliser une stratégie *"générer et tester"* pour évaluer le résultat d'une requête inductive $\sigma = (bd, \mathcal{L}, p)$. Cette stratégie consiste à générer tous les motifs φ du langage $\mathcal{L}$ et à vérifier si $p(\varphi, bd)$ est vrai. Il faut donc concevoir des stratégies plus évoluées.

1.4.1 Les contraintes anti-monotones et la frontière

Comme il n'est généralement pas possible d'explorer l'ensemble $\mathcal{L}$ de tous les motifs de manière exhaustive, il faut réussir à n'en parcourir qu'une partie. Cependant, il faut être sûr d'avoir découvert tous les motifs de $\mathcal{L}$ qui satisfont au prédicat p, i.e., il faut être complet. Cela signifie qu'à chaque fois que l'on décide de ne pas explorer toute une partie de l'ensemble $\mathcal{L}$ (on dira que l'on élague $\mathcal{L}$), il faut pouvoir prouver qu'aucun motif φ satisfaisant le prédicat p ne s'y trouve (dans ce cas on dira que l'élagage est sûr). Autrement dit, on cherche à n'explorer qu'une partie $\mathcal{P}$ de $\mathcal{L}$ en s'assurant que la théorie $\mathsf{Th}(\sigma) = \mathsf{Th}(bd, \mathcal{L}, p)$ est incluse dans $\mathcal{P}$.

Pour y parvenir, il faut utiliser de manière active le prédicat (ou la contrainte) p pendant l'extraction et non pas seulement après comme dans la stratégie "générer et tester". Le prédicat p sera donc "poussé" pendant l'extraction pour élaguer des portions de l'espace de recherche $\mathcal{L}$.

Supposons que l'on souhaite extraire tous les itemsets 0.5-fréquents dans une base de données binaire bd. Si, lors de l'exploration de $\mathcal{L} = 2^{\mathsf{Items}}$, un itemset S qui n'est pas fréquent est découvert alors, en vertu de la proposition 1, tout sur ensemble T de S a une fréquence inférieure à celle de S. Par conséquent, T ne peut pas être fréquent. On vient ainsi de montrer

qu'il n'est pas nécessaire d'explorer les sur ensembles de S, aucun d'entre-eux ne pouvant être fréquent.

Cette méthode d'élagage est en fait applicable à une large classe de contraintes, les contraintes anti-monotones.

Les contraintes anti-monotones

Nous allons d'abord définir une relation de spécialisation sur le langage des motifs.

Définition 12 (relation de spécialisation) *Une* relation de spécialisation $\preceq$ *sur les motifs est une relation d'ordre partiel. Un motif φ est* plus général *que μ si $\varphi \preceq \mu$ (μ est* plus spécifique *que φ). La notation $\mu \prec \varphi$ est une abréviation de $\mu \preceq \varphi \wedge \mu \neq \varphi$. Un motif φ* couvre *un motif μ si $\mu \prec \varphi$ et s'il n'existe pas de motif λ tel que $\mu \prec \lambda \prec \varphi$.*

Dans le cas des itemsets, la relation de spécialisation que nous utilisons est tout simplement l'inclusion ensembliste. Nous supposerons dans la suite que le langage $\mathcal{L}$ des motifs est muni de cette relation de spécialisation. Nous pouvons alors définir les contraintes anti-monotone par rapport à cette relation de spécialisation.

Définition 13 (contrainte anti-monotone) *Une contrainte $\mathcal{C}$ est* anti-monotone *si*

$$\forall bd\, \forall \varphi\, \forall \mu\, (\mu \preceq \varphi \wedge \mathcal{C}(\varphi, bd)) \Rightarrow \mathcal{C}(\mu, bd).$$

Ces contraintes sont appelées contraintes monotones par Mannila et Toivonen [MT97] et contraintes anti-monotones par Ng et al. [NLHP98]. Nous prendrons la terminologie et la notation de ces derniers et nous utiliserons la notation $\mathcal{C}_{\mathsf{am}}$ pour désigner une contrainte anti-monotone.

Une contrainte anti-monotone très utilisée est donc la contrainte de fréquence minimum : si un itemset S est fréquent, alors tout sous ensemble de S est également fréquent. Comme nous l'avons vu plus haut, c'est surtout la contraposée de cette propriété qui est utilisée pour élaguer l'espace de recherche 2^{Items} : si un itemset S est infréquent alors tout sur ensemble de S est infréquent.

Exemple 6 *Les contraintes $\mathcal{C}_{\gamma\text{-}\mathsf{freq}}$, $\mathcal{C}_1(S) = (S \cap \mathtt{BCD} = \emptyset)$, $\mathcal{C}_2(S) = (|S \cap \mathtt{BCD}| < 3)$ et $\mathcal{C}_3(S) = (S \subseteq \mathtt{ABDE})$ sont des contraintes anti-monotones.*

On peut également construire de nouvelles contraintes anti-monotones par conjonction ou disjonction de contraintes anti-monotones.

Proposition 2 *Soit $\mathcal{C}_{\mathsf{am}_1}$ et $\mathcal{C}_{\mathsf{am}_2}$ deux contraintes anti-monotones, alors la conjonction $\mathcal{C}_{\mathsf{am}_1} \wedge \mathcal{C}_{\mathsf{am}_2}$ et la disjonction $\mathcal{C}_{\mathsf{am}_1} \vee \mathcal{C}_{\mathsf{am}_2}$ de ces deux contraintes sont aussi des contraintes anti-monotones.*

Preuve : Montrons que $\mathcal{C}_{\mathsf{am}_1} \wedge \mathcal{C}_{\mathsf{am}_2}$ est anti-monotone.
Soit φ un motif tel que $[\mathcal{C}_{\mathsf{am}_1} \wedge \mathcal{C}_{\mathsf{am}_2}](\varphi)$ est vraie et soit $\mu \preceq \varphi$, alors $\mathcal{C}_{\mathsf{am}_1}(\varphi)$ et $\mathcal{C}_{\mathsf{am}_2}(\varphi)$ sont vraies et par anti-monotonicité de ces deux contraintes, $\mathcal{C}_{\mathsf{am}_1}(\mu)$ et $\mathcal{C}_{\mathsf{am}_2}(\mu)$ sont vraies. Ainsi, $[\mathcal{C}_{\mathsf{am}_1} \wedge \mathcal{C}_{\mathsf{am}_2}](\mu)$ est vraie et donc cette contrainte est bien anti-monotone.
Montrons que $\mathcal{C}_{\mathsf{am}_1} \vee \mathcal{C}_{\mathsf{am}_2}$ est anti-monotone.
Soit φ un itemset tel que $[\mathcal{C}_{\mathsf{am}_1} \vee \mathcal{C}_{\mathsf{am}_2}](\varphi)$ est vraie et soit $\mu \preceq \varphi$, alors soit $\mathcal{C}_{\mathsf{am}_1}(\varphi)$

est vraie, soit $\mathcal{C}_{\mathsf{am}_2}(\varphi)$ est vraie et par anti-monotonicité de ces deux contraintes, soit $\mathcal{C}_{\mathsf{am}_1}(\mu)$ est vraie, soit $\mathcal{C}_{\mathsf{am}_2}(\mu)$ est vraie. Ainsi, $[\mathcal{C}_{\mathsf{am}_1} \vee \mathcal{C}_{\mathsf{am}_2}](\mu)$ est vraie et donc cette contrainte est bien anti-monotone. □

Les frontières d'une collection de motifs

Si un motif φ vérifie une contrainte anti-monotone $\mathcal{C}_{\mathsf{am}}$, tout motif plus général que φ la vérifie également. On peut donc caractériser $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}})$ par l'ensemble de ses motifs les plus spécifiques. Cet ensemble est appelé la frontière positive de la théorie. De manière duale on peut également définir une frontière négative contenant les motifs les plus généraux qui ne satisfont pas la contrainte. Cette caractérisation a été introduite en extraction de connaissances par Mannila et Toivonen [MT97].

Définition 14 (frontière positive, frontière négative) *Soit $\mathcal{S}$ une collection de motifs. La* frontière positive *de $\mathcal{S}$ est la collection des motifs les plus spécifiques de $\mathcal{S}$,*

$$\mathcal{B}d^+(\mathcal{S}) = \max_{\preceq}(\mathcal{S}) = \{\varphi \in \mathcal{S} \mid \nexists \mu(\mu \in \mathcal{S} \wedge \varphi \prec \mu)\}\,.$$

La frontière négative *de $\mathcal{S}$ est la collection des motifs les plus généraux qui ne sont pas dans $\mathcal{S}$,*

$$\mathcal{B}d^-(\mathcal{S}) = \min_{\preceq}(\mathcal{L} \setminus \mathcal{S}) = \{\varphi \notin \mathcal{S} \mid \nexists \mu(\mu \notin \mathcal{S} \wedge \mu \prec \varphi)\}\,.$$

Finalement la frontière $\mathcal{B}d$ est la réunion de la frontière positive et de la frontière négative.

Les frontières permettent de caractériser les motifs qui sont dans $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}})$ et ceux qui n'y sont pas.

Proposition 3 *La collection de motifs $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}})$ est exactement la collection des motifs qui sont plus généraux qu'un motif de $\mathcal{B}d^+(\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}}))$,*

$$\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}}) = \left\{\varphi \in \mathcal{L} \mid \exists \mu \in \mathcal{B}d^+(\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}})), \varphi \preceq \mu\right\}.$$

De manière duale, la collection des motifs qui ne sont pas dans $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}})$ est exactement la collection des motifs qui sont plus spécifiques qu'un motif de $\mathcal{B}d^-(\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}}))$.

$$\mathcal{L} \setminus \mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}}) = \left\{\varphi \in \mathcal{L} \mid \exists \mu \in \mathcal{B}d^-(\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}})), \mu \preceq \varphi\right\}.$$

Preuve : D'après la propriété d'anti-monotonicité, tout motif plus général qu'un motif qui vérifie $\mathcal{C}_{\mathsf{am}}$ vérifie également $\mathcal{C}_{\mathsf{am}}$. Par conséquent, tous les motifs plus généraux que des éléments de $\mathcal{B}d^+(\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}}))$ vérifient $\mathcal{C}_{\mathsf{am}}$. Réciproquement et par définition de $\mathcal{B}d^+(\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}}))$, tout motif vérifiant $\mathcal{C}_{\mathsf{am}}$ est nécessairement plus général qu'un élément de $\mathcal{B}d^+(\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}}))$. On montre de la même manière la deuxième partie de la proposition. □

Exemple 7 *Soit $\mathcal{C}(S) = \mathcal{C}_{2\text{-}\mathsf{freq}}(S) \wedge \mathtt{BC} \not\subseteq S$. Cette contrainte est anti-monotone. Sur la base de données bd_1 de la figure 1.1 page 20, les frontières de $\mathsf{Th}(bd_1, 2^{\mathsf{Items}}, \mathcal{C})$ sont :*

$$\mathcal{B}d^+(\mathsf{Th}(bd_1, 2^{\mathsf{Items}}, \mathcal{C})) = \{\mathtt{CE}, \mathtt{ABC}, \mathtt{ABD}, \mathtt{ABE}\}$$
$$\mathcal{B}d^-(\mathsf{Th}(bd_1, 2^{\mathsf{Items}}, \mathcal{C})) = \{\mathtt{CD}, \mathtt{DE}, \mathtt{ACE}, \mathtt{BCE}\}$$

La figure 1.6 page suivante représente la théorie $\mathsf{Th}(bd_1, 2^{\mathsf{Items}}, \mathcal{C})$ et ses frontières.

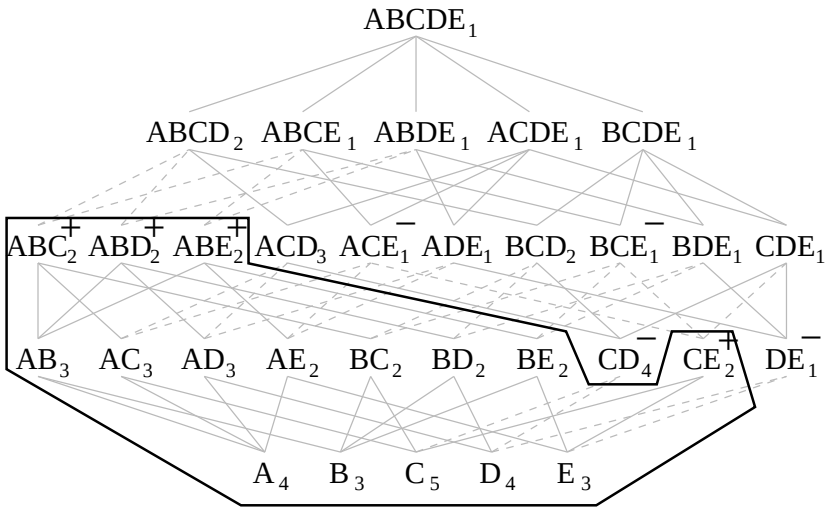


FIG. 1.6 – La théorie $\mathsf{Th}(bd_1, 2^{\mathsf{Items}}, \mathcal{C})$ et ses frontières. La fréquence de chaque itemset dans bd_1 est indiquée en indice. Les éléments de la frontière positive ont un "+" en exposant et ceux de la frontière négative, un "-". Les arêtes qui traversent la frontière sont en pointillés.

1.4.2 L'algorithme Guess and correct

Dans cette section, nous présentons l'algorithme Guess and correct de Mannila et Toivonen [MT97] pour calculer $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}})$.

Cet algorithme utilise une collection de motifs qui est une approximation de $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}})$. Meilleure est cette approximation, plus efficace est l'algorithme. Cette collection de motifs qui sert d'approximation doit être fermée vers le bas. Cela signifie que si un motif φ est dans la collection, alors tous les motifs plus généraux que φ doivent également y être.

Algorithme 1 (Guess and correct)

Entrée : une base de données bd, une contrainte anti-monotone $\mathcal{C}_{\mathsf{am}}$ et une collection de motifs $\mathcal{S}$ fermée vers le bas
Sortie : $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}})$

1 $\mathcal{E} := \emptyset$
2 $\mathsf{Cand} := \mathcal{B}d^+(\mathcal{S})$
3 **tant que** $\mathsf{Cand} \neq \emptyset$ **faire**
4 $\quad \mathcal{E} := \mathcal{E} \cup \mathsf{Cand}$
5 $\quad \mathcal{S} := \mathcal{S} \setminus \{\varphi \in \mathsf{Cand} \mid \mathcal{C}_{\mathsf{am}}(\varphi, bd) = faux\}$
6 $\quad \mathsf{Cand} := \mathcal{B}d^+(\mathcal{S}) \setminus \mathcal{E}$
7 **fait**
8 $\mathsf{Cand} := \mathcal{B}d^-(\mathcal{S}) \setminus \mathcal{E}$
9 **tant que** $\mathsf{Cand} \neq \emptyset$ **faire**
10 $\quad \mathcal{E} := \mathcal{E} \cup \mathsf{Cand}$
11 $\quad \mathcal{S} := \mathcal{S} \cup \{\varphi \in \mathsf{Cand} \mid \mathcal{C}_{\mathsf{am}}(\varphi, bd) = vrai\}$
12 $\quad \mathsf{Cand} := \mathcal{B}d^-(\mathcal{S}) \setminus \mathcal{E}$
13 **fait**
14 **retourne** $\mathcal{S}$

Dans la première boucle (lignes 3 à 7), l'algorithme retire de $\mathcal{S}$ les motifs qui ne vérifient pas $\mathcal{C}_{\mathsf{am}}$. Pour cela, les itemsets maximaux de $\mathcal{S}$ sont calculés (ligne 2 et 6), puis ceux qui ne vérifient pas $\mathcal{C}_{\mathsf{am}}$ sont retirés (ligne 5). L'ensemble $\mathcal{E}$ sert à ne pas considérer plusieurs fois le même motif. Cette étape est répétée jusqu'à ce que tous les itemsets maximaux de $\mathcal{S}$ vérifient $\mathcal{C}_{\mathsf{am}}$. À ce moment on est sûr, grâce à l'anti-monotonicité de $\mathcal{C}_{\mathsf{am}}$, que $\mathcal{S} \subseteq \mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}})$. Dans la seconde boucle (lignes 9 à 13), la collection $\mathcal{S}$ est étendue de manière à ce qu'elle contienne toute la théorie $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}})$ à la sortie de la boucle. Pour cela, l'algorithme calcule la frontière négative de $\mathcal{S}$ (ligne 8 et 12), c'est-à-dire l'ensemble des itemsets qui sont "juste en dehors" de $\mathcal{S}$. Ensuite, ceux de ces itemsets qui vérifient $\mathcal{C}_{\mathsf{am}}$ sont ajoutés à $\mathcal{S}$. Le processus cesse dès qu'aucun des itemsets de la frontière négative de $\mathcal{S}$ ne vérifie $\mathcal{C}_{\mathsf{am}}$. On est alors certain que $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}}) \subseteq \mathcal{S}$.

Il y a deux cas exceptionnels, lorsque $\mathcal{S} = \emptyset$ ou lorsque $\mathcal{S} = \mathcal{L}$. Si au début de l'algorithme $\mathcal{S} = \emptyset$ alors, à la ligne 2, Cand est vide et la première boucle n'est jamais parcourue. Ensuite la deuxième boucle correspond à un algorithme niveau par niveau qui se comprend très bien sur l'extraction d'itemsets fréquents. Dans ce cas, la relation de spécialisation est tout simplement l'inclusion et la contrainte de fréquence est bien anti-monotone par rapport à ce prédicat. L'algorithme commence par considérer les itemsets de taille 1, puis élimine ceux qui ne sont pas fréquents (ligne 11) et génère les itemsets de taille 2 (ligne 12). À ce point de l'algorithme, la collection $\mathcal{S}$ contient les itemsets fréquents de taille 1. Les itemsets de taille 2 potentiellement fréquents sont ceux dont tous les sous ensembles sont fréquents, ce sont donc les itemsets de taille 2 de la frontière négative de $\mathcal{S}$. L'algorithme continue ainsi à explorer le treillis des itemsets en partant des itemsets les plus petits et en progressant vers les itemsets les plus longs. L'algorithme s'arrête dès que tous les itemsets de la frontière négative de $\mathcal{S}$ sont tous infréquents. Ce parcours est efficace car l'algorithme ne va considérer que les itemsets dont tous les sous ensembles sont fréquents. C'est-à-dire qu'il ne considérera jamais un itemset ayant un sous ensemble infréquent.

Le deuxième cas extrême correspond à $\mathcal{S} = \mathcal{L}$ (ce cas de figure n'est évidemment possible que si $\mathcal{L}$ est fini). Dans ce cas, seule la première boucle de l'algorithme est parcourue. On se trouve dans une situation duale de celle du cas précédent. L'algorithme part des motifs les plus spécifiques et va vers les motifs les plus généraux jusqu'à ce qu'il découvre la frontière.

Cet algorithme est particulièrement intéressant du fait de sa généralité. On peut l'appliquer à chaque fois que l'on veut calculer une théorie $\mathsf{Th}(bd, \mathcal{L}, p)$ pour laquelle le prédicat de sélection est anti-monotone par rapport à la relation de spécialisation et que l'on dispose d'une approximation de la théorie.

Beaucoup d'algorithmes d'extraction de motifs sont basés sur ce modèle, bien qu'en général seule la seconde partie de l'algorithme soit effectivement utilisée. C'est-à-dire que les extractions commencent généralement avec une approximation $\mathcal{S}$ vide, c'est le cas par exemple de l'algorithme Apriori [AMS+96].

Avec l'algorithme Sampling [Toi96], Toivonen propose de calculer les itemsets fréquents dans une base de données bd en deux étapes. Dans la première étape, cet algorithme extrait un échantillon bd_1 de la base de données bd. Ensuite, il calcule la théorie $\mathsf{Th}(bd_1, 2^{\mathsf{Items}}, \mathcal{C}_{\gamma\text{-}\mathsf{freq}})$ avec l'algorithme Guess and correct avec comme paramètre une approximation $\mathcal{S}$ vide. Dans la seconde étape, Guess and correct est réutilisé pour calculer la théorie $\mathsf{Th}(bd, 2^{\mathsf{Items}}, \mathcal{C}_{\gamma\text{-}\mathsf{freq}})$. Dans ce cas, la théorie calculée sur l'échantillon bd_1 lors de la première étape est utilisée comme approximation $\mathcal{S}$ de la théorie sur la base de données complète. Cette méthode est très efficace car généralement l'approximation faite dans la première étape est très bonne, la seconde étape est donc très rapide. De plus, comme la première étape est faite sur un échantillon, elle est beaucoup plus rapide qu'une extraction sur la base de données complète.

Le gain est à peu près égal au rapport entre la taille de la base de données complète et la taille de l'échantillon. En fait, ce gain peut même être plus important. En effet, on choisira la taille de l'échantillon de manière à ce qu'il tienne entièrement en mémoire centrale. On évite ainsi les accès disques extrêmement coûteux lorsque la base de données complète ne tient pas en mémoire.

Le principe de l'algorithme Guess and correct est aussi utilisé lorsqu'on veut mettre à jour une collection de motifs après une modification de la base de données. Dans ce cas, la théorie calculée avant modification de la base de données sert d'approximation de la théorie après modification. Si les modifications sont suffisamment peu importantes, l'ancienne théorie sera une très bonne approximation. Cela permet d'effectuer la mise à jour de la théorie sans devoir la recalculer entièrement [AFLM99, TBAR97, AEMT01].

Complexité

Il est facile de montrer que l'algorithme Guess and correct parcourt toute la frontière de $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}})$ pour calculer cette théorie. Mannila et Toivonen ont montré que c'est en fait une propriété générale. On considère un modèle de calcul dans lequel un algorithme ne peut accéder aux données bd qu'au travers de questions du type : est-ce que $\mathcal{C}_{\mathsf{am}}$ est vrai sur le motif φ dans bd ? Dans ce cas, on a le théorème suivant :

Théorème 1 ([MT97]) *Considérons un algorithme pouvant calculer $\mathcal{T} = \mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}})$ pour toute contrainte anti-monotone $\mathcal{C}_{\mathsf{am}}$. On suppose que la seule source d'information de cet algorithme est l'ensemble des valeurs $\mathcal{C}_{\mathsf{am}}(\varphi)$ pour chaque motif φ. Alors cet algorithme devra utiliser les valeurs $\mathcal{C}_{\mathsf{am}}(\varphi)$ pour tout motif φ appartenant à un sur ensemble de $\mathcal{B}d(\mathcal{T}) = \mathcal{B}d^+(\mathcal{T}) \cup \mathcal{B}d^-(\mathcal{T})$ pour pouvoir calculer la théorie.*

Pour prouver ce théorème, on raisonne par l'absurde. On montre que s'il n'explore pas les deux frontières alors on peut construire deux contraintes $\mathcal{C}_{\mathsf{am}}$ et $\mathcal{C}'_{\mathsf{am}}$ avec lesquelles l'algorithme aura le même comportement et qui ont pourtant deux théories différentes, on aura donc montré qu'un algorithme qui n'explore pas la frontière ne peut pas être correct et complet.

Preuve : Supposons que l'algorithme n'évalue pas $\mathcal{C}_{\mathsf{am}}$ pour un motif φ appartenant à $\mathcal{B}d^+(\mathcal{T}) \cup \mathcal{B}d^-(\mathcal{T})$.
Premier cas, φ appartient à $\mathcal{B}d^+(\mathcal{T})$. Définissons une seconde contrainte $\mathcal{C}'_{\mathsf{am}}$ telle que $\mathcal{C}'_{\mathsf{am}}(\varphi) = faux$ et pour tout $\mu \neq \varphi$, $\mathcal{C}'_{\mathsf{am}}(\mu) = \mathcal{C}_{\mathsf{am}}(\mu)$. Cette contrainte $\mathcal{C}'_{\mathsf{am}}$ est bien anti-monotone. En effet, comme $\varphi \in \mathcal{B}d^+(\mathcal{T})$, pour tous les motifs λ plus généraux que φ, $\mathcal{C}'_{\mathsf{am}}(\lambda) = \mathcal{C}_{\mathsf{am}}(\lambda) = faux$. L'exécution de l'algorithme sera la même que l'on utilise la contrainte $\mathcal{C}_{\mathsf{am}}$ ou $\mathcal{C}'_{\mathsf{am}}$. En effet, l'algorithme ne calcule jamais $\mathcal{C}_{\mathsf{am}}(\varphi)$. Du point de vue de l'algorithme, les deux contraintes sont donc identiques. Il donnera donc la même réponse pour les deux contraintes alors que $\mathcal{T} \neq \mathsf{Th}(bd, 2^{\mathsf{Items}}, \mathcal{C}'_{\mathsf{am}})$ (car on a choisi $\mathcal{C}'_{\mathsf{am}}$ et φ de manière à ce que $\varphi \in \mathcal{T}$ et $\varphi \notin \mathsf{Th}(bd, 2^{\mathsf{Items}}, \mathcal{C}'_{\mathsf{am}})$).
Deuxième cas, φ appartient à $\mathcal{B}d^-(\mathcal{T})$. Définissons une seconde contrainte $\mathcal{C}'_{\mathsf{am}}$ telle que $\mathcal{C}'_{\mathsf{am}}(\varphi) = vrai$ et pour tout $\mu \neq \varphi$, $\mathcal{C}'_{\mathsf{am}}(\mu) = \mathcal{C}_{\mathsf{am}}(\mu)$. Cette contrainte $\mathcal{C}'_{\mathsf{am}}$ est elle aussi anti-monotone. Ici aussi, l'exécution de l'algorithme sera la même que l'on utilise la contrainte $\mathcal{C}_{\mathsf{am}}$ ou $\mathcal{C}'_{\mathsf{am}}$. Il donnera donc encore la même réponse pour les deux contraintes alors que $\mathcal{T} \neq \mathsf{Th}(bd, 2^{\mathsf{Items}}, \mathcal{C}'_{\mathsf{am}})$.
Donc finalement, si l'algorithme calcule correctement $\mathcal{T}$ pour toute contrainte anti-monotone $\mathcal{C}_{\mathsf{am}}$, il doit explorer $\mathcal{B}d^+(\mathcal{T}) \cup \mathcal{B}d^-(\mathcal{T})$. □

1.4.3 Stratégies avancées

Pousser les contraintes

Les collections de motifs extraites par un algorithme utilisant seulement la contrainte de fréquence sont souvent très grandes et ne sont donc pas utilisables directement par l'utilisateur. Ce dernier devra sélectionner dans une étape de post-traitement les motifs qui l'intéressent. Il peut par exemple utiliser des contraintes syntaxiques pour cela.

Toutefois, l'extraction de motifs, du fait de la taille souvent très importante de l'espace de recherche, est une opération très coûteuse. Si l'utilisateur spécifie précisément les motifs qui l'intéressent par le biais de contraintes avant l'étape d'extraction, il serait intéressant de les utiliser pendant l'extraction pour la rendre plus rapide. Idéalement, le temps pris par l'extraction devrait être proportionnel à la taille de la collection de motifs extraite. Ceci n'est possible que si les contraintes sont activement utilisées pendant l'extraction et non pas seulement dans une étape de post-traitement.

Dans la section précédente, nous avons vu comment pousser une contrainte anti-monotone pendant l'extraction des motifs pour élaguer l'espace de recherche. Bien qu'il y ait de nombreuses contraintes anti-monotones, toutes les contraintes ne le sont pas, il faut donc trouver d'autres stratégies.

On peut chercher à récrire la contrainte sous la forme d'une conjonction de deux contraintes dont l'une est anti-monotone. Il sera alors possible de pousser la contrainte anti-monotone et de profiter de l'élagage qu'elle apporte. Ensuite, la collection des motifs vérifiant la contrainte anti-monotone est filtrée dans une étape de post-traitement pour éliminer les motifs ne vérifiant pas l'autre contrainte.

Il n'est pas simple de récrire une contrainte arbitraire sous cette forme. Cependant, la contrainte d'extraction est souvent une simple conjonction de contraintes primitives (par exemple une contrainte de fréquence minimum et des contraintes syntaxiques). Dans ce cas, il suffit de regrouper celles de ces contraintes qui sont anti-monotones et de les utiliser comme une seule contrainte anti-monotone (une conjonction de contraintes anti-monotone est anti-monotone).

Pour faire apparaître une contrainte anti-monotone, il peut aussi être utile de relaxer la contrainte. Étant donné une contrainte non anti-monotone $\mathcal{C}$, on cherche une contrainte monotone $\mathcal{C}_{\mathsf{am}}$ telle que $\mathcal{C}$ implique $\mathcal{C}_{\mathsf{am}}$. Ainsi, $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}) \subseteq \mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}})$ et on peut pousser la contrainte $\mathcal{C}_{\mathsf{am}}$ pour calculer $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}})$ et ensuite filtrer les motifs qui vérifient $\mathcal{C}$ dans une étape de post-traitement. On profite ainsi de l'élagage supplémentaire apporté par la contrainte $\mathcal{C}_{\mathsf{am}}$. Dans [CBM02], un exemple de la relaxation d'une contrainte de similarité minimum par rapport à un motif de référence est présenté. Cette contrainte n'est pas anti-monotone mais les auteurs exhibent une contrainte anti-monotone qui peut être poussée lors de l'extraction. La relaxation de contrainte est également utilisée dans [NLHP98] et [GRS99]. Dans tous les cas, la relaxation est faite de manière ad-hoc pour certaines contraintes que l'on ne pourrait pas sinon pousser pendant l'étape d'exploration de l'espace de recherche.

Enfin, les contraintes anti-monotones ne sont pas les seules contraintes que l'on peut pousser. Nous verrons dans le chapitre 2 d'autres classes de contraintes qui peuvent être poussées pendant l'exploration de l'espace de recherche.

Représentations condensées et approximations

Il existe toute une classe de problèmes pour lesquels la stratégie consistant à pousser les contraintes n'est plus suffisante. Ce sont les cas où les motifs satisfaisant la contrainte sont encore trop nombreux pour pouvoir être extraits de manière efficace. On trouve ce cas de figure dans l'extraction d'itemsets fréquents lorsque la base de données est fortement corrélée. Cela se traduit par l'existence d'itemsets fréquents très grands même pour des seuils de fréquence relativement élevés. Si, par exemple, il existe un itemset fréquent de taille 40, un algorithme procédant niveau par niveau devra considérer les 2^{40} sous ensembles de cet itemset. Cela pose des problèmes au niveau du temps de calcul de la collection mais aussi au niveau sa taille et donc de son stockage et de sa manipulation.

Dans ce cas, les représentations condensées des itemsets fréquents apportent une solution efficace. Ce concept de représentation condensée est général et a été introduit par Mannila et Toivonen [MT96b]. Idéalement, une représentation condensée des itemsets fréquents doit être plus compacte que la collection complète des itemsets fréquents, elle doit pouvoir être extraite efficacement des données et on doit pouvoir régénérer la collection des itemsets fréquents et leurs fréquences (i.e., $\mathsf{Res}(bd, 2^{\mathsf{Items}}, \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq}, [0,1])$) efficacement à partir de la représentation condensée.

Nous formalisons maintenant ces notions pour donner des exemples de représentations condensées utiles pour les requêtes de fréquence. Adaptons la définition formelle d'une représentation ϵ-adéquate [MT96b].

Définition 15 (Représentation ϵ-adéquate) *Soit $\mathcal{S}$ un ensemble et $\mathcal{Q}$ un ensemble de fonctions de $\mathcal{S}$ dans $[0,1]$, une représentation ϵ-adéquate pour $\mathcal{S}$ par rapport à $\mathcal{Q}$ est un ensemble $\mathcal{C}$, une mise en correspondance $rep : \mathcal{S} \to \mathcal{C}$ et une fonction $m : \mathcal{Q} \times \mathcal{C} \to [0,1]$ telle que*

$$\forall Q \in \mathcal{Q}, \forall s \in \mathcal{S}, |Q(s) - m(Q, rep(s))| \leq \epsilon.$$

Exemple 8 *Considérons d'abord une illustration extraite de [Byk99]. Soit $\mathcal{S}$ l'ensemble des nombres flottants* `float`*. Soit $\mathcal{Q}$ un ensemble de fonctions de $\mathbb{R}$ dans $[0,1]$, par exemple $\{\sin, \cos\}$. Une représentation de $\mathcal{S}$ par rapport à $\mathcal{Q}$ peut être le couple $(\mathcal{C}, m)$, où $\mathcal{C}$ est l'ensemble des entiers courts,* `short`*, et m est un moyen alternatif pour évaluer une fonction de $\mathcal{Q}$, e.g. en utilisant une table des valeurs de* sin *et* cos *pour tous les angles compris dans $\{0, 1, \ldots, 359\}$. Maintenant, on a un nouveau moyen de calculer $\sin(x)$ et $\cos(x)$. Par exemple, au lieu de calculer directement $\sin(0.1)$, on utilise $rep(0.1) = round(0.1 \times 360/2\pi) \mod 360$, i.e., 6. Quand la valeur de $\sin(0.1)$ est nécessaire, on peut utiliser $m(\sin, 6)$ qui retourne la valeur stockée dans la table associée à* sin*. Le résultat est approché mais l'erreur commise est bornée et le résultat obtenu à moindre coût.*

Voyons maintenant des représentations ϵ-adéquates dans le cadre de l'ECD.

Exemple 9 (Approximation de la fréquence à l'aide des itemsets fréquents) *Soit la classe $\mathcal{B}$ de toutes les bases binaires sur l'ensemble d'items* Items*. On considère $\mathcal{Q}$ la collection de toutes les requêtes qui retournent la fréquence d'un itemset inclus dans* Items*. Si on note Q_S une requête de $\mathcal{Q}$ qui demande la fréquence de l'itemset S alors $\mathcal{Q} = \{Q_S \mid S \subseteq \mathsf{Items}\}$ et la valeur de Q_S sur l'instance $bd \in \mathcal{B}$ est définie comme $Q_S(bd) = \mathsf{freq}(S, bd)$.*

Un exemple de représentation $\gamma/2$-adéquate pour $\mathcal{B}$ par rapport à $\mathcal{Q}$ est la représentation de $bd \in \mathcal{B}$ au moyen de $\mathsf{Res}(bd, 2^{\mathsf{Items}}, \mathcal{C}_{\gamma\text{-}\mathsf{freq}})$. Ses composantes $rep, \mathcal{C}$ et m sont définies comme

suit. $\forall bd \in \mathcal{B}, rep(bd) = \mathsf{Res}(bd, 2^{\mathsf{Items}}, \mathcal{C}_{\gamma\text{-}\mathsf{freq}})$, $\mathcal{C} = \{rep(bd) \mid bd \in \mathcal{B}\}$, $\forall Q_S \in \mathcal{Q}, \forall c \in \mathcal{C}$, *si* $\exists (S, \alpha) \in rep(bd)$ *alors* $m(Q_S, c) = \alpha$ *sinon* $m(Q_S, c) = \gamma/2$. *Autrement dit, la fréquence d'un itemset* S *est approchée par* $\mathsf{freq}(S, bd)$ *quand il est* γ*-fréquent et par* $\gamma/2$ *sinon.*

On vérifie facilement que c'est une représentation $\gamma/2$*-adéquate pour* $\mathcal{B}$ *par rapport à* $\mathcal{Q}$ *puisque* $\forall Q_S \in \mathcal{Q}, \forall bd \in \mathcal{B}, |Q_S(bd) - m(Q_S, rep(bd))| \leq \gamma/2$.

Les représentations ϵ-adéquates qui nous intéressent particulièrement sont dites représentations condensées lorsque leurs structures ont une taille plus petite que la taille des structures initiales.

La frontière positive $\mathcal{B}d^+(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}})$ d'une collection de motifs vérifiant une contrainte anti-monotone $\mathcal{C}_{\mathsf{am}}$ est une bonne représentation condensée. Elle permet de répondre exactement aux questions du type "est-ce que le motif φ vérifie la contrainte anti-monotone ?". Par contre, si on veut utiliser la frontière positive et la fréquence des motifs de celle-ci pour calculer des approximations des fréquences de tous les motifs, l'incertitude est très grande :

Exemple 10 (Approximation de la fréquence à l'aide de la frontière positive) *On considère le même contexte et les mêmes requêtes que dans l'exemple 9. Nous allons construire une représentation utilisant la frontière positive. Soit* γ *un seuil de fréquence dans* $[0, 1]$. *Prenons l'ensemble* $rep(bd) = \left\{(S, \mathsf{freq}(S, bd)) \mid S \in \mathcal{B}d^+(bd, \mathcal{L}, \mathcal{C}_{\gamma\text{-}\mathsf{freq}})\right\}$ *comme représentation pour* $bd \in \mathcal{B}$ *par rapport à* $\mathcal{Q}$. *Soit* S *un itemset, s'il existe* T *dans* $\mathcal{B}d^+(bd, \mathcal{L}, \mathcal{C}_{\gamma\text{-}\mathsf{freq}})$ *tel que* $S \subseteq T$ *alors* $freq(S) \geq \gamma$ *et on peut approcher* $\mathsf{freq}(S)$ *par* $(1 + \gamma)/2$. *Sinon,* $freq(S) < \gamma$ *et on peut approcher* $\mathsf{freq}(S)$ *par* $\gamma/2$. *On obtient donc finalement une représentation* ϵ*-adéquate avec* $\epsilon = max(\gamma/2, (1 + \gamma)/2)$. *Donc, dans le meilleur des cas (qui correspond à* $\gamma = 0.5$*),* ϵ *vaut* 0.25 *ce qui est insuffisant pour la majorité des applications.*

Considérons l'extraction de règles d'association à partir $\mathsf{Res}(bd, 2^{\mathsf{Items}}, \mathcal{C}_{\gamma\text{-}\mathsf{freq}})$. La fréquence et la confiance des règles sont connues avec exactitude si la fréquence de la règle est supérieure à γ. Si ce n'est pas le cas, la fréquence d'une règle est bornée par $[0, \gamma]$. De plus, si le membre gauche de la règle est γ-fréquent, sa confiance est bornée par $[0, \gamma/\mathsf{freq}(S)]$. Sinon, sa confiance peut prendre n'importe quelle valeur dans $[0, 1]$, sa meilleure approximation est donc 0.5. L'ensemble $\mathsf{Res}(bd, 2^{\mathsf{Items}}, \mathcal{C}_{\gamma\text{-}\mathsf{freq}})$ est donc une représentation $\gamma/2$-adéquate pour l'évaluation de la fréquence des règles et une représentation 0.5-adéquate pour l'évaluation de leurs confiances. Cette dernière valeur (0.5) est clairement insuffisante pour la plupart des applications. Cependant, si on se restreint maintenant aux seules règles fréquentes, nous avons une représentation 0-adéquate (équivalente) pour l'évaluation de la fréquence et pour celle de la confiance.

On peut aussi s'intéresser aux règles généralisées, comportant par exemple des négations (exemple 11) et vouloir évaluer leurs fréquences et confiances au moyen de $\mathsf{Res}(bd, 2^{\mathsf{Items}}, \mathcal{C}_{\gamma\text{-}\mathsf{freq}})$. Le problème ici est que cette collection ne comportera généralement pas toutes les fréquences indispensables [MT96b, BBJ00].

Exemple 11 *Considérons les calculs de fréquence et de confiance pour la règle* $A \wedge \neg E \Rightarrow D$. *En appliquant des transformations classiques, on peut écrire :*

$$\mathsf{freq}(\mathtt{A} \wedge \neg \mathtt{E} \Rightarrow \mathtt{D}, bd) = \mathsf{freq}(\mathtt{AD}, bd) - \mathsf{freq}(\mathtt{ADE}, bd)$$

$$\mathsf{conf}(\mathtt{A} \wedge \neg \mathtt{E} \Rightarrow \mathtt{D}, bd) = \frac{\mathsf{freq}(\mathtt{AD}, bd) - \mathsf{freq}(\mathtt{ADE}, bd)}{\mathsf{freq}(\mathtt{A}, bd) - \mathsf{freq}(\mathtt{AE}, bd)}$$

Pour être exactes, ces évaluations demandent que A, AD, AE *et* ADE *soient des itemsets* γ*-fréquents dans* bd.

Si l'on veut autoriser plusieurs négations ou encore des disjonctions, le nombre de termes à évaluer augmente considérablement et le besoin en fréquences d'itemsets non fréquents également. Même si l'évaluation de toutes les fréquences est infaisable, on voit que la collection $\mathsf{Res}(bd, 2^{\mathsf{Items}}, \mathcal{C}_{\gamma\text{-}\mathsf{freq}})$ permet d'évaluer les fréquences et les confiances des règles généralisées avec des incertitudes. Borner les erreurs commises dans ce cas reste un problème largement ouvert.

L'intérêt des représentations condensées en ECD est de permettre d'évaluer plus efficacement certaines classes de requêtes. En particulier, nous étudierons en détail des représentations condensées des itemsets fréquents et de leur fréquence dans les chapitres 3 et 4. Les propriétés attendues de ces représentations condensées sont les suivantes.

- Elles doivent être calculables efficacement et être plus compactes (petites) que la collection des itemsets fréquents.
- Les itemsets fréquents et leurs fréquences doivent pouvoir être régénérés efficacement à partir de la représentation condensée.

Il faut en particulier que la somme du temps de calcul de la représentation condensée et de la régénération soit inférieur au temps de calcul direct des itemsets fréquents.

Nous présenterons plusieurs représentations condensées. Certaines sont exactes, c'est-à-dire que les fréquences des itemsets sont connues exactement, et d'autres sont approximatives. Les représentations exactes sont : les itemsets fermés, les libres et les $\vee$-libres. Les itemsets δ-libres sont une représentation approximative relativement précise alors que les itemsets maximaux sont une représentation approximative très peu précise. Différents algorithmes existent pour calculer ces diverses représentations condensées, e.g., les itemsets fermés fréquents (Close [PBTL99c], Closet [PHM00], Charm [ZH02]), les itemsets libres fréquents (Min-Ex [BB00], Pascal [BTP+00]), les itemsets δ-libres fréquents pour Min-Ex ou les itemsets $\vee$-libres pour Lin-Ex [BR01]. Ces algorithmes permettent de calculer les itemsets fréquents sur des données denses et fortement corrélées, i.e., des extractions faisables à des seuils de fréquence pour lesquels les algorithmes classiques, de type Apriori [AMS+96] échouent.

1.5 Questions ouvertes

1.5.1 Les primitives et les solveurs

L'un des problèmes les plus ardus posé par les bases de données inductives est la définition des primitives qui doivent être supportées. Ces primitives doivent être suffisamment riches pour permettre d'exprimer des requêtes potentiellement intéressantes. Les primitives doivent aussi permettre une évaluation efficace des requêtes, c'est-à-dire que la définition des primitives doit se faire conjointement à la conception des solveurs qui vont être utilisés pour calculer les réponses aux requêtes

Il existe pour l'instant de nombreux solveurs dédiés à l'extraction d'un type de motifs en utilisant certaines contraintes. Par exemple, des progrès importants ont été fait dans l'extraction d'itemsets fermés fréquents [PBTL99a, PHM00, Zak00a], l'extraction de séquences fréquentes sous contraintes [GRS99, Zak00b], etc.

Même s'il est possible de percevoir des concepts similaires dans tous ces algorithmes, comme l'élagage à l'aide de contraintes anti-monotones, leur efficacité vient aussi de l'utilisation de techniques très spécifiques à un type de motif ou un type de contrainte. En effet, on n'utilise pas les mêmes structures de données pour représenter des itemsets, des séquences ou des graphes. De même, la manière dont le calcul de la fréquence, le test des contraintes

ou le parcours de l'espace de recherche sont implémentés a une influence énorme sur les performances.

Un premier objectif est donc la recherche de solveurs les plus génériques possibles de manière à élargir le nombre des primitives supportées et ainsi élargir le champ des requêtes possibles. Ce problème est évidemment étroitement lié à la définition des primitives.

Un objectif encore plus ambitieux est la conception d'un système capable d'instancier automatiquement un algorithme générique pour résoudre un problème particulier. Dans ce cas, l'utilisateur serait à même de pouvoir définir des motifs et des contraintes primitives nouvelles que le système analyserait pour proposer un algorithme capable de répondre à des requêtes utilisant ces nouvelles primitives.

Un autre type de problème est posé par la complexité des algorithmes d'extraction. Celle-ci dépend d'une part de la taille de l'espace de recherche à explorer et d'autre part de la taille des données. Dans certains cas, il est tout à fait possible que la taille de l'un ou l'autre rende l'extraction impossible car trop coûteuse en ressources. Si la taille de la base de données est trop importante, nous avons vu que l'utilisation d'un échantillon pour faire une première estimation du résultat puis la correction de celle-ci permet d'améliorer considérablement les performances. Dans le cas où l'espace de recherche est trop grand, l'utilisation des représentations condensées peut permettre de le réduire dans le cas de données fortement corrélées (ce qui correspond justement au cas où le nombre de motifs satisfaisant les contraintes explose).

1.5.2 Vues et matérialisations

Vues

On peut imaginer, de la même manière que pour les bases de données relationnelles classiques, définir des collections de motifs par des vues. Dans ce cas, la collection de motifs dépend d'une ou plusieurs tables de la partie base de données et/ou de collections de motifs déjà définies. La différence avec une requête classique est la même que dans une base de données relationnelle. Toute modification de l'une des tables ou de l'une des collections de motifs, à partir desquelles la vue est définie, entraîne une modification de celle-ci. Par exemple, le fait de rajouter des tuples dans une table modifie la fréquence des motifs définis sur cette table. Certains motifs qui étaient fréquents peuvent devenir infréquents et inversement. Cela pose évidemment des problèmes liés à la matérialisation de ces vues. Si une vue est matérialisée, il faut propager toute modification des tables sous-jacentes.

Il faut pour cela sans doute s'inspirer des techniques utilisées pour gérer les vues dans les bases de données relationnelles. Cependant, il faut tenir compte du fait que les collections de motifs sont généralement bien plus coûteuses à créer et à mettre à jour que les tables relationnelles : l'ajout où la modification d'un faible nombre de tuples dans une table peut très bien modifier une grande partie des motifs calculés sur cette table (en particulier, cela modifie la fréquence de nombreux motifs).

Matérialisation

Le coût de calcul d'une collection de motifs est souvent beaucoup plus élevé que celui d'une table relationnelle pour une base de données classique. Il faut donc éviter à tout prix de matérialiser le résultat d'une requête inductive lorsqu'il n'est pas explicitement demandé par l'utilisateur.

Exemple 12 *Supposons que l'utilisateur demande le calcul des règles valides avec un seuil de fréquence de* 0.5 *et un seuil de confiance de* 0.7 *dans une première requête* σ_1*. Notons* s_1 *l'ensemble de ces règles. Dans une deuxième requête* σ_2*, l'utilisateur demande les règles de* s_1 *qui contiennent l'attribut* `A` *dans le corps de la règle. Dans ce cas, il suffit de faire une sélection des règles qui conviennent dans* s_1*.*

Si maintenant nous supposons que l'extraction de s_1 *ne soit pas faisable en un temps raisonnable et que l'utilisateur ne demande pas le calcul explicite du résultat, dans ce cas, il est sans doute préférable de ne pas matérialiser cette collection. Lorsque l'utilisateur définit sa deuxième requête, cette dernière étant plus restrictive, il sera éventuellement possible (en utilisant des techniques d'optimisation que nous verrons dans le chapitre 2) de calculer le résultat. Si maintenant l'utilisateur définit une nouvelle requête* σ_3 *où il demande les règles qui contiennent soit l'attribut* `A` *soit l'attribut* `B` *dans le corps de la règle, alors il est intéressant de pouvoir utiliser les résultats de la requête* σ_2 *pour répondre à la requête* σ_3*, par exemple en utilisant un cache (voir le chapitre 4) et en évitant de matérialiser le résultat de* σ_1*.*

Dans le cadre des bases de données inductives, on voit à travers cet exemple qu'il n'y a pas de différence fondamentale entre une requête de sélection sur des motifs déjà calculés et une requête qui demande un nouveau calcul. Toutes les deux sont définies comme des requêtes de sélection sur la collection des motifs. C'est à un éventuel système de gestion de la base de données inductive de s'apercevoir que pour calculer le résultat d'une nouvelle requête, il peut se baser sur le résultat d'une requête précédente si elle a été au moins partiellement matérialisée.

1.5.3 Optimisation de séquences de requêtes

Nous avons vu que le processus d'ECD est par nature interactif et itératif. L'utilisateur ne se contente pas d'une seule requête. En effet, il est peu probable que le résultat de sa première requête soit satisfaisant, il utilisera donc plusieurs requêtes. Pour répondre à une de ces requêtes, le système peut utiliser une technique "générer et tester". Si, par exemple, la requête porte sur les itemsets, le système extrait les itemsets fréquents et supprime ensuite les itemsets qui ne vérifient pas la contrainte de l'utilisateur. Cette technique est souvent beaucoup plus coûteuse que celle qui consiste à utiliser activement les contraintes pendant l'extraction. Seulement, même si les contraintes sont poussées, le calcul de la réponse à une requête reste souvent trop important pour que le processus soit réellement interactif.

De plus, lorsqu'on utilise une stratégie "générer et tester", l'étape d'extraction de la collection des itemsets fréquents peut être amortie si on utilise plusieurs fois cette collection pour répondre à une séquence de requêtes. En fait, si on considère que cette collection est disponible, l'étape de filtrage qui consiste à ne garder que les itemsets qui vérifient la contrainte de l'utilisateur est extrêmement rapide, même par rapport à une extraction utilisant activement les contraintes. Finalement, la stratégie "générer et tester" peut être plus efficace que de pousser les contraintes quand on considère une séquence de plusieurs requêtes.

On peut imaginer que les itemsets fréquents sont extraits au seuil le plus bas possible compte tenu de limitations en temps et en mémoire à un moment où l'utilisateur n'est pas présent. Ensuite, le système utilisera cette collection matérialisée pour répondre aux requêtes de l'utilisateur. Il pourra alors le faire très efficacement. Le principal problème de cette approche est que l'utilisateur est limité par la collection matérialisée : il est possible que le seuil de fréquence utilisé pour générer la collection de motifs soit trop élevé pour contenir tous les motifs intéressants. Dans ce cas, il est possible que ces motifs soient accessibles en poussant

les contraintes. Finalement, aucune de ces stratégies n'est satisfaisante dans tous les cas.

1.6 Introduction de la suite du mémoire par rapport à ces problèmes

Nous présentons dans le chapitre suivant un algorithme générique pour traiter des conjonctions de contraintes monotones et anti-monotones, c'est-à-dire une large classe de contraintes. Cet algorithme nous permettra dans le chapitre 3 de combiner le gain apporté par l'utilisation des représentations condensées et des contraintes monotones et anti-monotones. En particulier, nous montrerons que l'utilisation des représentations condensées permet de conduire des extractions dans des données fortement corrélées où l'utilisation des contraintes seules n'est pas praticable. Ainsi, nous allons dans le sens d'un algorithme générique pour traiter une classe de contrainte la plus large possible tout en rendant possible les extractions dans des données mêmes fortement corrélées.

Dans le chapitre 4, nous montrons comment optimiser une séquence de requêtes. Pour cela nous utiliserons une technique de cache similaire à un cache web. Un cache web conserve les pages qui ont été consultées par les utilisateurs. Ainsi, lors de la consultation suivante d'une page stockée dans le cache, celle-ci est immédiatement disponible et on peut éviter un téléchargement coûteux. La même idée peut être utilisée pour les motifs. Dans ce cas la phase coûteuse est le calcul de la fonction d'évaluation sur les données. Notre cache contiendra donc des motifs et la valeur de la fonction d'évaluation pour ceux-ci. Ainsi, lors du calcul de la réponse à une nouvelle requête, il ne sera pas nécessaire de calculer la fonction d'évaluation sur les motifs qui sont dans le cache. On combine donc ainsi la rapidité de l'extraction de motifs dans une collection déjà matérialisée à la flexibilité d'un algorithme capable d'utiliser efficacement les contraintes pour accélérer l'extraction.

Notre étude portera plus particulièrement sur l'utilisation des représentations condensées comme caches. Les deux principaux intérêts sont, d'une part, qu'elles permettent des extractions plus rapides dans le cas de données fortement corrélées et, d'autre part, que le cache reste de taille très réduite par rapport à un cache qui contiendrait tous les motifs.

Chapitre 2

Extraction sous contraintes

Dans ce chapitre nous allons voir comment utiliser efficacement les contraintes spécifiées par l'utilisateur dans sa requête inductive afin d'en optimiser l'évaluation. Les travaux exposés dans ce chapitre ont été publiés en partie dans [BJ00].

Le premier algorithme à rentrer dans ce cadre est l'algorithme Apriori où la contrainte de fréquence minimale est utilisée pour élaguer le treillis des itemsets. Cet algorithme est présenté dans la première section avec une généralisation naturelle pour prendre en compte une contrainte anti-monotone quelconque.

Dans la section 2, nous rappelons le principe de certains algorithmes qui calculent directement la frontière de la collection des motifs intéressants. Dans la section 3, nous introduisons le concept de contrainte monotone puis nous faisons un état de l'art sur l'extraction de motifs sous contraintes dans la section 4. Dans la section 5, nous présentons un algorithme générique pour extraire les itemsets sous contraintes monotones et anti-monotones qui généralise ceux vus dans la section 4. Finalement, la section 6 compare différentes stratégies que cet algorithme permet d'utiliser pour l'évaluation de requêtes inductives.

2.1 Le cas particulier de la contrainte de fréquence minimale

Le problème de la recherche d'itemsets fréquents et de règles d'association a été présenté la première fois dans [AIS93]. Le premier algorithme efficace que nous allons présenter ici, l'algorithme Apriori, a été décrit dans [AMS+96]. Cet algorithme extrait tous les itemsets γ-fréquents dans une base de données binaire. Le seuil de fréquence γ est un paramètre fourni par l'utilisateur. En utilisant le formalisme des requêtes inductives que nous avons introduit dans le chapitre précédent, l'algorithme Apriori calcule le résultat $\mathsf{Res}(\sigma)$ de la requête inductive étendue $\sigma = (bd, 2^{\mathsf{Items}}, \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq}, [0,1])$.

Dans l'algorithme ci-dessous, chaque itemset S est stocké dans un enregistrement (également noté S ici) composé de deux champs. Un champ $S.\mathsf{item}$ qui contient la liste des éléments ou items qui constituent l'itemset et un champ $S.\mathsf{freq}$ qui contient la fréquence de S une fois qu'elle a été calculée. L'enregistrement sera noté $S = (S.\mathsf{item}, S.\mathsf{freq})$. Par extension, si, par exemple, $\mathsf{Cand} = \{(\mathtt{A}, 0.3), (\mathtt{BC}, 0.2)\}$ est un ensemble d'enregistrements, on notera $\mathsf{Cand.item} = \{\mathtt{A}, \mathtt{BC}\}$.

Algorithme 2 (Apriori)

Entrée : Une base de données binaire bd, un seuil de fréquence γ
Sortie : $\mathsf{Res}(bd, 2^{\mathsf{Items}}, \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq}, [0,1])$

```
1   k := 1
2   Cand := {(I, 0) | I ∈ Items}
3   tant que Cand ≠ ∅ faire
4       Cand := passe_bd(Cand, bd)          /* calcul de S.freq pour tout S ∈ Cand */
5       Cand := {S ∈ Cand | S.freq ≥ γ}     /* les candidats infréquents sont éliminés */
6       S_k := Cand
7       Cand := apriori_gen(S_k)            /* génération des candidats du niveau suivant */
8       k := k + 1
9   fait
10  retourne ⋃_{1≤i≤k} S_i ∪ {(∅, |bd|)}
```

Cet algorithme effectue un parcours niveau par niveau du treillis des itemsets. Lors de la première itération, les itemsets fréquents de taille 1 sont calculés, puis lors de la deuxième itération ceux de taille 2 sont calculés, puis ceux de taille 3, 4, etc. jusqu'à ce qu'il n'y ait plus aucun candidat. À ce moment, l'algorithme se termine et retourne l'ensemble des itemsets fréquents et leur fréquence.

Examinons chaque étape plus en détail. Au début de chaque itération, l'ensemble Cand contient les itemsets candidats de taille k, c'est-à-dire ceux qui sont potentiellement fréquents. Lors de la première étape de chaque itération (ligne 4), la fréquence des itemsets candidats est calculée lors d'une passe sur la base de données (appel à la fonction **passe_bd** que nous détaillons plus loin). Ensuite (ligne 5), les itemsets qui ne sont pas fréquents sont éliminés et ceux qui sont fréquents sont stockés dans $\mathcal{S}_k$ (ligne 6). À la ligne 7, les candidats de taille $k+1$ sont générés à partir des itemsets fréquents de taille k par la fonction **apriori_gen**.

apriori_gen($\mathcal{S}$)
/* génération des candidats potentiels */
$C := \{S[1]S[2]\dots S[k-1]T[k-1] \mid S, T \in \mathcal{S}.\mathsf{item}$ et $S[1] = T[1], \dots, S[k-2] = T[k-2]$ et $T[k-1] \lhd S[k-1]\}$
/* élagage */
pour tout $S \in C$ **faire**
  pour tout $T \subset S$, $|T| = |S| - 1$ **faire**
    si $T \notin \mathcal{S}.\mathsf{item}$ **alors**
      $C := C \setminus \{S\}$ **finsi**
  fait
fait
retourne $\{(S, 0) \mid S \in C\}$

La fonction **apriori_gen** procède en deux étapes. Une étape de génération et une étape d'élagage. Pour l'étape de génération, on suppose qu'il existe un ordre $\lhd$ sur les items. Si S est un item, on note $S[i]$ le $i^{\text{ème}}$ plus grand item (par rapport à $\lhd$) contenu dans $S.\mathsf{item}$. Lors de la première étape, l'ensemble C est construit. Il contient l'ensemble des itemsets obtenus en joignant deux itemsets de $\mathcal{S}_k.\mathsf{item}$ qui les ont mêmes $k-1$ premiers items (lorsqu'on considère les items classés par l'ordre $\lhd$). La fonction renvoie ensuite les itemsets S tels que tout sous

ensemble T de S de taille $k-1$ soit dans $\mathcal{S}.\mathsf{item}$. Par exemple, si l'ordre est $\mathtt{A} \rhd \mathtt{B} \rhd \mathtt{C} \rhd \ldots$ et si $\mathcal{S}.\mathsf{item} = \{\mathtt{AB}, \mathtt{AC}, \mathtt{BC}, \mathtt{BD}\}$, alors C contiendra `ABC` (généré en joignant `AB` et `AC`) et `BCD` (généré en joignant `BC` et `BD`). Ensuite `BCD` sera éliminé car `CD` n'est pas dans $\mathcal{S}.\mathsf{item}$ et la fonction renverra $\{(\mathtt{ABC}, 0)\}$.

Revenons à la procédure **passe_bd** qui calcule la fréquence des itemsets candidats en faisant une passe sur les données. Pour chaque transaction de la base de données, les compteurs de fréquence de tous les itemsets qui sont inclus dans cette transaction sont incrémentés.

passe_bd(Cand, bd)
  pour toute transaction $t \in bd$ **faire**
    pour tout itemset candidat $S \in \mathsf{Cand}$ **faire**
      si $S.\mathsf{item} \subseteq t$ **alors** $S.\mathsf{freq} := S.\mathsf{freq} + 1$
    fait
  fait

$bd_1 =$

tid	
1	ABCDE
2	ABCD
3	ACD
4	ABE
5	CD
6	CE

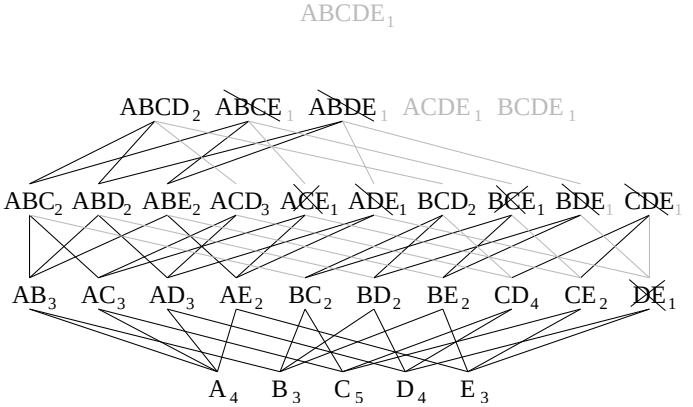


FIG. 2.1 – Exemple d'exécution d'Apriori avec un seuil de fréquence absolue de 2, les fréquences des itemsets sont données en indice. L'exploration du treillis est faite niveau par niveau en commençant par les itemsets de taille 1. Les candidats du niveau suivant sont générés en fusionnant deux itemsets fréquents qui ont le même préfixe, ces fusions sont indiquées par les traits noirs. Ensuite, l'algorithme vérifie que tous les sous ensembles des itemsets candidats sont bien fréquents. Si un itemset a un sous ensemble infréquent, il est éliminé (ce sont les itemsets barrés). Ensuite, l'algorithme calcule les fréquences des itemsets restants et élimine ceux qui ne sont pas fréquents (barrés deux fois). L'algorithme s'arrête lorsqu'il n'y a plus de candidat. Sur la figure, les itemsets fréquents sont donc les itemsets en noirs qui ne sont pas barrés (les itemsets en gris ne sont jamais générés et ne peuvent donc pas être fréquents).

L'algorithme Apriori exploite l'anti-monotonicité de la contrainte de fréquence minimale pour ne parcourir qu'une petite partie de ce treillis. En fait, tout est basé sur la fonction de génération des candidats. En effet, c'est cette fonction qui détermine la partie du treillis qui va effectivement être explorée. Cette fonction exploite l'anti-monotonicité dans les deux étapes de la génération. Premièrement, l'étape de jointure des itemsets fréquents du niveau précédent assure que les candidats ont au moins deux sous ensembles fréquents. On exploite le fait que tout itemset fréquent a tous ses sous ensembles fréquents, ainsi cette méthode de génération assure que l'on ne manque aucun itemset fréquent (l'algorithme Apriori est bien complet). Ensuite, la deuxième étape élimine des candidats qui ne peuvent pas être fréquents du fait qu'ils ont un sous ensemble infréquent. On remarque que dans cette deuxième étape,

on ne considère que les sous ensembles de taille $k-1$ des itemsets candidats de taille k. En effet, il est facile de voir que si un itemset possède un sous ensemble infréquent, alors il possède nécessairement un sous ensemble infréquent de taille $k-1$.

En fait, l'algorithme Apriori est une instanciation de l'algorithme Guess and correct (p. 34) lorsque la collection initiale est vide. En particulier, il est facile de se rendre compte que la fonction de génération des candidats retourne exactement la partie de la frontière négative de $\bigcup_{1\leq i\leq k}\mathcal{S}_i$ qui n'a pas encore été explorée (cela correspond à la ligne 12 de l'algorithme Guess and correct). Pour faire cette génération, l'algorithme utilise deux étapes, une étape de génération et une étape d'élagage. Cette méthode assure que la fréquence ne sera calculée que pour les itemsets qui satisfont $\mathcal{C}_{\gamma\text{-freq}}$ ou ceux qui sont dans la frontière négative.

Exemple 13 *La figure 2.1 page précédente présente une extraction des itemsets de fréquence absolue supérieure à 2 dans la base de données bd_1 de la figure 1.1. Sur cet exemple,*

$$\mathcal{B}d^+(\mathsf{Th}(bd_1, 2^{\mathsf{Items}}, \mathcal{C}_{2\text{-freq}})) = \{\mathtt{ABCD}, \mathtt{ABE}, \mathtt{CE}\}\,,$$
$$\mathcal{B}d^-(\mathsf{Th}(bd_1, 2^{\mathsf{Items}}, \mathcal{C}_{2\text{-freq}})) = \{\mathtt{DE}, \mathtt{ACE}, \mathtt{BCE}\}\,.$$

Apriori *calcule la fréquence des itemsets de* $\mathsf{Th}(bd_1, 2^{\mathsf{Items}}, \mathcal{C}_{2\text{-freq}}) \cup \mathcal{B}d^-(\mathsf{Th}(bd_1, 2^{\mathsf{Items}}, \mathcal{C}_{2\text{-freq}}))$.

L'algorithme Apriori présente un intérêt car il montre un exemple d'instanciation d'un algorithme générique et les problèmes pour en implémenter efficacement les différentes étapes. En particulier, la première étape de la fonction de génération des candidats est très importante. On pourrait imaginer une fonction de génération dont la première étape consisterait à générer tous les itemsets de taille k possibles. Cette fonction de génération serait complètement inexploitable. En effet, dès que k devient grand, le nombre d'itemsets de taille k explose. Par exemple, si on considère qu'il y a 1000 items différents alors il y a $\binom{1000}{5} \approx 8\cdot 10^{12}$ itemsets au niveau 5. Il sera donc impossible de tous les générer pour ensuite vérifier si leurs sous ensembles de taille 4 sont dans $\mathcal{S}_4$. Le fait que la fonction de génération des candidats crée de nouveaux candidats à partir des itemsets fréquents du niveau précédent joue un rôle important dans son efficacité en limitant le nombre de candidats. Nous verrons d'ailleurs que lorsqu'on essaye de pousser d'autres contraintes non anti-monotones, trouver une fonction de génération des candidats efficace pose problème.

2.1.1 Généralisation au cas d'une contrainte anti-monotone quelconque

On peut en fait utiliser Apriori pour calculer le résultat $\mathsf{Res}(bd, 2^{\mathsf{Items}}, \mathcal{C}_{\mathsf{am}}, \mathsf{freq}, [0,1])$ d'une requête inductive étendue utilisant une contrainte anti-monotone $\mathcal{C}_{\mathsf{am}}$ quelconque. Il suffit de modifier la ligne 5 de l'algorithme Apriori.

Algorithme 3 (Apriori pour une contrainte anti-monotone $\mathcal{C}_{\mathsf{am}}$)

Entrée : Une base de données binaire bd, une contrainte anti-monotone $\mathcal{C}_{\mathsf{am}}$
Sortie : $\mathsf{Res}(bd, 2^{\mathsf{Items}}, \mathcal{C}_{\mathsf{am}}, \mathsf{freq}, [0,1])$

1 $k := 1$
2 Cand $:= \{(I, 0) \mid I \in Items\}$
3 **tant que** Cand $\neq \emptyset$ **faire**
4 Cand $:=$ passe_bd(Cand, bd) /* calcul de S.freq pour tout $S \in$ Cand */

5 Cand := $\{S \in \mathsf{Cand} \mid \mathcal{C}_{\mathsf{am}}(S.\mathsf{item}) = vrai\}$ /* les candidats qui ne satisfont pas $\mathcal{C}_{\mathsf{am}}$
6 $\mathcal{S}_k$:= Cand sont éliminés */
7 Cand := apriori_gen($\mathcal{S}_k$) /* génération des candidats du niveau suivant */
8 $k := k + 1$
9 **fait**
10 **retourne** $\bigcup_{1 \leq i \leq k} \mathcal{S}_i \cup \{(\emptyset, |bd|)\}$

Dans cet algorithme, on utilise la contrainte anti-monotone après la passe sur les données. Cela est nécessaire si elle utilise la fréquence calculée pendant la passe sur les données. Sinon, nous verrons dans la suite qu'il vaut mieux la tester avant la passe sur les données et même avant la phase d'élagage de la génération des candidats. Par exemple, si la contrainte est $\mathcal{C}_{\mathsf{am}}(S) = \mathcal{C}_{\gamma\text{-}\mathsf{freq}}(S) \wedge (\mathtt{A} \notin S)$ alors la contrainte $\mathcal{C}_{\gamma\text{-}\mathsf{freq}}$ doit être testée à la ligne 5, par contre la contrainte $(\mathtt{A} \notin S)$ peut être testée avant car il n'est pas nécessaire de connaître la fréquence de S pour la tester. Ainsi on peut éviter de calculer la fréquence des itemsets qui ne satisfont pas cette dernière contrainte. Cette remarque mise à part, cet algorithme utilise les mêmes fonctions **passe_bd** et **apriori_gen** que l'algorithme 2. Il est toujours complet car la fonction de génération des candidats d'Apriori n'est basée que sur l'anti-monotonicité de la contrainte.

2.1.2 Complexité d'Apriori

Considérons une base de données binaire de m lignes sur un ensemble d'attributs Items de n éléments. Au niveau k, le coût du calcul de la fréquence des itemsets est, dans le pire cas, $O(m\,|\mathsf{Cand}|)$ (le coût réel n'est pas beaucoup plus faible en pratique). Le coût du test de la fréquence est en $O(|\mathsf{Cand}|)$ et le coût de la génération des candidats du niveau $k+1$ est en $O(k\,|\mathsf{Cand}|)$. Finalement, si la base de données est grande (i.e., m est grand), c'est le coût du calcul de la fréquence qui domine et la complexité totale de l'algorithme est $O(ms)$ où s est le nombre total de candidats, i.e., $s = \left|\mathsf{Th}(bd, 2^{\mathsf{Items}}, \mathcal{C}_{\mathsf{am}}) \cup \mathcal{B}d^-(\mathsf{Th}(bd, 2^{\mathsf{Items}}, \mathcal{C}_{\mathsf{am}}))\right|$. Dans le pire des cas, la théorie contient tous les itemsets possibles, $s = 2^n$ et donc la complexité est $O(m2^n)$. En pratique, si le seuil de fréquence n'est pas trop petit et la base de données peu dense, la complexité sera beaucoup plus faible, ce qui rendra l'algorithme utilisable même lorsque la base de données contient beaucoup de lignes (jusqu'à plusieurs millions) et beaucoup de colonnes (jusqu'à plusieurs milliers).

Dans ce cas, Apriori supporte bien le passage à l'échelle. À seuil de fréquence γ constant, le temps de calcul est linéaire par rapport au nombre de lignes de la base de données. La dépendance entre le nombre d'itemsets fréquents et le seuil de fréquence γ dépend de la base de données. En pratique, ce nombre est souvent exponentiel par rapport à $(1 - \gamma)$, le temps de calcul l'est donc également.

2.2 Calcul direct de la frontière

Pour calculer tous les itemsets qui satisfont une contrainte anti-monotone $\mathcal{C}_{\mathsf{am}}$, nous avons vu avec la propriété 3 page 33 qu'il suffit de calculer la frontière positive $\mathcal{B}d^+(\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{am}}))$. Nous allons voir dans cette section des algorithmes qui calculent cette frontière sans avoir à parcourir tous les éléments situés en dessous comme le fait Apriori.

Ces algorithmes calculent en fait les itemsets fréquents maximaux, i.e.,

$$\mathcal{B}d^{+}(\mathsf{Th}(bd, 2^{\mathsf{Items}}, \mathcal{C}_{\gamma\text{-}\mathsf{freq}})),$$

mais il est facile de les modifier pour qu'ils calculent $\mathcal{B}d^{+}(\mathsf{Th}(bd, 2^{\mathsf{Items}}, \mathcal{C}_{\mathsf{am}}))$.

Les algorithmes Max-clique et Max-eclat [ZPOL97], Max-miner [Bay98], Pincer-search [LK02] et DepthProject [AAP00] ne sont pas basés sur une exploration niveau par niveau. Ils tentent d'arriver plus vite à la frontière en n'explorant pas de manière exhaustive tous les niveaux du treillis. Ils sont donc particulièrement efficaces lorsqu'il existe de très longs itemsets car ils n'explorent pas tous leurs sous ensembles. Ce sont des algorithmes spécifiques pour rechercher des itemsets fréquents. On peut citer aussi l'algorithme Molfea [dRK01] qui lui est basé sur l'algorithme des espaces des versions de Mitchell [Mit80].

Les algorithmes les plus anciens, Max-clique et Max-eclat, calculent les paires fréquentes (itemsets de taille 2) et construisent ensuite les itemsets maximaux potentiels à partir des paires. Les sous ensembles de ces maximaux potentiels sont alors explorés soit des plus généraux vers les plus spécifiques soit d'une manière hybride, à la fois descendante et ascendante. D'après [Bay98], ces algorithmes ne diminuent cependant pas suffisamment le nombre d'itemsets considérés par rapport à Apriori pour être réellement efficaces avec des données très corrélées, i.e., avec de très grands itemsets fréquents.

Les algorithmes Max-miner et Pincer-search utilisent une exploration du treillis niveau par niveau comme Apriori. Cependant, ils génèrent aussi, pendant cette exploration niveau par niveau, des itemsets de taille plus grande que les itemsets du niveau courant (la principale différence entre ces algorithmes est la manière dont ces itemsets sont construits). Si l'un de ces itemsets s'avère fréquent, alors il n'est pas nécessaire d'explorer ses sous ensembles. Par exemple, si lors de l'exploration du niveau 2 l'algorithme découvre que l'itemset `ABCD` est fréquent, alors, lors de l'exploration du niveau 3, il ne sera pas nécessaire de considérer les itemsets `ABC`, `ABD`, `ACD` et `BCD` car il sont forcément fréquents. Plus l'itemset fréquent découvert est grand et plus il est découvert tôt, plus il est intéressant. En effet, un itemset fréquent de taille n découvert pendant l'exploration du niveau k permet d'éviter de considérer tous ses sous ensembles de taille comprise entre k et n. On observe qu'en pratique ces "sauts" sont très efficaces. Ces algorithmes sont notablement plus rapides qu'Apriori.

L'algorithme DepthProject utilise, quant à lui, une exploration du treillis en profondeur d'abord ce qui lui permet de découvrir rapidement des itemsets de grande taille sans avoir à considérer tous leurs sous ensembles.

Cependant, il faut remarquer que tous ces algorithmes ne résolvent pas le même problème que Apriori. Ils fournissent en effet les itemsets fréquents maximaux qui caractérisent tous les itemsets fréquents, mais ils ne permettent pas de connaître la fréquence de tous les itemsets fréquents. Par conséquent, ces algorithmes ne calculent pas le résultat d'une requête inductive étendue mais seulement sa théorie. On pourrait imaginer de les utiliser pour calculer la frontière, puis ensuite de calculer la fréquence de tous les itemsets sous la frontière, mais cette stratégie n'est pas efficace. En effet, il a été montré [BTP+00] que l'étape de calcul des fréquences est à elle seule pratiquement aussi coûteuse que d'utiliser l'algorithme Apriori.

Il existe des applications où l'on s'intéresse surtout aux longs itemsets ou bien pour lesquelles l'utilisation d'Apriori n'est pas possible car les itemsets fréquents sont trop longs. Dans ces cas, ces algorithmes sont particulièrement intéressants. En effet, il n'est pas nécessaire d'explorer tous les sous ensembles des itemsets fréquents maximaux. Ceci est justement un des inconvénients majeurs d'Apriori. Si un itemset de taille n est fréquent, Apriori devra pour le trouver considérer d'abord ses 2^n sous ensembles, ce qui le rend inutilisable en pratique dès qu'il y a des itemsets fréquents de taille 20 ou plus.

2.3 Contraintes monotones

Dans cette section nous présentons les contraintes monotones qui seront utilisées dans le reste du chapitre et dans les chapitres suivants.

Définition 16 (contrainte monotone) *Une contrainte $\mathcal{C}$ est* monotone *si*

$$\forall bd\, \forall \varphi\, \forall \mu\, (\varphi \preceq \mu \wedge \mathcal{C}(\varphi, bd)) \Rightarrow \mathcal{C}(\mu, bd).$$

Exemple 14 *Les contraintes $\mathcal{C}_1(S) = (\mathtt{A} \in S)$, $\mathcal{C}_2(S) = (S \cap \mathtt{BCD} \neq \emptyset)$ et $\mathcal{C}_3(S) = |S| \geq 3$ sont des contraintes monotones.*

On remarque que les contraintes monotones sont les négations des contraintes anti-monotones et réciproquement.

Proposition 4 *La négation d'une contrainte anti-monotone est monotone et la négation d'une contrainte monotone est anti-monotone*

> **Preuve :** Si $\mathcal{C}_{\mathsf{am}}$ est anti-monotone alors, d'après la définition, si $S \subseteq T$ alors $\mathcal{C}_{\mathsf{am}}(T) \Rightarrow \mathcal{C}_{\mathsf{am}}(S)$. Ceci est équivalent à, si $S \subseteq T$ alors $\neg\mathcal{C}_{\mathsf{am}}(S) \Rightarrow \neg\mathcal{C}_{\mathsf{am}}(T)$ et on retrouve la définition de la monotonicité pour $\neg\mathcal{C}_{\mathsf{am}}$. La réciproque se fait de la même manière. □

Cette propriété permet de transposer les propriétés de composition des contraintes anti-monotones.

Proposition 5 *Soit $\mathcal{C}_{\mathsf{m}_1}$ et $\mathcal{C}_{\mathsf{m}_2}$ deux contraintes monotones, alors la conjonction $\mathcal{C}_{\mathsf{m}_1} \wedge \mathcal{C}_{\mathsf{m}_2}$ et la disjonction $\mathcal{C}_{\mathsf{m}_1} \vee \mathcal{C}_{\mathsf{m}_2}$ de ces deux contraintes sont aussi des contraintes monotones.*

> **Preuve :** Comme les contraintes $\mathcal{C}_{\mathsf{m}_1}$ et $\mathcal{C}_{\mathsf{m}_2}$ sont monotones alors (proposition 4) les contraintes $\neg\mathcal{C}_{\mathsf{m}_1}$ et $\neg\mathcal{C}_{\mathsf{m}_2}$ sont anti-monotones. Donc, d'après la proposition 2 page 32, les contraintes $\neg\mathcal{C}_{\mathsf{m}_1} \wedge \neg\mathcal{C}_{\mathsf{m}_2}$ et $\neg\mathcal{C}_{\mathsf{m}_1} \vee \neg\mathcal{C}_{\mathsf{m}_2}$ sont anti-monotones. En réutilisant la proposition 4, les contraintes $\neg(\neg\mathcal{C}_{\mathsf{m}_1} \wedge \neg\mathcal{C}_{\mathsf{m}_2}) = \mathcal{C}_{\mathsf{m}_1} \vee \mathcal{C}_{\mathsf{m}_2}$ et $\neg(\neg\mathcal{C}_{\mathsf{m}_1} \vee \neg\mathcal{C}_{\mathsf{m}_2}) = \mathcal{C}_{\mathsf{m}_1} \wedge \mathcal{C}_{\mathsf{m}_2}$ sont monotones. □

Comme pour les contraintes anti-monotones, on peut aussi caractériser les motifs satisfaisant une contrainte monotone par une frontière. En effet, les motifs satisfaisant une contrainte monotone $\mathcal{C}_{\mathsf{m}}$ sont exactement les motifs qui ne satisfont pas la contrainte anti-monotone $\neg\mathcal{C}_{\mathsf{m}}$. On a donc une proposition duale de la proposition 3 page 33.

Proposition 6 *La collection de motifs $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{m}})$ est exactement la collection des motifs qui sont plus spécifiques qu'un motif de $\mathcal{B}d^-(\mathsf{Th}(bd, \mathcal{L}, \neg\mathcal{C}_{\mathsf{m}}))$,*

$$\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{m}}) = \left\{\varphi \in \mathcal{L} \mid \exists \mu \in \mathcal{B}d^-(\mathsf{Th}(bd, \mathcal{L}, \neg\mathcal{C}_{\mathsf{m}})), \mu \preceq \varphi\right\}.$$

De manière duale, la collection des motifs qui ne sont pas dans $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{m}})$ est exactement la collection des motifs qui sont plus généraux qu'un motif de $\mathcal{B}d^+(\mathsf{Th}(bd, \mathcal{L}, \neg\mathcal{C}_{\mathsf{m}}))$.

$$\mathcal{L} \setminus \mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_{\mathsf{m}}) = \left\{\varphi \in \mathcal{L} \mid \exists \mu \in \mathcal{B}d^+(\mathsf{Th}(bd, \mathcal{L}, \neg\mathcal{C}_{\mathsf{m}})), \varphi \preceq \mu\right\}.$$

2.4 État de l'art sur l'utilisation des contraintes

Nous présentons ici un état de l'art de l'extraction de motifs sous contraintes. Cet état de l'art ne se veut exhaustif que pour le cas des itemsets et des règles d'association. Le nombre de travaux publiés ces dernières années sur le sujet est en effet impressionnant, chacune des nombreuses conférences majeures (SIGKDD, PKDD, PAKDD, DAWAK, CIKM, SIGMOD, VLDB, ICDE, ...) consacre généralement plusieurs articles sur le sujet.

Les recherches se sont orientées tout d'abord vers l'extraction des règles d'association, puis d'autres types de motifs ont été étudiés. En général, ces études ont d'abord traité de contraintes assez simples comme la contrainte de fréquence minimale et des contraintes spécifiques à certains motifs (comme la confiance minimale pour les règles). Ensuite, de nouveaux algorithmes ont été développés qui utilisent des contraintes de plus en plus variées. L'objectif de ces travaux est d'essayer de tirer parti des contraintes pour réduire la taille de l'espace de recherche, autrement dit de pousser les contraintes. Les motivations de ces travaux sont doubles :

- d'une part l'optimisation ;
- d'autre part l'amélioration de la capacité de l'utilisateur à spécifier précisément ce qui l'intéresse.

On se trouve donc en général de plus en plus dans une approche privilégiant la capacité de l'utilisateur à interagir avec le système. Tous ces travaux s'intègrent donc très naturellement dans un cadre base de données inductive même si cela n'est en général pas revendiqué par les auteurs.

Cette section est organisée de la manière suivante. D'abord, nous présentons rapidement le principe général des algorithmes d'extraction de motifs sous contraintes. Ensuite, nous présentons différents travaux en les regroupant par type de motifs extraits (itemsets, séquences, etc.) et en précisant à chaque fois les types des contraintes qui sont "poussées" pendant l'exploration de l'espace de recherche.

Ensuite, dans la section suivante, nous faisons une synthèse des techniques utilisées par ces algorithmes pour traiter deux étapes importantes de l'extraction par rapport à notre problématique : la génération des candidats et l'élagage.

Le principe des algorithmes que nous allons présenter est basé sur un parcours de l'espace de recherche niveau par niveau. Ce parcours permet de profiter au mieux de l'élagage apporté par les contraintes anti-monotones. Nous présentons ci-dessous le principe de ces algorithmes.

```
1  Génération des candidats du premier niveau
2  tant que critère_d_arrêt = faux faire
3      Calcul des fonctions d'évaluation (passe sur les données)
4      Test des contraintes non syntaxiques sur les candidats
5      Sortie du résultat
6      Génération des candidats du niveau suivant
7      Élagage des nouveaux candidats
8      Test des contraintes syntaxiques sur les nouveaux candidats
```

On remarque qu'ici on a séparé les deux étapes de la génération des candidats, à savoir la génération proprement dite et l'étape d'élagage. Ces deux étapes étaient faites dans la même fonction lorsque nous avons présenté l'algorithme Apriori dans la section 2.1 mais il s'agit conceptuellement de deux phases distinctes.

2.4.1 Extraction d'itemsets

Algorithmes MultipleJoins, Reorder et Direct

En 1997, Srikant et al. [SVA97] sont les premiers à présenter un algorithme d'extraction d'itemsets utilisant des contraintes autre que la contrainte de fréquence minimale. Cet article propose des algorithmes d'extraction d'itemsets fréquents satisfaisant une contrainte syntaxique. Les contraintes syntaxiques primitives considérées sont de la forme ($i \in \mathsf{Items}$) :

$$\mathcal{C}_i(S) = i \in S,$$
$$\mathcal{C}_{\neg i}(S) = i \notin S.$$

Les auteurs introduisent d'autres contraintes dans le cas où il existe une taxonomie sur les items. Une taxonomie est une relation r acyclique (il n'existe pas de cycle $r(i_1, i_2)$, $r(i_2, i_3)$, ..., $r(i_n, i_1)$) sur les items. Cette relation est souvent représentée par son graphe orienté acyclique. Les taxonomies servent à classifier les items en plusieurs catégories qui peuvent elles-mêmes être divisées en sous catégories, etc. Elles sont notamment utilisées dans le cas où les items représentent des produits (dans ce cas on peut avoir par exemple les catégories "produits alimentaires", "vêtements", et les sous catégories "fruits et légumes", "boissons", etc.). Une taxonomie permet d'introduire de nouvelles contraintes syntaxiques

$$\mathcal{C}_{\mathrm{anc}(i)}(S) = (S \cap \text{ancêtre}(i) \neq \emptyset),$$
$$\mathcal{C}_{\neg\mathrm{anc}(i)}(S) = (S \cap \text{ancêtre}(i) = \emptyset),$$
$$\mathcal{C}_{\mathrm{desc}(i)}(S) = (S \cap \text{descendant}(i) \neq \emptyset),$$
$$\mathcal{C}_{\neg\mathrm{desc}(i)}(S) = (S \cap \text{descendant}(i) = \emptyset),$$

où les fonctions ancêtre et descendant sont définies à l'aide la fermeture transitive r^* de la taxonomie r :

$$\text{ancêtre}(i) = \left\{ i' \in \mathsf{Items} \mid r^*(i', i) \right\},$$
$$\text{descendant}(i) = \left\{ i' \in \mathsf{Items} \mid r^*(i, i') \right\}.$$

En fait ces nouvelles contraintes primitives ne sont que du sucre syntaxique. Elles peuvent en effet être récrites en utilisant les contraintes du type $\mathcal{C}_i$ et $\mathcal{C}_{\neg i}$, par exemple :

$$\mathcal{C}_{\mathrm{desc}(i)}(S) = \bigvee_{j \in \text{descendant}(i)} \mathcal{C}_j(S).$$

La contrainte syntaxique $\mathcal{C}_{\text{syntaxique}}$ considérée par les auteurs est une combinaison arbitraire de contraintes primitives (c'est-à-dire une formule du calcul des prédicats sans quantificateur et dont les termes sont des contraintes primitives au sens défini ci-dessus). Ils considèrent que cette formule est sous une forme normale disjonctive, i.e., $\mathcal{C}_{\text{syntaxique}} = D_1 \vee D_2 \vee \cdots \vee D_m$ où chaque D_k est de la forme $\mathcal{C}_{k1} \wedge \mathcal{C}_{k2} \wedge \cdots \wedge \mathcal{C}_{kn_k}$ et où $\mathcal{C}_{kj}$ est une contrainte primitive du type $\mathcal{C}_i$ ou $\mathcal{C}_{\neg i}$. On remarque que chaque D_k est la conjonction d'une contrainte monotone et d'une contrainte anti-monotone. En effet, pour chaque D_k, il existe deux itemsets V et W tels que D_k puisse se récrire sous la forme $D_k(S) = (V \subseteq S \subseteq W)$, par exemple, si $D_k = \mathcal{C}_{\mathtt{A}} \wedge \mathcal{C}_{\mathtt{B}} \wedge \mathcal{C}_{\mathtt{D}} \wedge \mathcal{C}_{\neg\mathtt{E}}$ alors $V = \mathtt{ABD}$ et $W = \mathsf{Items} \setminus \{\mathtt{E}\}$.

Les auteurs présentent des algorithmes pour extraire les itemsets satisfaisant la contrainte $\mathcal{C} = \mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\text{syntaxique}}$. Les deux premiers algorithmes (MultipleJoins et Reorder) utilisent une relaxation monotone de la contrainte syntaxique. Ils montrent comment calculer à partir de

la contrainte syntaxique sous forme normale disjonctive un itemset T tel que tout itemset satisfaisant la contrainte syntaxique satisfasse aussi la contrainte monotone $S \cap T \neq \emptyset$. Cette contrainte est alors poussée dans Apriori pour obtenir les algorithmes MultipleJoins et Reorder (Reorder est une simplification de MultipleJoins). Le troisième algorithme présenté, Direct, pousse entièrement la contrainte syntaxique pendant l'extraction.

On peut montrer que l'algorithme Direct peut pousser n'importe quelle contrainte syntaxiques. Pour cela, il suffit de montrer que toute contrainte syntaxique peut se mettre sous la forme qu'utilise Direct : Nous avons défini une contrainte syntaxique comme une contrainte $\mathcal{C}$ telle que la théorie $\mathsf{Th}(bd, 2^{\mathsf{Items}}, \mathcal{C})$ ne dépende pas de la base de données bd. Une contrainte syntaxique est donc entièrement caractérisée par sa théorie. Il faut donc voir si, pour tout sous ensemble $\mathcal{T}$ de 2^{Items}, il existe une contrainte syntaxique $\mathcal{C}$ écrite sous la forme utilisée par Direct et telle que $\mathsf{Th}(bd, 2^{\mathsf{Items}}, \mathcal{C}) = \mathcal{T}$. En fait, il suffit de prendre la contrainte

$$\mathcal{C}(S) = \bigvee_{V \in \mathcal{T}} (S \subseteq V \wedge V \subseteq S).$$

Bien sûr, cette écriture n'est pas optimale et le nombre de disjonctions peut être très grand. L'article ne discute d'ailleurs pas sur la complexité de la réécriture d'une contrainte syntaxique sous la forme normale disjonctive. Or, cette étape peut être coûteuse et peut mener à une formule comportant beaucoup de termes, ce qui est pénalisant pour l'algorithme.

Algorithme CAP

Dans [NLHP98], les auteurs présentent l'algorithme CAP pour extraire des itemsets fréquents satisfaisant une contrainte $\mathcal{C} = \mathcal{C}_{\gamma\text{-freq}} \wedge \mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\text{succ}}$ où $\mathcal{C}_{\mathsf{am}}$ est une contrainte anti-monotone syntaxique et $\mathcal{C}_{\text{succ}}$ une contrainte succincte.

Définition 17 (contrainte succincte [NLHP98]) *Une contrainte $\mathcal{C}$ est succincte*[1] *si elle est syntaxique et s'il existe des ensembles d'item I_1,I_2,...I_k tels que la contrainte puisse s'écrire sous la forme*

$$\mathcal{C}(S) = S \subseteq I_1 \wedge S \not\subseteq I_2 \wedge \cdots \wedge S \not\subseteq I_k.$$

On voit sur cette définition qu'une contrainte succincte est la conjonction d'une contrainte anti-monotone ($S \subseteq I_1$) et d'une contrainte monotone ($S \not\subseteq I_2 \wedge \cdots \wedge S \not\subseteq I_k$). Il est également clair qu'une conjonction de contraintes succinctes est succincte.

Les contraintes syntaxiques considérées dans cet article sont des conjonctions de contraintes primitives. Les contraintes primitives utilisées sont de deux types. Celles du premier type peuvent être construites comme des conjonctions de contraintes du type $\mathcal{C}_i$ et $\mathcal{C}_{\neg i}$ (définies précédemment). Les contraintes du second type sont des contraintes utilisant des agrégats. Pour les définir, on suppose l'existence d'une fonction v qui associe une valeur (entière ou

[1] Cette définition n'est pas celle des auteurs. En fait, ceux-ci ne donnent pas de définition formelle précise d'une contrainte succincte. Nous avons donc "inféré" cette définition à partir de leur article.

réelle) à chaque item. On peut ensuite définir des fonctions d'agrégat de la manière suivante :

$$\begin{aligned} |S| &= \text{ le nombre d'items de } S \\ \mathrm{MAX}(S) &= \max\{v(I) \mid I \in S\}, \\ \mathrm{MIN}(S) &= \min\{v(I) \mid I \in S\}, \\ \mathrm{SOM}(S) &= \sum_{I \in S} v(I), \\ \mathrm{MOY}(S) &= \frac{\mathrm{SOM}(S)}{|S|}. \end{aligned}$$

À partir de ces agrégats, les auteurs introduisent de nouvelles contraintes syntaxiques élémentaires de la forme $\mathrm{AGR}(S)\theta n$ où AGR est une fonction d'agrégat, θ un opérateur de $\{=, <, >\}$ et n un entier ou un réel. Si on considère une base de données binaire où les items sont des produits, la fonction v peut par exemple donner le prix des items. Les fonctions d'agrégat permettent alors d'exprimer que l'on veut, e.g., des itemsets dont le prix moyen des items est supérieur à un certain seuil ($\mathcal{C}(S) = \mathrm{AVG}(S) > 10$). Parmi ces contraintes, certaines sont anti-monotones (ex : $\mathrm{SUM}(S) < 100$ si les valeurs des items sont toutes positives, $\mathrm{MIN}(S) > 10$), succinctes (ex : $\mathrm{MAX}(S) > 10$, $|S| > 3$) ou n'ont aucune de ces propriétés et doivent donc être relaxées pour pouvoir être poussées dans l'algorithme CAP (e.g., $\mathrm{SUM}(S) < 10$, $\mathrm{AVG}(S) < 10$).

L'algorithme CAP présente une amélioration au niveau de la génération des candidats par rapport à l'algorithme Direct. Par contre, il ne peut pas utiliser toutes les contraintes syntaxiques comme le fait l'algorithme Direct. En particulier, l'algorithme CAP ne prend en compte que les *conjonctions* de contraintes primitives. Cependant, certaines contraintes utilisant les fonctions d'agrégat (comme $\mathrm{SUM}(S) < 100$) sont gérées par CAP. A priori, elles peuvent aussi l'être par l'algorithme Direct mais pour cela il faudrait les récrire sous forme normale disjonctive en utilisant seulement les contraintes du type $\mathcal{C}_i$ et $\mathcal{C}_{\neg i}$. Or cette étape de réécriture peut être très coûteuse et la contrainte récrite risque d'être très grande. On ne connaît pas en effet de méthode efficace pour transformer une contrainte syntaxique utilisant des agrégats en une contrainte écrite sous la forme utilisée par Direct. Ainsi, l'utilisation des contraintes d'agrégats dans Direct, bien que théoriquement possible, n'est pas forcément faisable en pratique.

Algorithme FP-growth et ses extensions

L'algorithme d'extraction d'itemsets sous contraintes présenté dans [HP00, PHL01] est une extension de FP-growth [HPY00] qui lui ne traite que la contrainte de fréquence minimale. Dans ces algorithmes, les auteurs utilisent le concept de base de données conditionnelle. Étant donné un ordre sur les items, une base de données conditionnelle de préfixe S (S est un itemset) contient les transactions de la base de données contenant S et réduites aux items plus grands que ceux de S. Par exemple, si on considère la base de données binaire $\{\mathtt{ABCDE}, \mathtt{BCDF}, \mathtt{ACEG}, \mathtt{ABCF}\}$ et l'ordre alphabétique sur les items, alors la base de données conditionnelle de préfixe $\mathtt{AC}$ est constituée des transactions $\{\mathtt{DE}, \mathtt{EG}, \mathtt{F}\}$. Ces bases de données conditionnelles sont stockées dans des arbres de préfixe améliorées (appelés FP-trees par les auteurs). L'espace de recherche est exploré en profondeur d'abord en utilisant un ordre sur les items (un exemple de ce parcours est donné dans la figure 2.2).

L'algorithme utilise une conjonction de contraintes de la forme $\mathcal{C}_{\gamma\text{-freq}} \wedge \mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge C_{\mathrm{conv}}$. Les contraintes $\mathcal{C}_{\mathsf{am}}$, $\mathcal{C}_{\mathsf{m}}$ et C_{conv} sont des contraintes syntaxiques respectivement anti-monotone, monotone et convertible (anti-)monotone.

L'ordre sur les items est déterminé soit par la fréquence des items soit par la contrainte convertible s'il y en a une (on ne peut utiliser qu'une contrainte convertible à la fois car chaque contrainte convertible impose un ordre sur les items et donc sur l'exploration du treillis des itemsets).

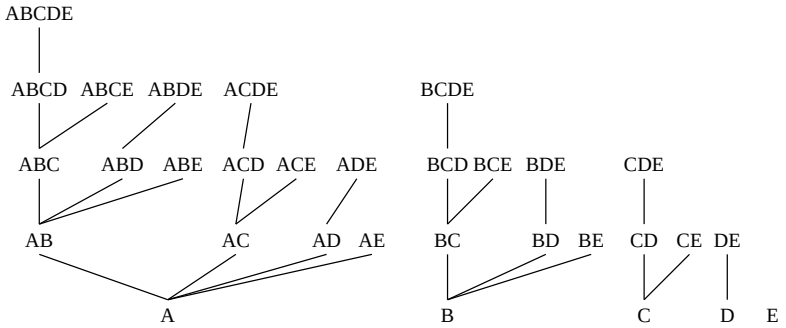


FIG. 2.2 – Cette figure montre un parcours en profondeur du treillis des itemsets lorsque l'ordre sur les items est alphabétique. Le parcours se fait alors de bas en haut et de droite à gauche en suivant les arêtes. Ainsi, les premiers itemsets considérés sont, dans l'ordre, `E, D, DE, C, CE, CD, CDE, B, ...`

Définition 18 (contrainte convertible (anti-)monotone [PHL01])
Soit $\mathcal{C}$ une contrainte, $\mathcal{C}$ est convertible *anti-monotone (respectivement monotone) si il existe une relation d'ordre $\lhd$ sur les items telle que, si un itemset S satisfait la contrainte, alors tous les itemsets préfixes de S (respectivement, tous les itemsets dont S est préfixe) par $\lhd$ satisfont aussi $\mathcal{C}$. Si S est un itemset s'écrivant $I_1 I_2 \dots I_k$ où les I_i sont des items tels que $\forall 1 < i \leq k$, $I_{i-1} \lhd I_i$, alors un itemset préfixe de S par $\lhd$ est un itemset de la forme $I_1 I_2 \dots I_j$ avec $1 \leq j \leq k$.*

Un exemple de contrainte convertible anti-monotone est la contrainte $\text{AVG}(S) < 10$, si on ordonne les items par valeur croissante de la fonction v. Par exemple, si l'itemset `ABCE` satisfait cette contrainte, alors, comme $v(\texttt{A}) \leq \dots \leq v(\texttt{E})$, $\text{MOY}(\texttt{ABC}) \leq \text{MOY}(\texttt{ABCE})$ et donc $\text{MOY}(\texttt{ABC}) < 10$ et l'itemset `ABC` satisfait bien aussi la contrainte. Il en est de même pour les autres préfixes `AB` et `A` de `ABCE`. Cette contrainte est aussi convertible monotone : il suffit d'ordonner les items par valeur décroissante de v.

Cet algorithme, à la différence de ceux vus précédemment, n'utilise pas les contraintes monotones aussi efficacement. Si un itemset ne satisfait pas la contrainte monotone, il sera tout de même généré si un des itemsets qui le suit (en considérant l'ordre d'énumération de la figure 2.2) la satisfait.

Exemple 15 *Considérons la contrainte monotone* `BD` $\in S$. *Bien qu'ils ne satisfassent pas cette contrainte, les itemsets* `A`, `AB`, `ABC`, `B` *et* `BC` *sont générés par* FP-growth. *En effet, chacun des arbres enracinés en ces itemsets contient des itemsets qui satisfont la contrainte (voir la figure 2.2). En revanche, l'algorithme* CAP *ne génère aucun de ces itemsets.*

Seuil de fréquence variable

Dans [WHH00], les auteurs proposent d'exploiter une contrainte de fréquence minimale avec un seuil variable selon les itemsets. Nous ne détaillerons pas la façon de préciser ces différents seuils. On peut par exemple indiquer que tous les itemsets contenant `A` doivent

être extraits avec un seuil de 0.1, ceux qui contiennent `B` avec un seuil de 0.25 et les autres avec un seuil de 0.2. Si plusieurs seuils différents peuvent s'appliquer sur un itemset alors les auteurs prennent la convention de garder le seuil le plus faible (par exemple ici, le seuil pour les itemsets contenant `AB` sera 0.1). Les auteurs montrent ensuite comment relaxer cette contrainte pour pouvoir la pousser dans un algorithme par niveau.

2.4.2 Extraction de séquences

De nombreuses sources d'information ont une nature séquentielle, c'est-à-dire qu'elles sont composées d'éléments qui sont ordonnés de manière spatiale ou temporelle. Cela inclus, par exemple, des chaînes de caractères et donc des textes, des séquences de bases sur un brin d'ADN, des séquences d'acides aminés dans la formule d'une protéine, des séquences temporelles stockées dans des logs : logs d'erreurs dans des systèmes, logs de connexions, etc. L'utilisation des itemsets et des règles d'association sur ces données conduit à une perte d'information importante. Pour tirer parti de l'ordre des séquences, des algorithmes spécifiques ont été introduits. Les premiers algorithmes traitant de l'extraction de séquences sont présentés dans [AS95, MTV95, MT96a]. Des algorithmes ont ensuite été développés pour prendre en compte diverses contraintes, on peut citer les algorithmes GSP [AS95, SA96], Spirit [GRS99], Spade [Zak01] et c-Spade [Zak00b], ou pour améliorer les performances de l'extraction des motifs fréquents [PHMA+01]. Des travaux sont en cours pour l'extraction de séquences du premier ordre [LRK02].

Définition 19 (séquence) *Soit* Items *un ensemble. Un événement est un sous ensemble de* Items *(autrement dit un événement est aussi un itemset). Une séquence est une suite notée* $\langle \mathtt{e}_1, \mathtt{e}_2, \ldots, \mathtt{e}_n \rangle$ *d'événements. L'entier n est appelé la longueur de la séquence. La taille de la séquence est la somme des cardinaux des événements qui la composent et enfin la largeur d'une séquence est le cardinal du plus grand événement de la séquence. Dans une base de données de séquences, chaque événement d'une séquence est généralement daté par un entier, notée e.date et tel que* $\forall i$, $1 < i \leq n$, $e_{i-1}.date < e_i.date$.

Si on considère la séquence $s = \langle \mathtt{a}, \mathtt{bc}, \mathtt{c}, \mathtt{a} \rangle$, la taille de cette séquence vaut 5, sa longueur vaut 4 et sa largeur vaut 2. On remarque que la taille d'une séquence est bornée par le produit de sa longueur par sa largeur, il y a égalité quand il y a le même nombre d'items dans chaque événement.

Définition 20 (sous séquence (contiguë)) *Une séquence* $s = \langle \mathtt{e}_1, \mathtt{e}_2, \ldots, \mathtt{e}_n \rangle$ *est une sous séquence de* $s' = \langle \mathtt{e}'_1, \mathtt{e}'_2, \ldots, \mathtt{e}'_m \rangle$, *noté* $s \preceq s'$, *si il existe une suite strictement croissante* α_1, α_2, ..., α_n, *telle que* $\forall i$, $1 \leq i \leq n$, $e_i \subseteq e'_{\alpha_i}$. *Une séquence* $s = \langle \mathtt{e}_1, \mathtt{e}_2, \ldots, \mathtt{e}_n \rangle$ *est une sous séquence* contiguë *de* $s' = \langle \mathtt{e}'_1, \mathtt{e}'_2, \ldots, \mathtt{e}'_m \rangle$ *si il existe un entier k tel que* $\forall i$, $1 \leq i \leq n$, $e_i \subseteq e'_{k+i}$.

La séquence $\langle \mathtt{a}, \mathtt{b}, \mathtt{c} \rangle$ est une sous séquence contiguë de $s = \langle \mathtt{a}, \mathtt{bc}, \mathtt{c}, \mathtt{a} \rangle$, $\langle \mathtt{bc}, \mathtt{a} \rangle$ est une sous séquence de s (mais pas une sous séquence contiguë), par contre, $\langle \mathtt{b}, \mathtt{c}, \mathtt{c} \rangle$ n'est pas une sous séquence de s.

Les séquences jouent le même rôle que les itemsets dans l'extraction des règles d'association. Les motifs similaires aux règles d'association pour les itemsets sont les motifs séquentiels. Un motif séquentiel sert à exprimer qu'une séquence est souvent suivie, précédée ou entrelacée avec une autre séquence. On ne donnera pas de définition formelle mais juste un exemple : le

motif séquentiel $\langle \mathtt{a\_}, \mathtt{bc}, \mathtt{\_} \rangle \Rightarrow \langle \mathtt{c\_}, \mathtt{\_\_}, \mathtt{c} \rangle$ signifie que lorsque la séquence $s = \langle \mathtt{a}, \mathtt{bc} \rangle$ apparaît dans les données alors on constate que la séquence $s' = \langle \mathtt{c}, \mathtt{c} \rangle$ apparaît souvent entrelacée avec s de telle sorte que le premier événement de s' apparaisse simultanément avec le premier événement de s et le deuxième événement de s' soit après le dernier événement de s. Dans le motif séquentiel, les symboles "_" permettent de déterminer où sont placés les items d'une des séquences par rapport aux items de l'autre séquence. On remarque qu'à la différence des itemsets, l'ensemble des séquences construites sur Items est infini. Il existe des séquences de longueur arbitrairement élevée.

Les bases de données sur lesquelles on peut extraire des séquences sont de deux types. Elles peuvent être constituées d'une seule très longue séquence S. Dans ce cas, la fréquence d'une séquence s peut être définie comme le nombre de fois où elle apparaît dans la séquence S. Cependant, donner une définition formelle de la fréquence n'est pas trivial [MTV97] et il y a d'ailleurs plusieurs définitions possibles. L'algorithme présenté dans [MTV97] permet d'extraire ces séquences avec différentes notions de fréquence.

La base de données peut aussi être constituée d'un ensemble de séquences, c'est le cas le plus courant. Dans ce cas, la fréquence d'une séquence s dans la base de données est définie comme le nombre de séquences de la base de données qui admettent s comme sous séquence.

Dans [GRS99, GR00], les auteurs présentent plusieurs variations d'un algorithme (Spirit) pour extraire des séquences fréquentes de largeur 1 satisfaisant une expression régulière. Dans la suite de ce paragraphe, une séquence qui satisfait l'expression régulière sera dite valide. Par exemple, si les événements constituant une séquence sont des lettres, l'expression régulière `a*(bb|cc)e` caractérise les séquences qui commencent par une suite de `a` suivie soit de `bbe` soit de `cce`. Cette contrainte syntaxique n'est évidemment pas anti-monotone dans le cas général. Les différents algorithmes présentés dans cet article utilisent des relaxations de plus en plus restrictives de la contrainte syntaxique. Le premier algorithme, Spirit(N), utilise une relaxation anti-monotone de la contrainte syntaxique. Cette contrainte $\mathcal{C}_N$ est définie de la manière suivante : une séquence s satisfait $\mathcal{C}_N$ si tous les items apparaissant dans s apparaissent aussi dans l'expression régulière. Avec notre exemple d'expression régulière, $\mathcal{C}_N(s)$ est vraie si la séquence s ne contient que les items `a`, `b`, `c` et `e`. La contrainte $\mathcal{C}_L$ utilisée par le deuxième algorithme Spirit(L) est satisfaite par une séquence s si s est une séquence légale par rapport à l'expression régulière. Une séquence s est dite légale si il existe une séquence s' valide telle que s soit une sous séquence contiguë suffixe de s' (i.e., on peut obtenir la séquence s' en rajoutant des items au début de s). Par exemple, la séquence `cce` est une séquence légale par rapport à notre exemple d'expression régulière. Cette contrainte n'est pas anti-monotone (la sous séquence `cc` de `cce` n'est pas légale par rapport à notre exemple). L'algorithme Spirit(V) utilise la contrainte $\mathcal{C}_V$ qui est satisfaite par les sous séquences contiguës des séquences valides par rapport à l'expression régulière. Finalement, la contrainte $\mathcal{C}_R$ utilisée par Spirit(R) requiert qu'une séquence soit valide. Pour les trois premiers algorithmes, une étape finale est nécessaire pour éliminer les séquences non valides. On remarque que les collections de séquences extraites par ces algorithmes sont incluses les unes dans les autres, $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_R) \subseteq \mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_V) \subseteq \mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_L) \subseteq \mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_N)$.

L'algorithme c-Spade [Zak00b] (qui est une extension de Spade [Zak01] traitant la contrainte de fréquence minimale) permet de traiter un certain nombre de contraintes. L'algorithme explore l'espace des séquences soit niveau par niveau, soit en profondeur d'abord. Les contraintes utilisées sont pour la plupart anti-monotones (comme la contrainte de longueur maximale, de largeur maximale). L'auteur introduit trois autres contraintes intéressantes car elles ne portent pas sur les séquences extraites elles-mêmes mais la manière dont est calculée leur fréquence. Soit s une séquence candidate et S une séquence dans la base de données. Tradi-

tionnellement, la séquence s est considérée comme apparaissant dans S si elle est une sous séquence de s. La contrainte "max-gap" impose que pour que s soit considérée comme apparaissant dans S, il faut de plus que les événements de s n'apparaissent pas dans S à des dates séparées de plus d'un seuil donné. Par exemple, si le max-gap vaut 10 et que $s = \langle \mathtt{ab}, \mathtt{c}, \mathtt{d} \rangle$, s ne sera compté dans une séquence S de la base de données que si les différences de dates dans S entre les événements `ab` et `c` d'une part et `c` et `d` d'autre part sont inférieurs à 10. Les deux autres contraintes sont la contrainte "min-gap" qui spécifie un écart minimal entre les événements d'une séquence et une contrainte qui spécifie l'écart maximal entre les événements extrêmes d'une séquence. On vérifie facilement que ces deux dernières contraintes sont anti-monotones (ce qui n'est pas le cas de "max-gap").

2.4.3 Fragments

Les fragments moléculaires [dRK01, KdRH01, KdR01] sont des motifs proches des séquences. Un *fragment moléculaire* est une séquence de largeur 1 servant à représenter une chaîne d'atomes ainsi que les liaisons chimiques qui les joignent. Par exemple, le fragment "o–s–c" représente un atome d'oxygène lié par une liaison simple à un atome de souffre lié à un atome de carbone. Les fragments moléculaires sont extraits d'une base de données de molécules représentées sous la forme d'un graphe où chaque sommet est un atome et chaque arête une liaison atomique. On dira qu'un fragment est couvert par une molécule si le fragment considéré comme un graphe est un sous graphe de la molécule. On peut ainsi définir la fréquence d'un fragment dans une base de données de molécules. La relation de spécialisation entre les fragments est définie par $f_1 \preceq f_2$ si f_1 est une sous séquence contiguë de f_2 ou de l'inverse de f_2. Ainsi, "c–o" est un sous fragment de "c–o–s" et de "n–o–c" mais pas de "c–s–o". Deux fragments représentés par des séquences inversées sont considérés comme identiques, ainsi "c–o–s" et "s–o–c" représentent le même fragment.

L'algorithme Molfea présenté dans [dRK01] permet d'extraire des fragments satisfaisant une conjonction de contraintes primitives. Les contraintes primitives sont :

- $f \preceq p$, $p \preceq f$, $\neg(f \preceq p)$ et $\neg(p \preceq f)$ où p est un fragment constant et f est le fragment qui doit être extrait ;
- $\mathsf{freq}(f, D) > \gamma$ ou $\mathsf{freq}(f, D) < \gamma$.

L'intérêt des deux contraires de fréquence apparaît quand on les utilise simultanément avec des jeux de données différents. Ainsi la contrainte $(\mathsf{freq}(f, Act) > \gamma_1) \wedge (\mathsf{freq}(f, InAct) < \gamma_2)$ permet d'extraire des fragments fréquents dans l'ensemble de molécules Act et infréquents dans $InAct$. Si on imagine que Act contient des molécules actives (sur une certaine bactérie, par exemple) et $InAct$ des molécules inactives, on trouvera des fragments qui caractérisent les molécules actives. Cet algorithme construit les frontières $\mathcal{S}$ et $\mathcal{G}$ de l'ensemble des fragments solutions de manière incrémentale en traitant les contraintes les unes après les autres. Les frontières $\mathcal{S}$ et $\mathcal{G}$ sont initialisées avec les fragments maximal et minimal et sont ensuite mise à jour en utilisant soit un algorithme niveau par niveau de type Apriori lorsque la contrainte est du type $\mathsf{freq}(f, D) > \gamma$ ou $\mathsf{freq}(f, D) < \gamma$, soit l'algorithme de Mellish [Mel92] dans les autres cas.

2.4.4 Requêtes datalog

Les types de motifs que nous avons présentés jusqu'à maintenant souffrent de certaines limitations. D'une part les données sont extraites d'une seule table de données ou alors de plusieurs tables ayant le même schéma (même si l'utilisation de taxonomies permet d'enrichir les

données). D'autre part, les motifs considérés sont assez simples. L'algorithme Warmr [DT01, DDR97] permet d'extraire des requêtes datalog fréquentes. Considérons par exemple la base de données de la table 2.1 (tirée de [DT01]) constituée de deux tables "Achète" et "Parent".

Parent	
alain	bill
alain	carole
bill	zoe
carole	diana

Achète	
alain	vin
bill	cola
bill	pizza
diana	pizza

TAB. 2.1 – Une base de données constituée de deux tables.

Dans ce cas l'algorithme Warmr extrait des requêtes datalog fréquentes. Un exemple d'une requête datalog est "parent(X, Y) achète(X,vin) achète(Y,pizza)" dans lesquels X et Y sont des variables.

Dans le cas de Warmr, les motifs valides ne sont pas définis par une contrainte qu'ils doivent satisfaire comme dans les autres travaux que nous avons considérés jusqu'ici. L'utilisateur doit plutôt fournir un *biais de langage* constitué d'une clef qui est le motif de base et d'atomes qui permettent d'étendre un motif valide pour former un nouveau motif valide plus complexe, par exemple, la clef "parent($-,-$)" et les atomes "achète($+$,pizza) ; achète($+$,vin)". Les symboles $+$ et $-$ indiquent des contraintes de mode. Le signe $+$ dans "achète($+$,pizza)" indique que lorsqu'on ajoute le prédicat "achète(X,pizza)" à un motif φ, la variable X doit déjà apparaître dans φ. Le signe $-$ indique que la variable ne doit pas apparaître dans φ. Ainsi, les requêtes datalog valides pour cette clef et ces atomes sont :

parent(X,Y)
parent(X,Y) achète(X,pizza)
parent(X,Y) achète(Y,pizza)
parent(X,Y) achète(X,vin)
parent(X,Y) achète(Y,vin)
parent(X,Y) achète(X,pizza) achète(Y,pizza)
...

Par contre, la requête "parent(X,Y) achète(Z,pizza)" n'est pas valide car elle viole la contrainte de mode (la variable Z n'apparaît pas dans le motif "parent(X,Y)").

L'extraction de requêtes datalog est plus complexe que l'extraction de motifs plus simples. En effet, l'espace de recherche est souvent très grand (si le nombre d'atome est important), le calcul de la fréquence d'une requête nécessite l'évaluation de cette requête dans la base de données (i.e., il faut trouver toutes les substitutions des variables de la requête qui donnent une formule vraie dans les données). Par exemple, avec la requête "parent(X,Y) achète(X,vin)" dans les données de la table 2.1, les substitutions ($X :=$ alain; $Y :=$ bill) et ($X :=$ alain; $Y :=$ carole) conviennent, ce qui donne une fréquence de 2 pour cette requête. L'évaluation de la relation de spécialisation est également très coûteuse (ce n'est pas juste un test d'inclusion comme avec les itemsets à cause des variables), ce qui rend l'étape d'élagage beaucoup plus coûteuse que dans le cas des autres types de motifs.

2.4.5 Autres types de motifs

D'autres types de motifs ont été étudiés dans la littérature. L'extraction de sous graphes dans une base de données de graphes est présentée dans [IWM00, DTK98]. L'application visée est encore la découverte de structures fréquentes dans des molécules. Des travaux ont commencé sur l'extraction d'arbres de décision sous contraintes [GR00, GHRS00]. Les contraintes considérées concernent la taille de l'arbre et sa précision. L'intérêt de ce travail est qu'il ne traite pas les contraintes en post-traitement comme cela est fait traditionnellement avec les arbres de décision. Ainsi, l'algorithme est très efficace quand on demande par exemple un petit arbre. L'utilisateur peut donc choisir d'obtenir un arbre simple et assez imprécis en un temps court ou un arbre plus précis et donc plus complexe et plus long à construire.

D'autres travaux [ZDK00] s'intéressent à la découverte de motifs émergeant. Un motif émergeant est un itemset S tel que $\mathsf{freq}(S, bd_1)/\mathsf{freq}(S, bd_2) > \rho$ où ρ est un seuil défini par l'utilisateur et bd_1 et bd_2 sont des bases de données binaires. Ces travaux sont à mettre en rapport avec [dRK01] où les auteurs cherchent les fragments fréquents dans une base de données et infréquents dans une autre (dans ce cas, il y a deux seuils de fréquence au lieu d'un ici).

Dans [BAG00], les auteurs proposent l'algorithme Dense-Miner pour extraire des règles de tête fixée (spécifiée par l'utilisateur) dans des données denses. Les règles doivent satisfaire des contraintes de fréquence minimum, de confiance minimum et d'amélioration minimum (*minimum improvement*). Cette dernière contrainte impose que la différence entre la confiance d'une règle et la confiance de toutes ses sous règles strictes (une sous règle stricte est obtenue en supprimant au moins un items dans le corps de la règle) soit supérieure à un seuil d'amélioration minimum.

Enfin, [BA99] présente un travail intéressant d'extraction avec une contrainte d'optimisation. Les auteurs montrent comment extraire un ensemble de règles d'association qui a la propriété remarquable de contenir toutes les règles maximisant une large classe de mesures d'intérêt (fréquence, confiance, laplace, gain [Fuk96], conviction [BMUT97], intérêt [BMUT97], etc.).

2.5 Discussion sur l'état de l'art

Dans cette section nous faisons une étude comparative de la manière dont les différents travaux en extraction de motifs sous contraintes traitent les problèmes de la génération des candidats, de l'élagage et de l'arrêt de l'algorithme.

2.5.1 Génération des candidats

L'une des étapes les plus importantes des algorithmes d'extraction de motifs est la génération des candidats. C'est en effet cette étape qui détermine d'une part la complétude de l'algorithme et d'autre part une partie de son efficacité. Pour que l'algorithme soit complet, la fonction de génération des candidats doit assurer que tous les motifs potentiellement valides sont bien générés. Cette fonction doit aussi utiliser au maximum les différentes contraintes syntaxiques et les motifs valides du niveau précédent pour ne pas générer de candidats inutilement et n'explorer qu'une partie la plus réduite possible de l'espace de recherche.

Il existe plusieurs méthodes pour générer les candidats. On peut en distinguer deux grands types selon que l'on utilise ou non les motifs valides du niveau précédent.

Génération par jointure ou par extension

L'algorithme Apriori génère les candidats en joignant deux itemsets de $\mathcal{S}_k$ (i.e., des itemsets de taille k qui satisfont la contrainte) qui partagent les mêmes $k-1$ premiers items. Cela suppose que les items sont ordonnés. Dans l'algorithme Apriori, on peut choisir un ordre quelconque sans affecter la complétude de l'algorithme. Cette méthode de génération est complète quand la contrainte est anti-monotone. En effet, dans ce cas, tout itemset de $\mathcal{S}_{k+1}$ a tous ses sous ensembles de taille k dans $\mathcal{S}_k$, et donc a fortiori les deux sous ensembles utilisés par la fonction pour le générer. Plusieurs autres algorithmes génèrent les candidats du niveau $k+1$ en joignant deux motifs valides du niveau k. Les algorithmes d'extraction de motifs séquentiels Spade [Zak01] et Spirit(N) [GRS99] utilisent une méthode de jointure (appelée "jointure temporelle" dans [Zak01]) quand les contraintes sont anti-monotones.

Toutefois cette méthode n'est pas utilisable dans le cas général. Si la contrainte $\mathcal{C}$ n'est plus anti-monotone, $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C})$ n'est plus fermée vers le bas, il peut y avoir des "trous". Dans ce cas, si φ est un motif valide, on ne peut plus garantir que les deux motifs qui servent à le générer satisfont aussi $\mathcal{C}$. La méthode de jointure n'est alors plus complète.

Exemple 16 *Soit $\mathcal{C}$ la contrainte définie par $\mathcal{C}(S) = \mathtt{B} \in S$. Cette contrainte n'est pas anti-monotone. Supposons que* Items $= \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}\}$. *L'ensemble des itemsets de taille 2 satisfaisant la contrainte est donc $\mathcal{S}_2 = \{\mathtt{AB}, \mathtt{BC}, \mathtt{BD}, \mathtt{BE}\}$. La fonction de génération des candidats d'*Apriori *ne produira pas* ABC *comme candidat pour le niveau 3 car* AC *n'est pas dans $\mathcal{S}_2$. Ainsi,* Apriori *n'est pas complet dans ce cas où la contrainte n'est pas anti-monotone.*

Dans certains cas, la technique d'Apriori est quand même utilisable. Dans les algorithmes CAP et Reorder, les itemsets candidats sont générés comme dans Apriori sauf dans les premiers niveaux où une technique ad-hoc est utilisée (du fait que les contraintes poussées ne sont pas toutes anti-monotones). De plus, l'ordre sur les items n'est plus indifférent, il doit être choisi de manière à assurer la complétude de la génération. C'est également en utilisant la génération des candidats d'Apriori et en choisissant un ordre judicieux sur les items que l'algorithme présenté dans [WHH00] fonctionne. La contrainte de seuils de fréquence variables selon les itemsets qui est utilisée n'est pourtant pas anti-monotone, mais les auteurs montrent que l'on peut construire une relaxation de cette contrainte et un ordre sur les items qui assure que tout itemset satisfaisant cette relaxation peut être généré de la même manière que dans Apriori.

D'autres algorithmes utilisent une fonction qui génère les candidats de niveau $k+1$ en prenant chaque motif valide du niveau k et en l'étendant de toutes les manières possibles. Pour qu'un algorithme utilisant cette technique soit complet, il est évidemment nécessaire que tout motif φ du niveau $k+1$ satisfaisant la contrainte possède un sous motif au niveau k qui satisfait aussi la contrainte. Sinon, le motif φ ne pourrait pas être généré. La contrainte doit donc posséder une propriété minimale que nous appellerons l'anti-monotonicité faible.

Définition 21 (anti-monotonicité faible) *Une contrainte $\mathcal{C}$ est faiblement anti-monotone si tout motif la satisfaisant couvre au moins un sous motif qui la satisfait également. On rappelle qu'un motif φ couvre un motif μ si $\mu \prec \varphi$ et s'il n'existe pas de motif λ tel que $\mu \prec \lambda \prec \varphi$.*

La différence avec l'anti-monotonicité est que l'on n'impose pas à *tous* les sous motifs de satisfaire la contrainte. Par contre, la propriété de caractérisation de l'ensemble des motifs valides par la frontière n'est plus valable et une conjonction ou une disjonction de contraintes

faiblement anti-monotones n'est plus nécessairement faiblement anti-monotone. De plus, on ne peut plus utiliser la même étape d'élagage que dans Apriori (mais nous verrons cela dans la section suivante).

La propriété suivante est une conséquence directe de la définition.

Proposition 7 *Soit S un motif satisfaisant une contrainte faiblement anti-monotone. Alors il existe une chaîne de motifs S_1, ..., S_k telle que S_1 est le motif vide, $S_k = S$ et pour tout $1 < i \leq k$, S_i couvre S_{i-1}.*

lorsqu'on extrait des motifs sous contrainte faiblement anti-monotone, il est possible de se servir des motifs du niveau k pour générer ceux du niveau $k+1$. En effet, tous les motifs valides du niveau $k+1$ possèdent au moins un sous motif du niveau k valide. Ainsi, en construisant tous les motifs couvrant les motifs valides niveau k, on est sûr, d'après la propriété précédente, de bien être complet. Cette technique produit dans le cas général beaucoup de candidats, mais leur nombre peut être réduit en utilisant des techniques ad-hoc.

L'algorithme c-Spade [Zak00b], par exemple, utilise une technique légèrement modifiée dans le cas des contraintes faiblement anti-monotones "max-gap". Les candidats sont générés en effectuant la jointure des séquences de taille k avec celles de taille 2. Les auteurs montrent que cette méthode est complète et qu'elle génère moins de candidats que si la jointure était faite avec les séquences de taille 1.

L'algorithme Direct utilise une fonction de génération similaire. Les itemsets valides du niveau k sont étendus en leur ajoutant un item fréquent.

Généralement, une contrainte est définie par une fonction qui prend la valeur vrai sur les motifs qui la satisfont et faux sur les autres. Par contre, cette fonction ne fournit pas de moyen pour énumérer les motifs satisfaisant la contrainte. Les biais de langages utilisés dans l'algorithme d'extraction de requêtes datalog fréquentes Warmr [DT01, DT99] définissent des motifs atomiques valides et indiquent comment ces motifs peuvent être étendus. Plutôt que de fournir une fonction qui teste la validité d'un motif, les biais de langages fournissent une méthode pour énumérer les motifs valides. Finalement, c'est l'utilisateur qui dans ce cas fourni la fonction de génération des candidats.

L'algorithme Spirit [GRS99] extrait des motifs séquentiels fréquents sous une contrainte syntaxique. La contrainte syntaxique est définie par une expression régulière. Par conséquent, elle n'a en général aucune bonne propriété comme l'anti-monotonicité. Cependant, les auteurs montrent comment relaxer cette contrainte pour obtenir diverses contraintes plus ou moins restrictives. Pour chacune de ces relaxations les auteurs proposent un algorithme. Dans les algorithmes Spirit(L) et Spirit(V), les relaxations de la contrainte considérées sont faiblement anti-monotones. Dans le cas de Spirit(L), les candidats de taille $k+1$ sont obtenus en joignant des séquences valides u et v de taille k telles que le préfixe de taille $k-1$ de u soit égal au suffixe de taille $k-1$ de v, on a donc finalement une fonction de génération très proche de celle d'Apriori. Dans Spirit(V), les séquences valides du niveau k sont étendues en utilisant l'automate qui représente l'expression régulière. On retrouve dans ce cas une méthode presque identique à celle utilisée par l'algorithme Warmr avec les biais de langage.

Les contraintes convertibles anti-monotones utilisées dans [PHL01] sont un autre exemple de contrainte faiblement anti-monotone. Dans ce cas, le sous ensemble d'un itemset S qui satisfait aussi la contrainte est tout simplement son préfixe quand les items sont ordonnés par l'ordre $\lhd$ de la définition 18 page 56 des contraintes convertibles anti-monotones. Leur algorithme est basé sur FP-growth [HPY00] et fait donc un parcours en profondeur du treillis des itemsets. Ce parcours utilise l'ordre $\lhd$ de la définition 18. Si on suppose que `A` $\lhd$ `B` $\lhd$

C $\lhd$ D $\lhd$ E, le parcours sera celui de la figure 2.2 page 56. Si on suppose que l'ordre est A $\lhd$ C $\lhd$ B $\lhd$ D $\lhd$ E alors le parcours sera celui de la figure 2.2 en ayant permuté B et C. Comme cet ordre de parcours est forcément unique, il est clair que cet algorithme ne peut utiliser plusieurs contraintes convertibles anti-monotones que s'il existe un ordre unique ayant la propriété de la définition 18 pour toutes les contraintes simultanément.

Génération directe

L'algorithme Spirit(R) [GRS99] génère ses candidats en énumérant toutes les séquences de taille k qui satisfont l'expression régulière. Ceci est fait en parcourant l'automate d'état fini qui reconnaît les séquences satisfaisant l'expression régulière. Cette technique n'utilise donc pas les séquences extraites au niveau précédent, elles sont utilisées seulement lors de l'étape d'élagage. Les auteurs montrent que cette technique n'est applicable que lorsque l'expression régulière est très sélective. Sinon, le nombre de candidats explose et l'algorithme devient inutilisable.

L'inconvénient de cette technique est qu'elle ne tire pas profit des motifs déjà extraits pour limiter le nombre de candidats. En particulier, tous les sur motifs d'un motif infréquent seront générés s'ils satisfont l'expression régulière. Bien que ces motifs soient ensuite élagués, il est possible, comme le remarquent les auteurs de Spirit, dans le cas où l'expression régulière est peu sélective, que le nombre de candidats explose. Dans ce cas, l'extraction ne sera plus possible alors même qu'un petit nombre seulement de ces motifs est fréquent. L'étape d'élagage est dans ce cas indispensable puisque c'est elle qui permet en définitive de réduire le nombre de candidats pour lesquels le calcul de la fonction d'évaluation sera fait (c'est-à-dire la passe sur les données qui constitue l'étape la plus coûteuse). Finalement, la génération directe des candidats ne semble pas très intéressante sauf si la contrainte est très sélective.

2.5.2 Critère d'arrêt de l'algorithme

L'espace de recherche étant très grand, on ne peut pas se permettre de l'explorer en totalité. Dans tous ces algorithmes d'extraction, il faut donc savoir décider quand l'extraction est finie. C'est-à-dire à quel moment l'algorithme a exploré une partie de l'espace de recherche suffisante pour pouvoir garantir qu'aucun motif valide n'existe dans la partie non explorée (on considère en effet des algorithmes complets).

Dans l'algorithme Apriori, l'extraction s'arrête dès qu'un niveau où l'ensemble des candidats est vide est atteint. Dans ce cas, on est sûr qu'aucun itemset satisfaisant la contrainte ne peut exister dans les niveaux supérieurs. Cette propriété est vraie tant que la contrainte poussée pendant l'extraction est au moins faiblement anti-monotone. En effet, tout motif satisfaisant une telle contrainte possède un sous motif au niveau inférieur qui la satisfait également. Ainsi, si l'ensemble des motifs valide est vide à un niveau donné, il ne peut exister de motif satisfaisant la contrainte dans les niveaux supérieurs.

Par contre, dans le cas où la contrainte n'est plus faiblement anti-monotone, il faut recourir à un test d'arrêt ad-hoc. Dans l'algorithme Spirit(R), les candidats sont générés en utilisant un automate. Pour pouvoir arrêter l'exploration, il faut s'assurer qu'aucune séquence plus grande satisfaisant la contrainte ne peut plus être générée par l'automate. Pour cela, les auteurs montrent qu'il suffit que l'ensemble des candidats soit vide sur c niveaux où c et la taille du plus grand cycle de l'automate qui représente l'expression régulière.

2.5.3 Élagage

L'étape d'élagage consiste à éliminer de la collection des motifs candidats ceux qui ne peuvent pas satisfaire les contraintes anti-monotones car ils possèdent un sous motif qui ne la satisfait pas. L'élagage est donc une manière rapide de vérifier qu'un motif ne satisfait pas une contrainte anti-monotone. En fait, l'élagage correspond au test d'une contrainte qui est une relaxation de la contrainte anti-monotone. Celle-ci peut s'exprimer par : un motif satisfait à la contrainte de non élagage si tous ses sous motifs satisfont la contrainte anti-monotone. Il faut remarquer que cette étape n'est pas nécessaire pour assurer la complétude de l'algorithme.

L'intérêt de cette "contrainte" est qu'elle est souvent plus simple à vérifier que la contrainte anti-monotone $\mathcal{C}_{\mathsf{am}}$. En effet, le test de la contrainte anti-monotone implique souvent de calculer une fonction d'évaluation comme la fréquence. Or le calcul de cette fonction demande de faire une passe sur les données, c'est donc une opération très coûteuse. Évidemment, si le test de la contrainte anti-monotone est moins coûteux que celui de l'élagage (si c'est uniquement une contrainte syntaxique par exemple), il peut être préférable de faire le test de cette contrainte avant l'élagage (voir la discussion sur le coût de l'élagage plus loin).

Lorsqu'un algorithme n'utilise que des contraintes anti-monotones, l'élagage se fait simplement en vérifiant que tous les sous motifs d'un motif candidat φ du niveau k sont dans l'ensemble $\mathcal{S}_{k-1}$ des motifs valides trouvés lors de l'examen du niveau $k-1$. Lorsque d'autres contraintes sont utilisées, si un sous motif de φ ne se trouve pas dans $\mathcal{S}_{k-1}$, c'est qu'il ne satisfait pas la conjonction $\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}$ de la contrainte anti-monotone et de la contrainte $\mathcal{C}$ non anti-monotone. Dans ce cas, s'il satisfait la contrainte $\mathcal{C}$ alors il ne satisfait pas $\mathcal{C}_{\mathsf{am}}$ et par conséquent on peut élaguer φ. Par contre, s'il ne satisfait pas $\mathcal{C}$, alors on ne sait pas s'il satisfait ou non $\mathcal{C}_{\mathsf{am}}$ et on ne peut donc pas élaguer φ. L'algorithme d'élagage correspondant est :

élagage($\mathsf{Cand}, \mathcal{S}_k$) /* Cand est l'ensemble des motifs candidats du niveau $k+1$ et $\mathcal{S}_k$ l'ensemble des motifs du niveau k satisfaisant la contrainte $\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}$ */

```
1  pour tout φ ∈ Cand faire
2     pour tout sous motif μ couvert par φ faire
3        si C(μ) = vrai et μ ∉ S_k alors
4           Cand := Cand \ {φ}
5     fait
7  fait
8  retourne Cand
```

Cette fonction est utilisée dans la plupart des algorithmes qui effectuent un élagage. Les algorithmes FP-growth, Spade et c-Spade ne font pas d'élagage : FP-growth car l'algorithme fonctionne en profondeur d'abord et ne garde pas trace des itemsets des niveaux précédents, Spade et c-Spade car l'élagage impose des coûts supplémentaires et n'apporte pas de gain significatif d'après l'auteur.

Remarquons que cette technique d'élagage n'est pas toujours parfaite. Elle peut ne pas élaguer un motif φ alors que l'algorithme a déjà découvert un sous motif μ de φ qui ne satisfait pas $\mathcal{C}_{\mathsf{am}}$. On dira que l'élagage est incomplet lorsqu'il y a des motifs qui pourraient être élagués et qui ne le sont pas.

Exemple 17 *Considérons l'extraction de séquences fréquentes et satisfaisant l'expression régulière* `(ab|abcd)e*`. *Au niveau k, l'algorithme génère les séquences de taille k qui satisfont*

l'expression régulière. Supposons que la séquence `abee` *ne soit pas fréquente et que* `abcde` *le soit. Au niveau 6, l'algorithme génère la séquence* `abcdee`*. Si la fonction d'élagage utilisée est* **élagage***, alors cette séquence ne sera pas élaguée. En effet, la seule sous séquence de* `abcdee` *de taille 5 qui satisfait l'expression régulière est* `abcde` *qui est fréquente. Ainsi, l'élagage est ici incomplet car* `abcdee` *a une sous séquence infréquente,* `abee`*, et pourrait donc être élaguée.*

Si l'algorithme d'extraction, à chaque fois qu'il trouve un motif qui ne satisfait pas une contrainte anti-monotone, le stocke dans un ensemble $\mathcal{I}$ alors on peut utiliser $\mathcal{I}$ pour faire un algorithme d'élagage complet.

élagage_complet1(Cand, $\mathcal{I}$) /* Cand est l'ensemble des motifs candidats du niveau $k+1$ et $\mathcal{I}$ est l'ensemble des motifs trouvés par l'algorithme qui ne satisfont pas $\mathcal{C}_{\mathsf{am}}$. */
pour tout $\varphi \in$ Cand **faire**
si il existe $\mu \in \mathcal{I}$ tel que $\mu \preceq \varphi$ **alors**
Cand := Cand $\setminus \{\varphi\}$
fait
retourne Cand

À notre connaissance, aucun algorithme n'utilise une telle fonction d'élagage. On peut remarquer que celle-ci risque de devenir coûteuse (tant en espace mémoire qu'en temps) si l'ensemble $\mathcal{I}$ devient grand. Une autre fonction d'élagage complète (mais également coûteuse) consisterait à modifier la fonction **élagage** de manière à effectuer le test de la ligne 3 pour tous les sous motifs de φ et non plus seulement ceux qui sont couverts par φ, c'est-à-dire remplacer les lignes 2 à 5 par :

2 **pour tout** sous motif μ de φ **faire**
3 **si** $\mathcal{C}(\mu) = vrai$ **et** $\mu \notin \mathcal{S}_{|\mu|}$ **alors**
4 Cand := Cand $\setminus \{\varphi\}$
5 **fait**

Les algorithmes Spirit utilisent une version améliorée de cet élagage complet dans laquelle ils ne considèrent pas tous les sous motifs de φ mais seulement les sous motifs satisfaisant $\mathcal{C}$ maximaux.

élagage_complet2(Cand, $\bigcup_{i \leq k} \mathcal{S}_i$)
1 **pour tout** $\varphi \in$ Cand **faire**
2 $M := \max_{\preceq} \{\mu \prec \varphi \mid \mathcal{C}(\mu) = vrai\}$
3 **pour tout** $\mu \in M$ **faire**
4 **si** $\mu \notin \mathcal{S}_{|\mu|}$ **alors**
5 Cand := Cand $\setminus \{\varphi\}$
6 **fait**
7 **fait**
8 **retourne** Cand

On remarque que la première fonction d'élagage, **élagage** est une instanciation de cette fonction **élagage_complet2** lorsque la contrainte $\mathcal{C}$ est la contrainte qui est toujours vraie,

i.e., quand on fait l'extraction avec seulement une contrainte anti-monotone. En effet, dans ce cas il est clair que l'ensemble M est exactement l'ensemble des motifs couverts par φ. On montrera dans la section 2.6.2 que c'est encore le cas quand la contrainte $\mathcal{C}$ est monotone.

Montrons que cette fonction d'élagage est bien complète. Supposons que φ est un motif qui peut être élagué. Cela signifie qu'il existe un sous motif μ de φ qui satisfait $\mathcal{C}$ et qui ne satisfait pas $\mathcal{C}_{\mathsf{am}}$. Considérons l'ensemble des motifs satisfaisant $\mathcal{C}$, compris entre μ et φ et différents de φ. Cet ensemble est non vide (il contient μ). Il contient donc au moins un élément maximal. Celui-ci est dans M et ne satisfait pas $\mathcal{C}_{\mathsf{am}}$ (car μ ne satisfait pas $\mathcal{C}_{\mathsf{am}}$) donc il n'est pas dans $\bigcup_{i\leq k} \mathcal{S}_i$. Par conséquent, φ est bien élagué par la fonction **élagage_complet2** et celle-ci est bien complète.

La fonction peut sembler coûteuse, surtout du fait du calcul de l'ensemble M à la ligne 2. Dans le cas de l'algorithme Spirit, la contrainte $\mathcal{C}$ est une expression régulière. Le calcul des séquences maximales qui satisfont $\mathcal{C}$ se fait alors assez bien en utilisant l'automate représentant l'expression régulière. D'après les auteurs, cet élagage est plus rapide que de calculer la fréquence, il est donc bien intéressant.

D'une manière générale, le coût de l'élagage est souvent plus élevé que le coût du test de la satisfaction d'une contrainte syntaxique $\mathcal{C}$. En effet, comme on le voit dans ces fonctions d'élagage, vérifier si un motif φ doit être élagué nécessite souvent de tester $\mathcal{C}$ sur plusieurs sous motifs de φ. Pour cette raison, il vaut mieux tester les contraintes syntaxiques avant l'étape d'élagage. Par contre, le coût de l'étape d'élagage est souvent très inférieur à celui du calcul d'une fonction d'évaluation.

L'étape d'élagage impose aussi que les motifs valides des niveaux précédents soient gardés en mémoire. Ceci n'est pas toujours simple selon les choix d'implémentation. En particulier, les algorithmes qui fonctionnent en profondeur d'abord (c-Spade et FP-growth) n'ont pas d'étape d'élagage.

2.5.4 Conclusion sur l'état de l'art

Nous avons présenté ici des algorithmes d'extraction de motifs satisfaisant une contrainte de la forme $\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}$, où $\mathcal{C}_{\mathsf{am}}$ est une contrainte anti-monotone et $\mathcal{C}$ une contrainte syntaxique. Par rapport à l'algorithme Apriori qui ne peut traiter que des contraintes anti-monotones, l'objectif est ici de pousser également la contrainte $\mathcal{C}$ pour limiter la taille de l'espace de recherche à explorer. En particulier, ces algorithmes cherchent à diminuer la taille de l'ensemble des motifs sur lesquels on calcule les fonctions d'évaluation, comme la fréquence, qui sont très coûteuses (car nécessitant une passe sur les données).

Au lieu de faire une exploration par niveau de tout l'ensemble des motifs, ces algorithmes n'explorent que $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C})$. La manière précise dont les candidats sont générés dépend des propriétés de la contrainte $\mathcal{C}$. Dans le cas où elle est anti-monotone, il suffit d'utiliser l'algorithme Apriori. Dans les autres cas, il faut trouver des méthodes ad-hoc qui assurent la complétude de l'algorithme tout en ne générant pas trop de candidats. Il faut pour cela souvent faire un compromis entre la sélectivité de la contrainte que l'on pousse et les propriétés de l'espace de recherche $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C})$.

Un bon exemple de ce compromis est étudié dans [GRS99] avec les différentes versions de l'algorithme Spirit. Cet algorithme extrait des séquences fréquentes qui doivent satisfaire une expression régulière. Dans les différentes versions, des relaxations de plus en plus sélectives de cette contrainte sont poussées pendant l'exploration. On a donc des contraintes $\mathcal{C}_N$, $\mathcal{C}_L$, $\mathcal{C}_V$ et $\mathcal{C}_R$ qui correspondent respectivement aux algorithmes Spirit(N), Spirit(L), Spirit(V)

et Spirit(R) telles que $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_R) \subseteq \mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_V) \subseteq \mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_L) \subseteq \mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_N)$. Les auteurs montrent qu'en général, l'algorithme le plus efficace est celui qui explore $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_V)$. En effet, cet espace de recherche est plus petit que $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_N)$ et $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_L)$ et plus simple à explorer que $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_R)$ pour lequel il n'y a pas de bonne fonction de génération. En fait dans ce dernier cas, l'étape de génération des candidats devient plus coûteuse que la passe sur les données et le calcul des fréquences. L'exploration de $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C}_R)$ n'est valable que lorsque la contrainte $\mathcal{C}_R$ est très restrictive.

Un autre problème apparaît lorsqu'on pousse des contraintes non anti-monotones. Il est possible qu'il existe des motifs hors de $\mathsf{Th}(bd, \mathcal{L}, \mathcal{C})$ qui permettent d'élaguer une large partie de l'espace de recherche car ils ne satisfont pas $\mathcal{C}_{\mathsf{am}}$. Cependant, si la contrainte $\mathcal{C}$ est poussée, ces motifs ne seront pas découverts et donc le treillis ne sera pas élagué.

Exemple 18 *Soit la contrainte $\mathcal{C}(S) = (|S| \geq 10) \wedge \mathcal{C}_{\gamma\text{-}\mathsf{freq}}(S)$, i.e., S doit contenir au moins 10 items et S doit être fréquent. Si la contrainte monotone $|S| \geq 10$ est poussée jusque dans l'étape de génération des candidats, aucun candidat de taille inférieure à 10 ne sera généré. Tous les candidats de taille 10 doivent alors être générés puis leurs fréquences doivent être calculées lors d'une lecture de la base de données (et éventuellement poursuivre pour des itemsets de tailles supérieures). Puisqu'aucun élagage n'aura été réalisé, on aura donc C_n^{10} candidats pour la première phase de comptage des fréquences (n désigne le nombre d'items). Dès que n sera grand, il sera clairement impossible d'énumérer ces candidats et donc, le problème sera intraitable quelque soit le seuil de fréquence. Dans ce cas, une approche "générer et tester" calculant tous les itemsets fréquents (pour ne finalement retenir que ceux dont la taille est supérieure ou égale à 10) demandera plusieurs passes de lectures sur les données mais pourra être faisable grâce aux élagages apportés par la contrainte de fréquence minimale. On peut d'ailleurs, dans ce cas, prendre des seuils de fréquence plus élevés pour assurer la faisabilité du calcul.*

2.6 Extraction d'itemsets sous contraintes anti-monotone et monotone

Nous avons vu dans la section 2.1.1 comment utiliser activement les contraintes anti-monotones pour élaguer l'espace de recherche. Ensuite, dans l'état de l'art nous avons présenté de nombreux travaux qui utilisent d'autres types de contraintes. Ici, nous allons présenter un algorithme qui permet d'extraire des itemsets satisfaisant la conjonction d'une contrainte anti-monotone et d'une contrainte monotone. Cet algorithme sera utilisé dans le chapitre suivant pour l'extraction de représentations condensées sous contraintes. Plus formellement, on s'intéresse à l'extraction de $\mathsf{Res}(bd, 2^{\mathsf{Items}}, \mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}}, e, \mathcal{D})$. Comme la base de données bd ne changera pas et que l'on s'intéressera toujours à des itemsets, on ne précisera plus bd et 2^{Items} dans les notations.

2.6.1 Caractérisation par les frontières

Nous avons déjà vu que les théories $\mathcal{T}_m = \mathsf{Th}(\mathcal{C}_{\mathsf{m}})$ et $\mathcal{T}_{am} = \mathsf{Th}(\mathcal{C}_{\mathsf{am}})$ sont caractérisées par des frontières (propositions 3 et 6) : $\mathcal{T}_m$ est l'ensemble des sur ensembles des itemsets de $\mathcal{B}d^-(\mathsf{Th}(\neg\mathcal{C}_{\mathsf{m}}))$ et $\mathcal{T}_{am}$ l'ensemble des sous ensembles des itemsets de $\mathcal{B}d^+(\mathcal{T}_{am})$. La théorie $\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}})$ qui est l'intersection de $\mathcal{T}_m$ et $\mathcal{T}_{am}$ est donc l'ensemble des itemsets qui sont situés entre ces deux frontières.

On peut en fait restreindre les ensembles $\mathcal{B}d^+(\mathsf{Th}(\mathcal{C}_{\mathsf{am}}))$ et $\mathcal{B}d^-(\mathsf{Th}(\neg\mathcal{C}_{\mathsf{m}}))$. En effet, certains itemsets de $\mathcal{B}d^+(\mathsf{Th}(\mathcal{C}_{\mathsf{am}}))$ ne sont sur ensemble d'aucun des éléments de $\mathcal{B}d^-(\mathsf{Th}(\neg\mathcal{C}_{\mathsf{m}}))$ et réciproquement certains itemsets de $\mathcal{B}d^-(\mathsf{Th}(\neg\mathcal{C}_{\mathsf{m}}))$ ne sont inclus dans aucun itemset de $\mathcal{B}d^+(\mathsf{Th}(\mathcal{C}_{\mathsf{am}}))$.

Définition 22 *Soit $\mathcal{C}_{\mathsf{am}}$ une contrainte anti-monotone et $\mathcal{C}_{\mathsf{m}}$ une contrainte monotone, les ensembles $\min_{\subseteq}(\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}}))$ et $\max_{\subseteq}(\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}}))$ contiennent respectivement les éléments les plus généraux et les plus spécifiques de $\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}})$. Ils sont notés $\mathcal{G}(\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}}))$ et $\mathcal{S}(\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}}))$ (pour Général et Spécifique).*

Les notations $\mathcal{S}$ et $\mathcal{G}$ pour désigner les éléments les plus spécifiques et les plus généraux viennent du domaine de l'apprentissage avec l'algorithme classique des espaces de version (Mitchell [Mit80]).

Proposition 8 *Soit $\mathcal{C}_{\mathsf{am}}$ une contrainte anti-monotone et $\mathcal{C}_{\mathsf{m}}$ une contrainte monotone, alors la théorie $\mathcal{T} = \mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}})$ est exactement l'ensemble des itemsets S tels qu'il existe $V \in \mathcal{G}(\mathcal{T})$ et $W \in \mathcal{S}(\mathcal{T})$ tels que $V \subseteq S \subseteq W$.*
De plus, $\mathcal{G}(\mathcal{T}) \subseteq \mathcal{B}d^-(\mathsf{Th}(\neg\mathcal{C}_{\mathsf{m}}))$ et $\mathcal{S}(\mathcal{T}) \subseteq \mathcal{B}d^+(\mathsf{Th}(\mathcal{C}_{\mathsf{am}}))$.

Exemple 19 *Si on considère la conjonction des contraintes $\mathcal{C}_{2\text{-}\mathsf{freq}}$ et $\mathcal{C}_{\mathsf{m}}(S) = (S \cap \mathtt{AB} \neq \emptyset) \vee (\mathtt{DE} \subseteq S)$ dans la base de données de la figure 1.1 alors*

$$\mathcal{S}(\mathsf{Th}(\mathcal{C}_{2\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{m}})) = \{\mathtt{ABCD}, \mathtt{ABE}\} \subseteq \mathcal{B}d^+(\mathsf{Th}(\mathcal{C}_{2\text{-}\mathsf{freq}})) = \{\mathtt{ABCD}, \mathtt{ABE}, \mathtt{CE}\},$$
$$\mathcal{G}(\mathsf{Th}(\mathcal{C}_{2\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{m}})) = \{\mathtt{A}, \mathtt{B}\} \subseteq \mathcal{B}d^-(\mathsf{Th}(\neg\mathcal{C}_{\mathsf{m}})) = \{\mathtt{A}, \mathtt{B}, \mathtt{DE}\}.$$

La figure 2.3 montre la théorie $\mathsf{Th}(\mathcal{C}_{2\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{m}})$ avec les frontières et les ensembles $\mathcal{S}(\mathsf{Th}(\mathcal{C}_{2\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{m}}))$ et $\mathcal{G}(\mathsf{Th}(\mathcal{C}_{2\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{m}}))$.

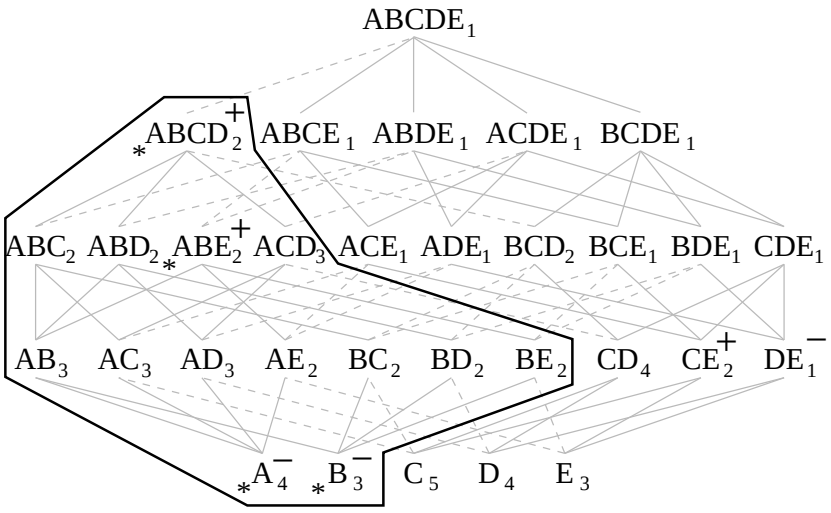


FIG. 2.3 – La théorie $\mathsf{Th}(\mathcal{C}_{2\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{m}})$ ($\mathcal{C}_{\mathsf{m}}$ est défini dans l'exemple 19), les itemsets des frontières $\mathcal{B}d^+(\mathsf{Th}(\mathcal{C}_{2\text{-}\mathsf{freq}}))$ et $\mathcal{B}d^-(\mathsf{Th}(\neg\mathcal{C}_{\mathsf{m}}))$ sont représentés avec les exposants $+$ et $-$ et les ensembles de $\mathcal{S}(\mathcal{T})$ et $\mathcal{G}(\mathcal{T})$ sont indicés par $*$.

Cet ensemble des motifs valides par rapport à $\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}}$ a une propriété de convexité qui découle de la proposition ci-dessus. Si V et W sont deux itemsets de $\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}})$ alors tout

itemset S tel que $V \subseteq S \subseteq W$ appartient aussi à $\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}})$. Un sous ensemble de 2^{Items} qui a cette propriété de convexité est appelé un *espace des versions* [Mit80].

De Raedt étudie dans [dR02b] ces espaces des versions sur les itemsets. Cet article aborde en particulier le problème de la combinaison d'espaces des versions (intersection et réunion) et discute de l'intérêt des frontières $\mathcal{S}(\mathcal{T})$ et $\mathcal{G}(\mathcal{T})$ en temps que représentations condensées d'une théorie $\mathcal{T}$.

2.6.2 Algorithme générique

On considère que la contrainte $\mathcal{C}_{\mathsf{am}}$ est la conjonction d'une contrainte syntaxique et d'une contrainte non syntaxique[2]. Le problème devient donc le calcul de $\mathsf{Res}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}}, e, \mathcal{D})$ où $\mathcal{C}_{\mathsf{am}_s}$ est une contrainte syntaxique anti-monotone. La fonction d'évaluation e est souvent la fréquence, mais cela n'est pas obligatoire ; dans le chapitre suivant nous utiliserons aussi la fermeture. L'algorithme que nous présentons généralise les algorithmes que nous avons vus dans l'état de l'art. Il nous servira dans le prochain chapitre pour l'extraction de représentations condensées sous contraintes.

Cet algorithme effectue un parcours niveau par niveau de l'espace de recherche. Seulement, au lieu de parcourir 2^{Items} comme Apriori, cet algorithme effectue son parcours dans $\mathsf{Th}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_{\mathsf{m}})$. Il profite donc de la contrainte monotone et de la contrainte syntaxique pour restreindre l'espace de recherche. De plus, comme Apriori, l'algorithme utilise la contrainte anti-monotone pour élaguer l'espace de recherche. Nous faisons une différence entre la contrainte anti-monotone syntaxique et la contrainte anti-monotone non syntaxique car elles ne sont pas utilisées par l'algorithme au même moment : la contrainte syntaxique est utilisée lors de la génération des candidats alors que la contrainte non syntaxique est utilisée après la passe sur les données et le calcul de la fonction d'évaluation e.

Chaque itemset S est représenté par un enregistrement avec deux champs, un champ $S.\mathsf{item}$ qui contient la liste des items de S et un champ $S.e$ qui contient la valeur de la fonction d'évaluation une fois qu'elle a été calculée.

Algorithme 4 (algorithme générique)

Entrée : Une contrainte de la forme $\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}}$
Sortie : $\mathsf{Res}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}}, e, \mathcal{D})$

1 $k := n\_min$
2 $\mathcal{F}_1 := \{i \in \mathsf{Items} \mid \mathcal{C}_{\mathsf{am}_s}(i) \wedge \mathcal{C}_{\mathsf{am}}(i)\}$
3 $\mathsf{Cand} := \mathbf{gen\_cand}(k, \emptyset)$
4 $\mathsf{Cand} := \{S \in \mathsf{Cand} \mid \mathcal{C}_{\mathsf{am}_s}(S) = vrai\}$ /* test de $\mathcal{C}_{\mathsf{am}_s}$ */
5 **tant que** $\mathsf{Cand} \neq \emptyset$ **ou** $k \leq n\_max$ **faire**
6 $\quad \mathsf{Cand} := \mathbf{passe\_bd}(\mathsf{Cand}, bd)$ /* calcul de $S.e$ pour tout $S \in \mathsf{Cand}$ */
7 $\quad \mathcal{S}_k := \{S \in \mathsf{Cand} \mid \mathcal{C}_{\mathsf{am}}(S) = vrai\}$ /* test de $\mathcal{C}_{\mathsf{am}}$ */
8 $\quad \mathsf{Cand} := \mathbf{gen\_cand}(k+1, \mathcal{S}_k)$ /* génération des candidats */
9 $\quad \mathsf{Cand} := \{S \in \mathsf{Cand} \mid \mathcal{C}_{\mathsf{am}_s}(S) = vrai\}$ /* test de $\mathcal{C}_{\mathsf{am}_s}$ */
10 $\quad \mathsf{Cand} := \mathbf{élagage}(\mathsf{Cand}, \mathcal{S}_k)$ /* élagage */
11 $\quad k := k + 1$
12 **fait**

[2] La contrainte monotone peut elle aussi avoir une composante syntaxique et une composante non syntaxique mais il n'est pas utile de les différencier.

13 **retourne** $\bigcup_{0\leq i\leq k} \mathcal{S}_i$

Cet algorithme ne commence pas son exploration au niveau 0 mais au niveau n_min qui correspond à la taille des plus petits itemsets qui satisfont la contrainte monotone $\mathcal{C}_\mathsf{m}$. Si on initialisait k à 0, l'algorithme fonctionnerait mais il ferait plusieurs parcours de la boucle **tant que** avec un ensemble de candidats vide. L'exploration se fait en partant des itemsets minimaux de l'espace $\mathsf{Th}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_\mathsf{m})$. Comme ils peuvent être de taille variable, on ne peut pas stopper l'exploration dès que Cand est vide (comme dans Apriori). En effet, si on stoppe l'exploration au niveau k car Cand est vide, les itemsets minimaux de $\mathsf{Th}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_\mathsf{m})$ de taille supérieure à k n'auront jamais été considérés. Il faut donc continuer l'exploration jusqu'à un niveau où on est sûr d'avoir considéré tous les itemsets minimaux de $\mathsf{Th}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_\mathsf{m})$. Pour cela, on fixe le niveau n_max, qui intervient dans le test d'arrêt de l'algorithme, à la taille maximale des itemsets de $\min(\mathsf{Th}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_\mathsf{m})) = \mathcal{B}d^-(\mathsf{Th}(\neg\mathcal{C}_\mathsf{m})) \cap \mathsf{Th}(\mathcal{C}_{\mathsf{am}_s})$. Par exemple, si $\mathcal{C}_\mathsf{m}(S) = \mathsf{B} \in S$ et qu'il n'y a pas de contrainte anti-monotone syntaxique, alors $\mathcal{B}d^-(\mathsf{Th}(\neg\mathcal{C}_\mathsf{m})) = \{\mathsf{B}\}$ et $n\_min = n\_max = 1$.

Dans un algorithme niveau par niveau, lorsque la contrainte est anti-monotone, les candidats du niveau k sont les motifs de taille k de la frontière négative de $\bigcup_{0\leq i<k} \mathcal{S}_i$ ($\mathcal{S}_i$ est la collection des itemsets de taille i qui satisfont les contraintes) comme nous l'avons vu dans l'algorithme Guess and correct (algorithme 1 page 34). Autrement dit, ce sont les motifs minimaux de la partie encore inexplorée de l'espace de recherche. Ici, on utilise le même principe. Les candidats sont les itemsets minimaux de la partie inexplorée de l'espace de recherche. Comme l'espace de recherche est ici $\mathsf{Th}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_\mathsf{m})$, les candidats pour le niveau k sont les itemsets appartenant à :

$$\min_\subseteq \left(\mathsf{Th}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_\mathsf{m}) \setminus \bigcup_{0\leq i<k} \mathcal{S}_i \right).$$

Comme dans l'algorithme Apriori, nous générons les candidats en plusieurs étapes (lignes 8, 9 et 10). Dans une première étape, l'algorithme génère un sur ensemble de l'ensemble des candidats (fonction **gen_cand**). Ensuite, les candidats qui ne satisfont pas la contrainte syntaxique sont éliminés (ligne 9). Enfin, un élagage est fait (ligne 10) pour éliminer les candidats qui ont un sous ensemble qui ne satisfait pas la contrainte anti-monotone.

L'ordre dans lequel sont effectuées les phases de test de la contrainte syntaxique et l'élagage n'a pas d'influence sur la correction ou la complétude de l'algorithme. Nous effectuons l'élagage en dernier car c'est une étape généralement plus coûteuse que de tester une contrainte syntaxique, mais l'algorithme fonctionnerait aussi bien en échangeant les lignes 9 et 10.

Détaillons maintenant la fonction **gen_cand**. Celle ci doit générer un sur ensemble des itemsets minimaux de l'espace de recherche qui n'ont pas encore été considérés. Ces candidats sont de deux types :

- les itemsets qui ont un sous ensemble dans $\mathcal{S}_k$;
- et ceux qui ont tous leurs sous ensembles hors de l'espace de recherche, autrement dit, ceux dont aucun sous ensemble ne satisfait $\mathcal{C}_\mathsf{m}$.

Les candidats du premier type peuvent être générés en étendant un itemset valide du niveau k avec un item de $\mathcal{F}_1$ (l'ensemble $\mathcal{F}_1$ contient les items qui satisfont $\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_\mathsf{am}$; il est calculé ligne 2 de l'algorithme générique). Ceux du second type doivent être générés séparément. Nous notons Items_k l'ensemble des itemsets de taille k.

gen_cand$(k, \mathcal{S})$ /* génération des candidats du niveau k */
1 $C := \emptyset$
2 **pour tout** $S \in \mathcal{S}$ **faire** /* génération des itemsets du premier type */
3 **pour tout** $i \in \mathcal{F}_1$ **faire**
4 $C := C \cup \{S.\mathsf{item} \cup \{i\}\}$
5 **fait**
6 **fait**
7 $C := C \cup \left(\mathcal{B}d^-(\mathsf{Th}(\neg\mathcal{C}_\mathsf{m})) \cap \mathsf{Items}_k\right)$ /* génération des itemsets du second type */
8 **retourne** $\{(S, 0) \mid S \in C\}$

Nous verrons dans la section suivante comment calculer les itemsets appartenant à la frontière $\mathcal{B}d^-(\mathsf{Th}(\neg\mathcal{C}_\mathsf{m}))$. Nous montrons maintenant que cette fonction de génération est bien complète, i.e., elle ne manque aucun itemset satisfaisant aux contraintes.

Proposition 9 *La fonction de génération des candidats* **gen_cand** *est complète, i.e., l'appel* **gen_cand**$(k+1, \mathcal{S}_k)$ *où* $\mathcal{S}_k = \mathsf{Th}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_\mathsf{am} \wedge \mathcal{C}_\mathsf{m}) \cap \mathsf{Items}_k$ *retourne un sur ensemble de* $\mathsf{Th}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_\mathsf{am} \wedge \mathcal{C}_\mathsf{m}) \cap \mathsf{Items}_{k+1}$.

Preuve : Soit S un itemset de taille $k+1$ appartenant à l'espace des versions $\mathcal{T} = \mathsf{Th}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_\mathsf{am} \wedge \mathcal{C}_\mathsf{m})$. D'après la proposition 8 page 69, il existe $V \in \min \subseteq (\mathcal{T})$ tel que $V \subseteq S$. Il y a alors deux possibilités.
- Soit $V \neq S$, dans ce cas il existe un sous ensemble de S dans $\mathcal{T} \cap \mathsf{Items}_k$ et donc S est généré dans les lignes 2 à 6.
- Soit $V = S$, donc $S \in \min \subseteq (\mathcal{T}) \subseteq \mathcal{B}d^-(\mathsf{Th}(\neg\mathcal{C}_\mathsf{m}))$ et comme $|S| = k+1$, il est bien généré à la ligne 7.

□

On peut diminuer le nombre de candidats générés par cette fonction en utilisant les techniques proposées dans les algorithmes Reorder ou CAP. L'idée est de remplacer les lignes 2 à 6 par une fusion d'itemsets partageant $k-2$ items. On peut encore diminuer le nombre de candidats en imposant que les itemsets que l'on fusionne aient des préfixes de taille $k-2$ égaux lorsque les items sont classés dans un ordre donné (quand cet ordre est l'ordre alphabétique on retrouve la génération d'Apriori). Cet ordre dépend évidemment de la contrainte monotone considérée. Par exemple, si la contrainte monotone est $\mathtt{B} \in S$ alors on ordonnera les items en plaçant B en premier. Ainsi, à partir du niveau 3, on peut générer les candidats par fusion d'itemsets du niveau précédent qui ont des préfixes de taille $k-2$ égaux.

En ce qui concerne l'élagage, nous utilisons la fonction d'élagage **élagage** que nous avons déjà présentée page 65 et que nous rappelons ici (dans le cas des itemsets avec une contrainte monotone).

élagage$(\mathsf{Cand}, \mathcal{S}_k)$ /* Cand contient les candidats du niveau $k+1$ et $\mathcal{S}_k$ l'ensemble des itemsets du niveau k satisfaisant la contrainte $\mathcal{C}_\mathsf{am} \wedge \mathcal{C}_\mathsf{m}$ */
1 **pour tout** $S \in \mathsf{Cand}$ **faire**
2 **pour tout** sous ensemble T de $S.\mathsf{item}$ de taille k **faire**
3 **si** $\mathcal{C}_\mathsf{m}(T) = vrai$ **et** $T \notin \mathcal{S}_k.\mathsf{item}$ **alors**
4 $\mathsf{Cand} := \mathsf{Cand} \setminus \{S\}$
5 **fait**
7 **fait**
8 **retourne** Cand

Nous allons montrer que cette fonction d'élagage est complète (quand on pousse seulement des contraintes monotones) en montrant qu'elle se réduit à la fonction **élagage_complet2** de la page 66.

Proposition 10 *La fonction* **élagage** *est complète lorsqu'on ne pousse que des contraintes monotones et anti-monotones.*

Preuve : En fait, il suffit de montrer que étant donné un itemset S de taille k généré par l'algorithme alors, l'ensemble $M = \max_{\subseteq} \{T \subseteq S \mid \mathcal{C}_{\mathsf{m}}(T) = vrai\}$ (utilisé dans la fonction **élagage_complet2**) ne contient que des itemsets de taille $k-1$. Soit $T \in M$, si $|T| < k-1$ alors il existe un itemset de taille $k-1$ inclus dans S et contenant T. Comme $\mathcal{C}_{\mathsf{m}}$ est monotone et que T la satisfait, cet itemset la satisfait aussi. Donc T n'est pas un élément maximal et n'appartient donc pas à M ce qui est absurde. Donc finalement, M ne contient que des sous ensemble de S de taille $k-1$ et on constate facilement que dans ce cas les fonctions **élagage** et **élagage_complet2** sont les mêmes. Comme cette dernière est complète, il en est de même de **élagage**. □

2.6.3 Calcul de la frontière inférieure

L'algorithme de la section précédente utilise la frontière de la contrainte monotone pour générer les candidats. Cette génération se fait différemment selon que la contrainte monotone est ou non syntaxique.

Dans le cas d'une contrainte syntaxique, le calcul de la frontière peut se faire directement sans accéder à la base de données. On peut pour cela utiliser des techniques semblables à celles qui sont utilisées dans les algorithmes que nous avons vus dans l'état de l'art.

Dans l'algorithme Direct [SVA97], la contrainte syntaxique est sous forme normale disjonctive en utilisant les contraintes élémentaires $\mathcal{C}_i$ et $\mathcal{C}_{\neg i}$. Nous avons vu que cette contrainte peut s'écrire comme une disjonction de contraintes de la forme $V_i \subseteq S \subseteq W_i$. Si la contrainte est monotone, alors ces termes seront en fait du type $V_i \subseteq S$. Dans ce cas, la frontière est exactement la collection $\min_{\subseteq} V_i$. Seulement, comme nous l'avons déjà souligné, la réécriture de la contrainte sous cette forme est souvent coûteuse (à moins, évidemment, qu'elle ne soit déjà sous cette forme).

L'algorithme CAP [NLHP98] extrait des itemsets sous contraintes anti-monotones et succinctes. Les auteurs utilisent des MGF (*Member Generating Function*) pour générer les itemsets qui satisfont une contrainte succincte. Plus précisément, étant donné une contrainte succincte, ils montrent comment calculer un entier n et des ensembles d'items A_i tels que les itemsets satisfaisant cette contrainte succincte peuvent s'écrire sous la forme $X_1 \cup X_2 \cup \cdots \cup X_n$ avec $X_i \subseteq A_i$ et $X_i \neq \emptyset$ pour $1 \leq i < n$. Il est donc facile de calculer la frontière à partir des A_i. En effet, c'est tout simplement la collection $\min_{\subseteq} \{\{i_1, \ldots, i_{n-1}\} \mid i_j \in A_j,\ 1 \leq j < n\}$.

L'article [dRK01] utilise une technique empruntée à la programmation logique inductive pour générer la frontière inférieure (qui est dans cet article notée G). Les contraintes monotones syntaxiques utilisées sont des contraintes de la forme $f \leq p$, $p \leq f$, $\neg(f \leq p)$ et $\neg(p \leq f)$. Les auteurs utilisent l'algorithme de Mitchell [Mit80] étendu par Mellish [Mel92] pour générer la frontière. Bien que dans leur article les auteurs traitent des fragments moléculaires, les algorithmes de Mitchell et Mellish sont généraux et peuvent être utilisés pour les itemsets. Ils permettent de traiter des conjonctions de contraintes monotones du type $V \subseteq S$ et $\neg(S \subseteq V)$ où V est un itemset fixé.

Dans le cas où la contrainte n'est pas syntaxique, il faut utiliser une autre méthode pour calculer la frontière. On peut utiliser par exemple une méthode de calcul de frontière comme celles qui sont expliquées dans la section 2.2 page 49 (échantillonnage puis Guess and correct, ou des algorithmes tels que Max-miner) en exploitant la dualité entre les contraintes monotones et anti-monotones. Ces algorithmes permettent en effet de calculer la frontière positive $\mathcal{B}d^+(\mathsf{Th}(\mathcal{C}_{\mathsf{am}}))$ de l'ensemble des itemsets satisfaisant une contrainte anti-monotone $\mathcal{C}_{\mathsf{am}}$ (dans notre cas, la contrainte anti-monotone sera $\neg\mathcal{C}_{\mathsf{m}}$). Or, nous sommes intéressés par la frontière négative de $\mathsf{Th}(\neg\mathcal{C}_{\mathsf{m}})$. A priori, passer de la frontière positive à la frontière négative est un problème difficile [MT97]. Cependant, d'après le théorème 1 page 36, un algorithme qui calcule $\mathsf{Th}(\mathcal{C}_{\mathsf{am}})$ doit nécessairement parcourir les deux frontières. Les algorithmes tels que Max-miner doivent donc parcourir la frontière négative même s'ils ne la retournent pas. Il suffit donc de les modifier légèrement pour qu'ils renvoient cette frontière négative $\mathcal{B}d^-(\mathsf{Th}(\neg\mathcal{C}_{\mathsf{m}}))$.

2.6.4 Utiliser les contraintes monotones sans élaguer

Dans le cas où le calcul de la frontière inférieure d'une contrainte monotone non syntaxique n'est pas fait, on peut tout de même pousser celle-ci dans l'algorithme en utilisant une méthode similaire à celle de [PH00]. Le test de cette contrainte monotone est fait après la passe sur les données (car c'est une contrainte non syntaxique). Les itemsets qui ne la satisfont pas sont marqués mais ils ne sont pas éliminés immédiatement (ils doivent être utilisés pour l'étape de génération du niveau suivant). L'intérêt du marquage est que lors de l'itération suivante, il n'est pas nécessaire de tester la contrainte monotone sur les itemsets qui ont un sous ensemble non marqué. En effet, si le sous ensemble est non marqué, cela signifie qu'il satisfait la contrainte et comme celle-ci est monotone, tous ses sur ensembles la satisfont également.

Cette technique permet d'économiser le test de la contrainte monotone sur de nombreux itemsets mais cela ne permet pas d'élagage du treillis, le gain obtenu dépend donc entièrement du coût d'évaluation de cette contrainte.

2.6.5 Disjonctions

Avec notre algorithme générique, on peut utiliser des conjonctions de contraintes monotones et anti-monotones. Par contre, il ne permet pas de traiter le cas des disjonctions, i.e., si la contrainte s'écrit $(\mathcal{C}_{\mathsf{am}_1} \wedge \mathcal{C}_{\mathsf{m}_1}) \vee \cdots \vee (\mathcal{C}_{\mathsf{am}_n} \wedge \mathcal{C}_{\mathsf{m}_n})$. Dans ce cas, l'ensemble des motifs valides pour chacune des disjonctions est un espace des versions. On peut donc faire l'extraction en plusieurs étapes, une par disjonction, en utilisant notre algorithme générique. L'inconvénient est que certains itemsets (qui satisfont plusieurs disjonctions) vont être examinés plusieurs fois ce qui n'est pas souhaitable. Pour améliorer cette situation, nous faisons plusieurs propositions :

- On peut explorer les espaces des versions les uns après les autres en utilisant un cache. Dans ce cas, les itemsets présents dans plusieurs espaces des versions ne seront calculés qu'une fois. Ensuite, il suffira de les récupérer dans le cache. C'est une solution qui a l'avantage d'être simple et assez efficace. Nous présentons l'extraction avec cache dans le chapitre 4.
- On peut récrire la contrainte sous une forme où les espaces des versions sont distincts. Dans ce cas, chaque itemset est bien calculé une seule fois. Cette méthode comporte deux inconvénients : premièrement, pour pouvoir s'assurer que les espaces des versions sont bien disjoints, il est nécessaire que les contraintes soient syntaxiques, ensuite cette transformation peut être coûteuse.

– Finalement, on peut modifier l'algorithme pour pouvoir explorer simultanément plusieurs espaces des versions.

La fonction de génération des candidats est assez simple à modifier. Dans la seconde partie de celle-ci, i.e., la ligne 7 de la fonction **gen_cand**, il faut rajouter les itemsets appartenant aux frontières négatives pour toutes les contraintes monotones. Il faut donc remplacer cette ligne par :

$$7 \quad C := C \cup \left(\bigcup_{1 \leq i \leq n} \mathcal{B}d^-(\mathsf{Th}(\neg \mathcal{C}_{\mathsf{m}_i})) \cap \mathsf{Items}_k \right)$$

Il est aussi nécessaire de modifier l'élagage. Dans ce cas, un itemset candidat ne peut être élagué que si on n'est sûr qu'il n'est dans *aucun* des espaces des versions. Un itemset S de taille k n'est pas dans l'espace des versions correspondant à la contrainte $\mathcal{C}_{\mathsf{am}_i} \wedge \mathcal{C}_{\mathsf{m}_i}$

– soit si $\mathcal{C}_{\mathsf{m}_i}(S)$ est faux ;
– soit s'il existe $T \subset S$ de taille $k-1$ tel que $\mathcal{C}_{\mathsf{m}_i}(T)$ soit vrai et tel que T ne soit pas dans $\mathcal{S}_{k-1}$. Dans ce cas cela signifie que T ne satisfait pas $\mathcal{C}_{\mathsf{am}_i}$ et donc S non plus.

On obtient donc une nouvelle fonction d'élagage :

élagage_disjonctions($\mathsf{Cand}, \mathcal{S}_k$) /* Cand est l'ensemble des motifs candidats du niveau $k+1$ et $\mathcal{S}_k$ l'ensemble des motifs du niveau k satisfaisant la contrainte $\bigvee_{1 \leq i \leq n} \mathcal{C}_{\mathsf{am}_i} \wedge \mathcal{C}_{\mathsf{m}_i}$ */
pour tout $S \in \mathsf{Cand}$ **faire**
$elague := vrai$
pour tout i tel que $\mathcal{C}_{\mathsf{m}_i}(S) = vrai$ **faire** /* détermine si on peut élaguer S dans l'espace des versions i */
$elague_i := faux$
pour tout sous ensemble T de S de taille k **faire**
si $\mathcal{C}_{\mathsf{m}_i}(T) = vrai$ **et** $T \notin \mathcal{S}_k$ **alors** $elague_i := vrai$
fait
$elague := elague \wedge elague_i$ /* pour que S puisse être élagué, il faut qu'il puisse l'être dans chaque espace des versions */
fait
si $elague$ **alors** $\mathsf{Cand} := \mathsf{Cand} \setminus \{S\}$
fait
retourne Cand

2.6.6 Contraintes faiblement anti-monotones

Le fait de pousser ce type de contrainte fait perdre la caractérisation des itemsets valides par les frontières (proposition 8 page 69). Il est cependant possible de pousser une contrainte faiblement anti-monotone (comme par exemple une contrainte convertible anti-monotone) dans notre algorithme pour extraire des itemsets satisfaisant une contrainte $\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{am}_f}$ où $\mathcal{C}_{\mathsf{am}_f}$ est la contrainte faiblement anti-monotone. Notre fonction de génération des candidats est bien complète (voir la définition de l'anti-monotonicité faible 21 page 62). Quant à la fonction d'élagage, il faut utiliser la fonction **élagage** page 65 en remplaçant la contrainte $\mathcal{C}$ par $\mathcal{C}_{\mathsf{am}_f}$. Il faut noter que comme la conjonction de deux contraintes faiblement anti-monotones n'est pas forcément anti-monotone, il n'est pas toujours possible de pousser plusieurs de ces contraintes (il y a la même limitation dans FP-growth, une conjonction de contraintes convertibles anti-monotones n'est pas forcément convertible anti-monotone).

En revanche, il n'est pas toujours possible de combiner une contrainte faiblement anti-monotone et une contrainte monotone. En effet, il est possible que le chemin (proposi-

tion 7 page 63) qui permet de générer un itemset satisfaisant la contrainte faiblement anti-monotone sorte de $\mathsf{Th}(\mathcal{C}_{\mathsf{m}})$, dans ce cas l'itemset n'est plus accessible.

Exemple 20 *Considérons la conjonction d'une contrainte monotone et d'une contrainte faiblement anti-monotone* $\mathcal{C}(S) = \mathtt{A} \in S \wedge (MOY(S) < 10)$ *et les items* $\mathtt{A}, \mathtt{B}, \mathtt{C}$ *tels que* $v(\mathtt{A}) = 20$, $v(\mathtt{B}) = 1$ *et* $v(\mathtt{C}) = 1$. *L'itemset* ABC *satisfait la contrainte* $\mathcal{C}$. *Par contre aucun de ses sous ensembles appartenant à* $\mathsf{Th}(\mathtt{A} \in S)$ *ne la satisfait. En effet,* $MOY(\mathtt{AB}) = MOY(\mathtt{AC}) = 10.5 > 10$. *Par conséquent,* ABC *ne sera pas généré par notre algorithme.*

2.7 Différentes stratégies d'évaluation de requêtes inductives

2.7.1 Des contraintes de règles aux contraintes d'itemsets

Il faut d'abord transformer les contraintes de règles en contraintes sur les itemsets. C'est une tâche difficile et peu de travaux théoriques ont été réalisés sur ce sujet. Généralement, on utilise des transformations ad-hoc. On notera d'ailleurs que toutes les contraintes ne peuvent pas être poussées, notamment la contrainte de confiance [LNHP99].

Exemple 21 *Les contraintes utilisées dans la requête MINE RULE de la figure 1.4 page 26 sont :* $\mathcal{C}_{\gamma\text{-freq}}$, $\mathcal{C}_{\theta\text{-conf}}$, $\mathcal{C}_a(X \Rightarrow Y) \equiv \forall y \in Y, y.price > 100$ *et* $\mathcal{C}_b(X \Rightarrow Y) \equiv |(X \cup Y) \cap \{\mathtt{A}, \mathtt{B}\}| \leq 1$. $\mathcal{C}_b$ *peut être récrit comme la contrainte d'itemset* $\mathcal{C}_b(S) \equiv |S \cap \{\mathtt{A}, \mathtt{B}\}| \leq 1$. *D'autre part, comme les règles extraites avec cette requête ne peuvent pas avoir une tête vide,* $\mathcal{C}_a(X \Rightarrow Y) \equiv (\forall y \in Y, y.price > 100) \wedge \mathcal{C}'_a(X \cup Y)$ *où* $\mathcal{C}'_a(S) \equiv |S \cap \{I \in \mathsf{Items}, I.price > 100\}| \geq 1$ *est une contrainte d'itemset. Finalement, on peut pousser la contrainte d'itemset* $\mathcal{C}_{\gamma\text{-freq}} \wedge \mathcal{C}_b \wedge \mathcal{C}'_a$. *Les autres contraintes,* $\mathcal{C}_{\theta\text{-conf}}$ *et* $\mathcal{C}_a$, *devront être testées dans la phase de génération des règles.*

Pour transformer une contrainte portant sur les règles en une contrainte sur les itemsets, nous considérons, comme [GdB00, GdB99], qu'il faut extraire tous les itemsets S tels qu'il existe une règle $X \Rightarrow (S \setminus X)$, $X \subseteq S$, satisfaisant la contrainte portant sur les règles. Cependant, s'il faut calculer certaines fonctions d'évaluation sur les règles comme la confiance, il faut aussi connaître la fréquence de X en plus de celle de S. Pour cela, il suffit de faire une dernière passe sur les données pour calculer les fréquences de ces itemsets [GdB00, GdB99].

2.7.2 Pousser les contraintes anti-monotones

La méthode la plus simple consiste à n'utiliser que les contraintes anti-monotones pendant l'extraction et à repousser l'évaluation des autres contraintes pendant une phase ultérieure. Cette technique est simple à mettre en oeuvre, on peut utiliser directement l'algorithme Apriori.

D'un point de vue efficacité, c'est une bonne stratégie de pousser les contraintes anti-monotones. En effet, l'élagage apporté par celles-ci permet de diminuer le nombre d'itemsets pour lesquels le calcul de la fonction d'évaluation est fait (calcul de la fréquence, par exemple). Or ce calcul est souvent l'étape la plus coûteuse car il nécessite de parcourir la base de données. Plus généralement, nous allons montrer que l'on effectue moins de tests lorsqu'on pousse les contraintes anti-monotones.

Si $\mathcal{C}$ est la conjonction de deux contraintes anti-monotones $\mathcal{C}_{\mathsf{am}}$ et $\mathcal{C}'_{\mathsf{am}}$, on peut se demander si il est plus efficace (stratégie `push`) d'utiliser Apriori avec la contrainte $\mathcal{C} = \mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}'_{\mathsf{am}}$

```
push
1   k := 1
2   Cand := {(I, 0) | I ∈ Items}
3   tant que Cand ≠ ∅ faire
4       Cand := passe_bd(Cand, bd)
5       Cand := {S ∈ Cand | C_am(S.item)}
6       Cand := {S ∈ Cand | C'_am(S.item)}
7       S_k := Cand
8       Cand := apriori_gen(S_k)
9       k := k + 1
10  fait
11  retourne ⋃_{1≤i≤k} S_i
```

```
g&t
1   k := 1
2   Cand := {(I, 0) | I ∈ Items}
3   tant que Cand ≠ ∅ faire
4       Cand := passe_bd(Cand, bd)
5       Cand := {S ∈ Cand | C_am(S.item)}
6       S_k := Cand
7       Cand := apriori_gen(S_k)
8       k := k + 1
9   fait
10  retourne {S ∈ ⋃_{1≤i≤k} S_i | C'_am(S.item)}
```

FIG. 2.4 – Les algorithmes `push` et `g&t`.

ou (stratégie `g&t`) d'utiliser une stratégie "générer et tester", i.e., d'abord générer $\mathsf{Th}(\mathcal{C}_{\mathsf{am}})$ avec Apriori puis tester chaque $S \in \mathsf{Th}(\mathcal{C}_{\mathsf{am}})$ avec la contrainte $\mathcal{C}'_{\mathsf{am}}$ (les deux algorithmes sont présentés dans la figure 2.4).

Étant donné un algorithme $\mathcal{A}$, désignons par $\mathrm{Test}_{\mathcal{A}}(\mathcal{C})$ la collection des itemsets pour lesquels l'algorithme $\mathcal{A}$ effectue le test de la contrainte $\mathcal{C}$. Par exemple, dans le cas de l'algorithme Apriori, on sait que $\mathrm{Test}_{apriori}(\mathcal{C}_{\gamma\text{-}\mathsf{freq}}) = \mathsf{Th}(\mathcal{C}_{\gamma\text{-}\mathsf{freq}}) \cup \mathcal{B}d^{-}(\mathsf{Th}(\mathcal{C}_{\gamma\text{-}\mathsf{freq}}))$ [MT97]. On a la proposition suivante.

Proposition 11 *Si* $Test_{\mathtt{push}}$ *correspond à la stratégie* `push` *et* $Test_{\mathtt{g\&t}}$ *correspond à la stratégie* `g&t`*, alors* $Test_{\mathtt{push}}(\mathcal{C}_{\mathsf{am}}) \subseteq Test_{\mathtt{g\&t}}(\mathcal{C}_{\mathsf{am}})$ *et* $Test_{\mathtt{push}}(\mathcal{C}'_{\mathsf{am}}) \subseteq Test_{\mathtt{g\&t}}(\mathcal{C}'_{\mathsf{am}})$*.*

Ce théorème établit que l'algorithme `push` conduit à moins de tests de contraintes que l'algorithme `g&t`. En d'autres termes, il est toujours intéressant de pousser des contraintes anti-monotones.

En particulier, le test de la contrainte $\mathcal{C}_{\mathsf{am}}$ est souvent très coûteux (si c'est un test de fréquence minimale par exemple). Par conséquent, la complexité de l'algorithme dépend essentiellement de la taille de $\mathrm{Test}(\mathcal{C}_{\mathsf{am}})$. On voit alors que si on pousse une autre contrainte anti-monotone $\mathcal{C}'_{\mathsf{am}}$ alors la taille de cet ensemble diminue. Nous verrons que ce n'est plus nécessairement le cas si la contrainte n'est pas anti-monotone (voir section 2.7.4 page suivante).

2.7.3 Utiliser seulement les contraintes syntaxiques

Si la contrainte $\mathcal{C} = \mathcal{C}_{synt} \wedge \mathcal{C}_{autre}$ contient une contrainte syntaxique $\mathcal{C}_{synt}$, on peut imaginer d'utiliser la technique suivante.

- générer tous les itemsets qui satisfont cette contrainte (i.e., $\mathsf{Th}(\mathcal{C}_{synt})$) ;
- calculer la fonction d'évaluation pour tous les itemsets de $\mathsf{Th}(\mathcal{C}_{synt})$ en une passe sur les données ;
- et finalement tester la contrainte $\mathcal{C}_{autre}$ sur $\mathsf{Th}(\mathcal{C}_{synt})$ (en utilisant les valeurs de la fonction d'évaluation calculées pendant l'étape précédente).

Avec cette stratégie, on n'utilise pas du tout l'élagage apporté par les contraintes anti-monotones. Seule la contrainte syntaxique permet de limiter la taille de la collection considérée. Cette stratégie a été expérimentée sur les itemsets avec l'algorithme Hybrid(0) [NLHP98]. Les auteurs montrent que, sans surprise, cette technique est très inefficace. En fait, elle ne

pourrait être valable que si la contrainte syntaxique est extrêmement sélective (i.e., $\mathsf{Th}(\mathcal{C}_{synt})$ est très petit). Toutefois, dans ce cas, l'extraction devient un test d'hypothèse.

Une stratégie d'évaluation raisonnable lorsque l'espace de recherche est grand ne peut pas se passer de l'élagage apporté par les contraintes anti-monotones.

2.7.4 Utilisation d'autres contraintes

Notre algorithme générique nous donne de nouvelles possibilités d'optimisation. Alors que l'algorithme Apriori n'utilise que les contraintes anti-monotones, nous avons maintenant la possibilité de pousser également les contraintes monotones ou faiblement anti-monotones.

Cependant, la situation n'est plus aussi claire qu'avec les contraintes anti-monotones. Il n'est pas toujours vrai que pousser des contraintes qui ne sont pas anti-monotones améliore les performances.

Il y a deux raisons pour cela. Premièrement, lorsqu'on utilise des contraintes non monotones, les étapes de génération des candidats et d'élagage peuvent devenir complexes. L'exemple extrême est donné par l'algorithme Spirit(R) où les auteurs poussent une contrainte syntaxique qui n'a pas forcément de bonnes propriétés. Dans ce cas, il n'y a pas de méthode efficace pour générer les candidats, si bien que le nombre de candidats générés est si grand que le temps mis par cette génération devient le facteur limitant de l'algorithme.

Deuxièmement, le fait de pousser des contraintes non anti-monotones peut aussi diminuer l'efficacité de l'élagage apporté par les contraintes anti-monotones. En effet, pour que cet élagage puisse avoir lieu, il faut que l'algorithme trouve des itemsets qui ne satisfont pas la contrainte anti-monotone. Dans ce cas, les sur ensembles de ces itemsets seront éliminés soit parce qu'ils ne seront pas générés, soit parce qu'ils seront élagués. Cependant, si un itemset est éliminé parce qu'il ne satisfait pas une contrainte non anti-monotone, alors il ne pourra pas servir pour l'élagage. Cela se traduit dans les algorithmes au niveau de la fonction de génération des candidats et au niveau de la fonction d'élagage. Il suffit de comparer les fonctions de génération d'Apriori (page 46) et de notre algorithme générique (page 72) pour voir que dans le cas de la génération par Apriori, pour qu'un itemset soit généré, il faut que deux de ses sous ensembles aient été extraits alors que dans l'algorithme générique un seul suffit. De même, dans la fonction d'élagage d'Apriori, tous les sous ensembles d'un itemset sont vérifiés, alors que dans notre algorithme (page 72), seulement ceux qui satisfont la contrainte non anti-monotone sont testés.

Dans la proposition suivante, on compare l'ensemble des itemsets sur lesquels la fonction d'évaluation est calculée suivant que l'on pousse ou non la contrainte non anti-monotone. Cette comparaison nous renseigne sur le gain en efficacité que l'on peut espérer en poussant des contraintes. On suppose que l'étape la plus coûteuse de l'algorithme d'extraction est le calcul de la fonction d'évaluation.

Proposition 12 *Soit une contrainte $\mathcal{C} = \mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}}$. Si on extrait $\mathsf{Th}(\mathcal{C})$ en utilisant* Apriori *pour extraire $\mathsf{Th}(\mathcal{C}_{\mathsf{am}})$ et en filtrant ensuite les itemsets qui ne satisfont pas $\mathcal{C}_{\mathsf{m}}$, alors la fonction d'évaluation sera calculée sur les itemsets de*

$$\mathcal{E}_1 = \mathsf{Th}(\mathcal{C}_{\mathsf{am}}) \cup \mathcal{B}d^-(\mathsf{Th}(\mathcal{C}_{\mathsf{am}})).$$

Si on pousse la contrainte $\mathcal{C}_{\mathsf{m}}$ en utilisant notre algorithme générique, la fonction d'évaluation est calculée sur

$$\mathcal{E}_2 = \mathsf{Th}(\mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\mathsf{am}}) \cup \min(\mathsf{Th}(\mathcal{C}_{\mathsf{m}}) \setminus \mathsf{Th}(\mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\mathsf{am}})).$$

Preuve : La première relation pour l'algorithme Apriori est bien connue [MT97].
Si on appelle $\mathcal{E}_2$ l'ensemble des itemsets sur lesquels la fonction d'évaluation est calculée par notre algorithme générique, alors il est clair que $\mathsf{Th}(\mathcal{C}_\mathsf{m} \wedge \mathcal{C}_\mathsf{am}) \subseteq \mathcal{E}_2$. Soit S un itemset de $\min(\mathsf{Th}(\mathcal{C}_\mathsf{m}) \setminus \mathsf{Th}(\mathcal{C}_\mathsf{m} \wedge \mathcal{C}_\mathsf{am}))$, donc $\mathcal{C}_\mathsf{m}(S)$ est vrai et $\mathcal{C}_\mathsf{am}(S)$ et faux. De plus, tous les sous ensembles de S sont soit dans $\mathsf{Th}(\mathcal{C}_\mathsf{m} \wedge \mathcal{C}_\mathsf{am})$, soit hors de $\mathsf{Th}(\mathcal{C}_\mathsf{m})$. Il y a alors plusieurs cas :
- si il existe un sous ensemble T de S tel que $|T| = |S| - 1$ et $T \in \mathsf{Th}(\mathcal{C}_\mathsf{m} \wedge \mathcal{C}_\mathsf{am})$ alors S est généré à partir de T par la fonction **gen_cand** (72) à la ligne 4 ;
- sinon, tous les sous ensembles de S sont hors de $\mathsf{Th}(\mathcal{C}_\mathsf{m})$, et dans ce cas S est dans $\mathcal{B}d^-(\mathsf{Th}(\neg\mathcal{C}_\mathsf{m}))$ et il est donc généré à la ligne 7.

Enfin, S ne peut pas être élagué car aucun de ces sous ensemble ne satisfait $\mathcal{C}_\mathsf{m}$ sans satisfaire aussi $\mathcal{C}_\mathsf{am}$. Finalement, $S \in \mathcal{E}_2$ et donc $\mathcal{E}_2 \supseteq \mathsf{Th}(\mathcal{C}_\mathsf{m} \wedge \mathcal{C}_\mathsf{am}) \cup \min(\mathsf{Th}(\mathcal{C}_\mathsf{m}) \setminus \mathsf{Th}(\mathcal{C}_\mathsf{m} \wedge \mathcal{C}_\mathsf{am}))$.
Montrons l'inclusion inverse. Soit $S \in \mathcal{E}_2$. S a donc été généré par la fonction de génération des candidats et il n'a pas été élagué. Il y a donc deux cas.
- Soit il a été généré par la ligne 7, auquel cas il est dans $\mathcal{B}d^-(\mathsf{Th}(\neg\mathcal{C}_\mathsf{m}))$ ce qui veut dire que $\mathcal{C}_\mathsf{m}(S)$ est vrai et que tous ses sous ensembles sont hors de $\mathsf{Th}(\mathcal{C}_\mathsf{m})$. Il est donc soit dans $\mathsf{Th}(\mathcal{C}_\mathsf{m} \wedge \mathcal{C}_\mathsf{am})$ s'il satisfait $\mathcal{C}_\mathsf{am}$ soit dans $\min(\mathsf{Th}(\mathcal{C}_\mathsf{m}) \setminus \mathsf{Th}(\mathcal{C}_\mathsf{m} \wedge \mathcal{C}_\mathsf{am}))$ sinon.
- Soit il est généré en étendant un élément de $\mathsf{Th}(\mathcal{C}_\mathsf{m} \wedge \mathcal{C}_\mathsf{am})$. Comme il n'a pas été élagué, cela signifie que chacun de ses sous ensembles soit ne satisfait pas $\mathcal{C}_\mathsf{m}$ soit satisfait $\mathcal{C}_\mathsf{m} \wedge \mathcal{C}_\mathsf{am}$ (si un de ses sous ensembles satisfaisait $\mathcal{C}_\mathsf{m}$ et pas $\mathcal{C}_\mathsf{am}$, il serait élagué). Par conséquent, S est dans $\min(\mathsf{Th}(\mathcal{C}_\mathsf{m}) \setminus \mathsf{Th}(\mathcal{C}_\mathsf{m} \wedge \mathcal{C}_\mathsf{am}))$.

□

En fait, $\min(\mathsf{Th}(\mathcal{C}_\mathsf{m}) \setminus \mathsf{Th}(\mathcal{C}_\mathsf{m} \wedge \mathcal{C}_\mathsf{am}))$ joue le même rôle que $\mathcal{B}d^-(\mathsf{Th}(\mathcal{C}_\mathsf{am}))$ dans Apriori. Si on se restreint à $\mathsf{Th}(\mathcal{C}_\mathsf{m})$, l'ensemble des itemsets minimaux qui ne satisfont pas $\mathcal{C}_\mathsf{am}$ est exactement $\min(\mathsf{Th}(\mathcal{C}_\mathsf{m}) \setminus \mathsf{Th}(\mathcal{C}_\mathsf{m} \wedge \mathcal{C}_\mathsf{am}))$. Nous rappelons que, par définition, $\mathcal{B}d^-(\mathsf{Th}(\mathcal{C}_\mathsf{am})) = \min(2^{\mathsf{Items}} \setminus \mathsf{Th}(\mathcal{C}_\mathsf{am}))$, le parallèle est donc évident.

Il est clair que $\mathsf{Th}(\mathcal{C}_\mathsf{m} \wedge \mathcal{C}_\mathsf{am}) \subseteq \mathsf{Th}(\mathcal{C}_\mathsf{am})$. C'est même cette inclusion qui motive le fait de pousser des contraintes non monotones. Par contre on ne peut rien dire dans le cas général sur $\min(\mathsf{Th}(\mathcal{C}_\mathsf{m}) \setminus \mathsf{Th}(\mathcal{C}_\mathsf{m} \wedge \mathcal{C}_\mathsf{am}))$ et $\mathcal{E}_1$. En particulier, la différence entre les cardinaux de ces deux ensembles peut être arbitrairement élevée (revoir l'exemple 18 page 68).

2.7.5 Conclusion

Les différentes étapes d'une stratégie simple pour extraire les règles d'association satisfaisant une contrainte $\mathcal{C}$ sont :

1. génération de la base de données binaire dans laquelle les règles sont extraites (par exemple par l'utilisation de requêtes SQL sur une base de données relationnelle) ;
2. transformation de la contrainte $\mathcal{C}$ sur les règles en une contrainte sur les itemsets $\mathcal{C}_i = \mathcal{C}_\mathsf{am} \wedge \mathcal{C}_{autre}$;
3. extraction des itemsets satisfaisant la contrainte $\mathcal{C}_\mathsf{am}$ avec l'algorithme Apriori ;
4. élimination des itemsets qui ne satisfont pas $\mathcal{C}_{autre}$;
5. génération des règles qui satisfont $\mathcal{C}$ à partir des itemsets extraits.

Dans cette stratégie, la troisième étape est généralement très coûteuse car elle requiert souvent d'accéder aux données (e.g., pour un calcul de fréquence). Cependant, nous avons vu des techniques qui permettent de pousser d'autres contraintes que celles qui sont seulement anti-monotones. Cela nous permet d'envisager une nouvelle stratégie :

1. génération de la base de données binaire ;
2. transformation de la contrainte $\mathcal{C}$ sur les règles en une contrainte sur les itemsets $\mathcal{C}_i = \mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{autre}$ où $\mathcal{C}_{\mathsf{m}}$ et une contrainte monotone ou faiblement anti-monotone ;
3. extraction des itemsets satisfaisant la contrainte $\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}}$ avec l'algorithme générique ;
4. élimination des itemsets qui ne satisfont pas $\mathcal{C}_{autre}$;
5. calcul des fréquences des itemsets qui pourront servir de corps dans une règle (voir la section 2.7.1) ;
6. génération des règles qui satisfont $\mathcal{C}$ à partir des itemsets extraits.

Dans ce cas, des contraintes plus fortes sont utilisées dans la phase d'extraction des itemsets. Cette stratégie peut être beaucoup plus performante que la précédente malgré la nécessité d'accéder une nouvelle fois aux données dans la $5^{\text{ème}}$ étape [GdB00].

Dans le chapitre suivant, nous étudions comment les représentations condensées peuvent être utilisées pour fournir d'autres stratégies d'évaluation et nous présentons une nouvelle stratégie utilisant simultanément les représentations condensées et les contraintes.

Chapitre 3

Extraction de représentations condensées sous contraintes

Dans ce chapitre, nous utilisons notre algorithme générique pour extraire des représentations condensées sous contraintes. Les représentations condensées que nous utilisons sont les clos, les libres et les δ-libres. Les travaux exposés dans ce chapitre ont été publiés en partie dans [JB02a, JB02b, BJ01a, BJ01b, BBJ00].

Dans la première section, nous présentons différentes représentations condensées ainsi que les algorithmes existants pour les extraire.

Dans la section 2, nous abordons notre contribution principale, l'extraction des représentations condensées sous contraintes. Pour cela, nous utilisons notre algorithme générique défini dans le chapitre précédent. Nous serons amenés à introduire une nouvelle représentation condensée, les itemsets *libres contextuels* pour résoudre les problèmes liés à l'utilisation de contraintes monotones.

La section 3 présente les nouvelles stratégies que cette approche nous permet d'introduire. Ces stratégies sont évaluées expérimentalement dans la section 4 sur le problème de l'extraction des règles d'association avec négations.

3.1 Diverses formes de représentations condensées

3.1.1 Itemsets fermés

Les itemsets fermés ont été pour la première fois utilisés en ECD dans [PBTL99a, PBTL99b]. On rappelle la définition de la fermeture et des itemsets fermés donnée dans le premier chapitre.

Définition 23 (fermeture, fermé) *La* fermeture *d'un itemset S dans une base de données binaire bd, notée $\mathsf{ferm}(S, bd)$, est le plus grand sur ensemble de S qui a la même fréquence que S dans bd. Un itemset S est* fermé *ou* clos *si il est égal à sa propre fermeture.*

Lorsqu'il n'y a pas d'ambiguïté possible sur la base de données, celle-ci est omise dans les notations. Il découle de cette définition qu'un itemset fermé est un itemset dont la fréquence est différente de tous ses sur ensembles. La fermeture a plusieurs propriétés.

Proposition 13 *Soit S et T deux itemsets et bd une base de données binaire alors :*

(i) la fermeture est idempotente $\mathsf{ferm}(\mathsf{ferm}(S, bd), bd) = \mathsf{ferm}(S, bd)$ *(et donc la fermeture d'un itemset est un fermé) ;*
(ii) la fermeture est croissante par rapport à l'inclusion : si $S \subseteq T$ *alors* $\mathsf{ferm}(S, bd) \subseteq \mathsf{ferm}(T, bd)$ *;*
(iii) $S \subseteq \mathsf{ferm}(S, bd)$ *;*
(iv) la règle $S \Rightarrow T$ *est exacte dans bd (i.e., de confiance 1) si et seulement si* $T \subseteq \mathsf{ferm}(S, bd)$*. En particulier, la règle* $S \Rightarrow (\mathsf{ferm}(S, bd) \setminus S)$ *est une règle exacte dans bd.*

Preuve : Les propriétés (i), (iii) et (iv) découlent directement de la définition. Pour la propriété (ii) : Soit $S \subseteq T$. Comme $S \subseteq T$ et que $S \Rightarrow (\mathsf{ferm}(S) \setminus S)$ est une règle exacte (iv), alors $T \Rightarrow (\mathsf{ferm}(S) \setminus T)$ est aussi une règle exacte et en appliquant encore une fois (iv), $\mathsf{ferm}(S) \subseteq \mathsf{ferm}(T)$. □

On peut aussi introduire les itemsets fermés en utilisant une connexion de galois entre 2^{Items} et $2^{\mathcal{T}}$ où $\mathcal{T}$ est l'ensemble des transactions dans la base de données. Pour cela, on définit deux fonctions notées f et g entre ces deux ensembles. La fonction f associe à chaque itemset l'ensemble des transactions dans lesquels il apparaît (f est donc la fonction que nous avons appelée support dans le chapitre 1). La fonction g associe à un ensemble de transactions, l'ensemble des items qui apparaissent simultanément dans toutes ces transactions (autrement dit $g(\{t_1, \ldots, t_n\})$ est l'intersection des transactions t_1, ..., t_n). Le couple de fonctions (f, g) est appelé une connexion de galois entre 2^{Items} et $2^{\mathcal{T}}$. On peut alors définir un concept comme étant un couple $(I, T) \in 2^{\mathsf{Items}} \times 2^{\mathcal{T}}$ tel que $T = f(I)$ et $I = g(T)$. On montre alors que si on munit l'ensemble des concepts de l'ordre $\leq$ définit par $(I, T) \leq (I', T')$ si $I \subseteq I'$ et $T' \subseteq T$ alors il forme un treillis appelé treillis de concepts. Si nous revenons à la définition d'un concept, nous voyons que $I = g \circ f(I)$. Or il est facile de voir que $g \circ f = \mathsf{ferm}$, par conséquent l'ensemble des itemsets fermés muni de l'ordre induit par l'inclusion est un treillis. Un exemple d'un tel treillis est présenté dans la figure 3.2. Le lecteur intéressé par les propriétés du treillis de concept et du treillis des fermés peut se reporter à [PBTL99c].

Une fois que la fermeture est définie, nous pouvons introduire une relation d'équivalence sur les itemsets.

Définition 24 (relation d'équivalence [BTP+02]) *Deux itemsets* S *et* T *sont équivalents dans la base de données bd* ($S \sim_f T$) *si ils ont même fermeture dans bd.*

Proposition 14
(i) deux itemsets équivalents ont même fréquence ;
(ii) si deux itemsets S *et* T *ont même fréquence et que* $S \subseteq T$ *alors ils sont dans la même classe d'équivalence ;*
(iii) chaque classe d'équivalence a exactement un itemset maximal qui est fermé ;
(iv) si S *et* T *sont dans la même classe d'équivalence et que* $S \subseteq V \subseteq T$ *alors* V *est aussi dans cette classe d'équivalence.*

Le point (iv) de la proposition nous dit que les classes d'équivalence de cette relation sont des collections convexes d'itemsets. Elles sont caractérisées par leurs éléments maximaux et minimaux. Les éléments maximaux sont les fermés et les éléments minimaux sont appelés les générateurs [PBTL99c], les libres [BBR00] ou les motifs clés [BTP+02]. Une notion de "libre" très proche de celle que nous utilisons est aussi utilisée dans le cadre de l'extraction de dépendances fonctionnelles et de dépendances d'inclusion [NC00].

Définition 25 (itemset libre) *Un itemset S est libre si c'est un élément minimal d'une classe d'équivalence de la relation $\sim_f$.*

Proposition 15 *Les propositions suivantes sont équivalentes :*
(i) S est un itemset libre ;
(ii) la fréquence de S est différente de la fréquence de tous ses sous ensembles stricts ;
(iii) S n'est pas inclus dans la fermeture d'un de ses sous ensembles stricts ;
(iv) il n'existe pas de règle exacte $X \Rightarrow Y$, $Y \neq \emptyset$ telle que $(X \cup Y) \subseteq S$.

$bd_1 =$

tid	
1	ABCDE
2	ABCD
3	ACD
4	ABE
5	CD
6	CE

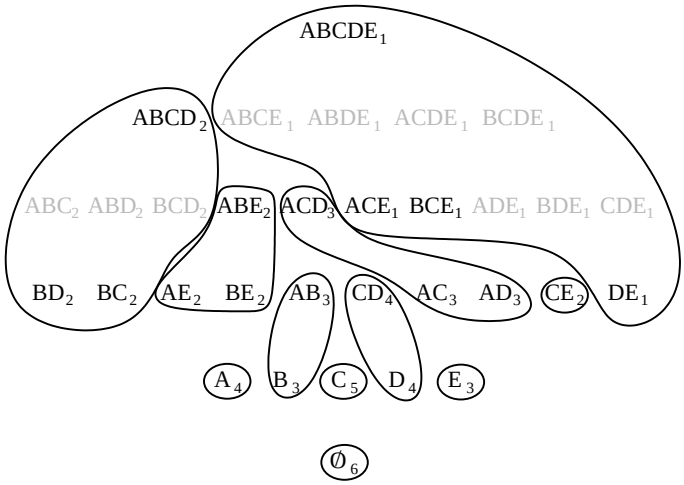


FIG. 3.1 – Classes d'équivalence pour la relation $\sim_f$.

Exemple 22 *La figure 3.1 donne la partition de 2^{Items} en classes d'équivalence pour $\sim_f$ sur la base de données bd_1. Il y a 11 classes d'équivalence donc 11 itemsets fermés qui sont les itemsets maximaux des classes d'équivalences (∅, A, C, E, AB, CD, CE, ABE, ACD, ABCD et ABCDE). Il y a 16 itemsets libres qui sont les itemsets minimaux des classes d'équivalences (∅, A, B, C, D, E, AE, BC, BD, BE, AC, AD, CE, DE, ACE et BCE). On voit que lorsqu'une classe d'équivalence est réduite à un seul élément alors celui-ci est à la fois libre et fermé ; c'est le cas ici de l'ensemble vide et de A, C, E et CE.*

Dans la suite, on notera $\mathsf{FreqFerm}(\gamma, bd)$ la collection des itemsets fermés γ-fréquents dans bd et $\mathsf{FreqLibre}(\gamma, bd)$ la collection des itemsets libres γ-fréquents dans bd. Les collections des itemsets fermés et libres sont représentées dans la figure 3.2 page suivante.

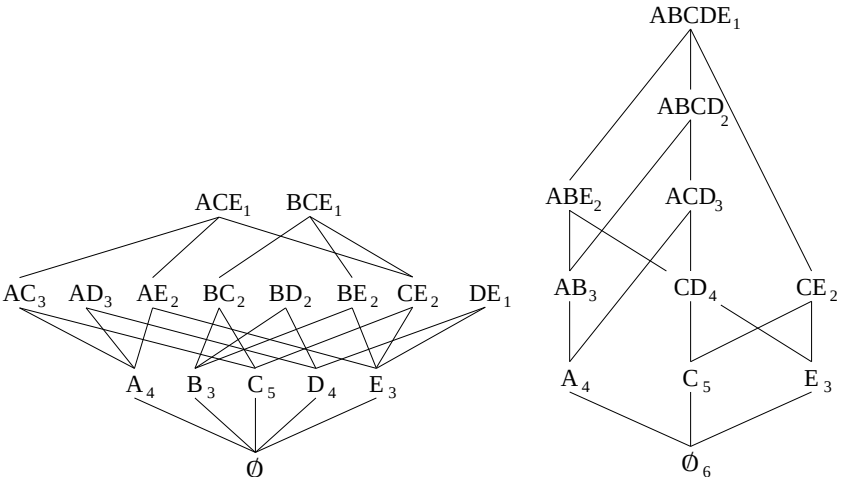
3.1.2 Algorithmes d'extraction des itemsets fermés

Plusieurs algorithmes existent dans la littérature pour l'extraction des itemsets fermés fréquents tels que Close [PBTL99c], a-Close [PBTL99a], Pascal [BTP+02], Closet [PHM00] et Charm [ZH02]. Les trois premiers effectuent un parcours niveau par niveau de l'espace de recherche alors que les deux derniers font un parcours en profondeur d'abord.

Introduisons deux nouvelles contraintes.

Définition 26 (Contrainte $\mathcal{C}_{\mathsf{clos}}$ et $\mathcal{C}_{\mathsf{libre}}$) *On définit la contrainte de fermeture $\mathcal{C}_{\mathsf{clos}}$ par :*

$$\mathcal{C}_{\mathsf{clos}}(S) = (S \text{ est un itemset fermé}).$$



(a) libres (b) fermés

FIG. 3.2 – La collection des itemsets libres et le treillis des itemsets fermés.

On définit également la contrainte de liberté $\mathcal{C}_{\mathsf{libre}}$ *par :*

$$\mathcal{C}_{\mathsf{libre}}(S) = (S \textit{ est un itemset libre}).$$

Ainsi, les collections $\mathsf{FreqFerm}(\gamma, bd)$ des itemsets fermés γ-fréquents et $\mathsf{FreqLibre}(\gamma, bd)$ des itemsets libres γ-fréquents ne sont autre que les théories $\mathsf{Th}(bd, \mathcal{C}_{\mathsf{clos}} \wedge \mathcal{C}_{\gamma\text{-}\mathsf{freq}})$ et $\mathsf{Th}(bd, \mathcal{C}_{\mathsf{libre}} \wedge \mathcal{C}_{\gamma\text{-}\mathsf{freq}})$.

Algorithme Close

L'algorithme Close [PBTL99c] utilise l'anti-monotonicité de la contrainte $\mathcal{C}_{\mathsf{libre}}$. En effet, on constate que tous les sous ensembles d'un itemset libre sont libres. Cela est clair avec la caractérisation des itemsets libres par le point (iv) de la proposition 15 page précédente. Par contre, on se convainc en examinant la figure 3.1 page précédente que $\mathcal{C}_{\mathsf{clos}}$ n'est pas anti-monotone ni monotone. Ainsi, pour extraire les itemsets fermés fréquents, il suffit d'extraire les itemsets libres fréquents et de calculer leur fermeture. La complétude de cette méthode est donnée par la proposition suivante.

Proposition 16 *La collection des itemsets fermés fréquents est obtenue en calculant la collection des fermetures des itemsets libres fréquents, i.e.,*

$$\mathsf{FreqFerm}(\gamma, bd) = \{\mathsf{ferm}(S) \mid S \in \mathsf{FreqLibre}(\gamma, bd)\}.$$

Preuve : $\mathsf{FreqFerm}(\gamma, bd) \supseteq \{\mathsf{ferm}(S) \mid S \in \mathsf{FreqLibre}(\gamma, bd)\}$: les fermetures des libres sont des fermés (proposition 13 point (i)).
$\mathsf{FreqFerm}(\gamma, bd) \subseteq \{\mathsf{ferm}(S) \mid S \in \mathsf{FreqLibre}(\gamma, bd)\}$: si S est un fermé fréquent alors soit T un itemset libre de sa classe d'équivalence. T est fréquent (car il a même fréquence que S) et sa fermeture est S. □

On a en fait montré que la fonction ferm est une surjection de l'ensemble des libres sur l'ensemble des fermés qui conserve la fréquence. On a donc également montré que les ensembles fermés fréquents sont moins nombreux que les libres fréquents quel que soit le seuil

de fréquence. Comme il y a exactement un fermé par classe d'équivalence, certaines classes d'équivalence contiennent donc plusieurs itemsets libres (voir la figure 3.1).

L'algorithme Close est une instanciation de l'algorithme Apriori pour extraire des itemsets libres fréquents. Pour obtenir les fermés fréquents, l'algorithme Close retourne en fait les fermetures des libres fréquents. Un itemset S est représenté par un enregistrement S comportant 3 champs :

- un champ $S.\mathsf{item}$ contenant les items composant l'itemset ;
- un champ $S.\mathsf{freq}$ pour la fréquence ;
- un champ $S.\mathsf{ferm}$ pour la fermeture. En fait ce champ contient $\mathsf{ferm}(S) \setminus S$, en effet il n'est pas utile de stocker les items de S deux fois, dans $S.\mathsf{item}$ et dans $S.\mathsf{ferm}$.

On notera parfois cet enregistrement comme le triplet $(S.\mathsf{item}, S.\mathsf{freq}, S.\mathsf{ferm})$.

Algorithme 5 (Close)

Entrée : Une base de données binaire bd, un seuil de fréquence γ
Sortie : $\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{clos}}, \mathsf{freq})$

```
1   k := 0
2   Cand := {(∅, 0, ∅)}
3   tant que Cand ≠ ∅ faire
4       Cand := passe_bd_close(Cand, bd)     /* calcul de S.freq et S.ferm pour S ∈ Cand */
5       Cand := {S ∈ Cand | S.freq ≥ γ}        /* les candidats infréquents sont éliminés */
6       S_k := {(S.ferm, S.freq) | S ∈ Cand}
7       Cand := close_gen(S_k)              /* génération des candidats du niveau suivant */
8       k := k + 1
9   fait
10  retourne ⋃_{1≤i≤k} S_i
```

Lors de la passe sur la base de données, l'algorithme Close calcule la fermeture des itemsets candidats en plus de leur fréquence. Pour calculer la fermeture de S, il suffit de faire l'intersection de toutes les transactions qui supportent S. Cela est fait en parallèle avec le calcul de la fréquence en utilisant la fonction suivante pour la passe sur les données. Le champ $S.\mathsf{ferm}$ est initialisé avec $t \setminus S.\mathsf{item}$ où t est la première transaction supportant S. De cette manière, ce champ contient bien $\mathsf{ferm}(S) \setminus S$ à la fin de la passe sur les données.

```
passe_bd_close(Cand, bd)
1   pour toute transaction t ∈ bd faire
2       pour tout itemset candidat S ∈ Cand faire
3           si S.item ⊆ t alors
4               S.freq := S.freq + 1
5               si S.freq = 1 alors S.ferm := t \ S.item
6               sinon S.ferm := S.ferm ∩ t finsi
7           finsi
8       fait
9   fait
```

Lors de la génération des candidats pour le niveau suivant, il faut vérifier que les candidats sont bien des itemsets libres. Pour cela, l'algorithme Close utilise la caractérisation des libres par le point (iii) de la proposition 15 page 83 (un itemset est libre s'il n'est pas inclus dans la fermeture d'un de ses sous ensembles stricts).

close_gen($\mathcal{S}$)
1 /* génération des candidats potentiels */
2 $C := \{S[1]S[2]\dots S[k-1]T[k-1] \mid S,T \in \mathcal{S}.\mathsf{item}$ et $S[1]=T[1],\dots,S[k-2]=T[k-2]$ et $T[k-1] \lhd S[k-1]\}$
3 /* élagage et vérification que les itemsets sont libres*/
4 **pour tout** $S \in C$ **faire**
5 **pour tout** $T \subset S$, $|T| = |S|-1$ **faire**
6 **si** $T \notin \mathcal{S}.\mathsf{item}$ **ou** $S \subseteq \mathsf{ferm}(T)$ **alors**
7 $C := C \setminus \{S\}$ **finsi**
8 **fait**
9 **fait**
10 **retourne** $\{(S, 0, \emptyset) \mid S \in C\}$

Par rapport à Apriori, Close élague les itemsets qui ne sont pas libres (ligne 6 dans la fonction de génération des candidats). Dans les données fortement corrélées, un grand nombre d'itemsets ne sont pas libres. Donc cet élagage est très efficace. Par contre, si ce n'est pas le cas, il est possible que tous les itemsets soient libres (voir l'exemple 23). Dans ce cas, Close explore exactement la même partie du treillis que Apriori. Comme Close doit supporter le surcoût du calcul des fermetures pendant la passe sur les données, il est dans ce cas un peu plus lent que Apriori.

Si on utilise Close pour extraire les itemsets fermés 1-fréquents dans la base de données bd_1, celui-ci parcourt exactement la collection des itemsets libres représentée dans la figure 3.2(a).

Exemple 23 *Nous allons montrer comment construire une base de données dans laquelle tous les itemsets sont à la fois libres et fermés. Considérons une base de données mise sous la forme d'une matrice carrée de 0 et de 1, il y a donc autant de lignes que d'items. La matrice est remplie de 1 sauf la diagonale qui est constituée de 0. Dans cet exemple, on suppose que les items sont des entiers* $\mathsf{Items} = \{1, 2, \dots, n\}$. *Soit* $S = \{i_1, i_2, \dots, i_k\}$ *un itemset de taille* k. *S est supporté par toutes les lignes de la base de données sauf les lignes de numéros* i_1, i_2, ..., i_k. *Sa fréquence vaut donc* $\mathsf{freq}(S) = n - |S|$. *Il est donc clair qu'aucun itemset ne peut avoir une fréquence égale à l'un de ses sous ensembles stricts. Par conséquent tous les itemsets sont libres et donc également fermés.*

Algorithme Pascal

L'algorithme Pascal [BTP+02] est une optimisation de Close pour avoir des performances similaires à Apriori dans le cas où la plupart des itemsets sont fermés. Pour cela, l'algorithme Pascal ne calcule pas la fermeture des candidats pendant la passe sur les données.

L'algorithme Pascal fonctionne comme Apriori. Il considère tous les itemsets fréquents. Par contre, lorsqu'il sait qu'un itemset ne peut pas être libre, il ne calcule pas sa fréquence pendant la passe sur les données. En effet, si un itemset n'est pas libre, il a nécessairement la même fréquence que l'un de ses sous ensembles (voir le point (ii) de la proposition 14 page 82). Comme la fréquence est décroissante par rapport à l'inclusion, sa fréquence est égale au minimum des fréquences de ses sous ensembles.

Ce calcul est fait pendant l'étape de génération des candidats. Si un itemset généré a un sous ensemble qui est marqué non libre alors il est lui aussi marqué non libre et sa fréquence est calculée comme expliqué ci-dessus. Ainsi, il ne sera pas nécessaire de calculer sa fréquence lors de la passe sur les données.

Pour déterminer si un candidat qui n'est pas marqué non libre est effectivement libre, il faut comparer sa fréquence avec celle de ses sous ensembles après la passe sur les données. Si il n'est pas libre il est alors marqué comme tel.

Cet algorithme considère exactement les mêmes itemsets que Apriori. Par contre, il ne calculera la fréquence que des itemsets libres et des itemsets qui ont tous leurs sous ensembles libres (i.e., la frontière négative de la collection des itemsets libres). De plus, le surcoût par rapport à Apriori est limité à la gestion des marques qui indique qu'un itemset n'est pas libre : ce surcoût est donc totalement négligeable.

Pascal est donc toujours plus performant que Apriori (ce qui est vérifié expérimentalement) y compris sur des données où presque tous les itemsets sont fermés et libres. Par rapport à Close, Pascal ne fait pas ce calcul de fermetures pendant la passe sur les données. Par contre, il calcule la fréquence des itemsets qui sont dans la la frontière négative de la collection des itemsets libres ce que ne fait pas Close. Si on utilise Pascal pour extraire les itemsets 1-fréquents dans la base de données bd_1, il calculera la fréquence de tous les itemsets libres (comme Close, voir la figure 3.2(a)) et il calculera en plus la fréquence de `ADE` et `BDE` car tous leurs sous ensembles sont libres. Finalement, d'après les expérimentations des auteurs, Pascal semble toujours meilleur que Close.

L'algorithme Pascal calcule tous les itemsets fréquents et non pas seulement les itemsets fermés fréquents. Cependant, il est assez simple de calculer ces derniers à partir de la collection des itemsets fréquents.

Algorithme Closet

L'algorithme Closet [PHM00] effectue un parcours en profondeur de l'espace de recherche pour calculer les itemsets fermés. Lorsque l'algorithme a fini de traiter un nœud S, il explore les fils fréquents de S par ordre de fréquence croissante. Les fils d'un nœud S sont définis comme étant les itemsets de la forme $S \cup \{I\}$, où I est un item, qui n'ont pas encore été explorés dans d'autres branches de l'arbre. Par exemple, dans la figure 3.3, l'itemset `AB` n'est pas un fils de `A` car il a déjà été considéré dans la branche issue de `B`. Cet algorithme utilise deux optimisations pour accélérer son exploration :

1. lorsque l'algorithme traite un itemset S et que certains des fils de S sont inclus dans la fermeture de S alors l'algorithme saute directement au nœud $T = \mathsf{ferm}(S) \cap U$ (U est la réunion des fils de S qui sont inclus dans $\mathsf{ferm}(S)$) sans considérer les descendants de S qui ne sont pas descendants de T ;
2. si S est un itemset et que l'algorithme a déjà exploré $\mathsf{ferm}(S)$ alors les descendants de S ne sont pas explorés.

L'algorithme utilise en fait une troisième optimisation mais celle-ci est très dépendante de la structure de donnée utilisée (FP-tree) et elle ne change pas la manière dont le treillis des itemsets est exploré. Nous ne l'aborderons donc pas ici.

Exemple 24 *La figure 3.3 présente un exemple d'extraction des itemsets fermés 1-fréquents sur la base de données bd_1 (voir figure 3.1 page 83) par l'algorithme* Closet. *Les itemsets fermés sont encadrés et les numéros en indice indiquent l'ordre de parcours de l'arbre. Avec un seuil de fréquence de 1, tous les itemsets sont fréquents. Donc, les fils d'un nœud de l'arbre ne sont pas explorés soit parce que la première optimisation a été utilisée, dans ce cas, une flèche arrondie indique le saut effectué, soit car la seconde optimisation a été utilisée, dans ce cas il y a une astérisque en exposant de l'itemset dont les fils ont été élagués.*

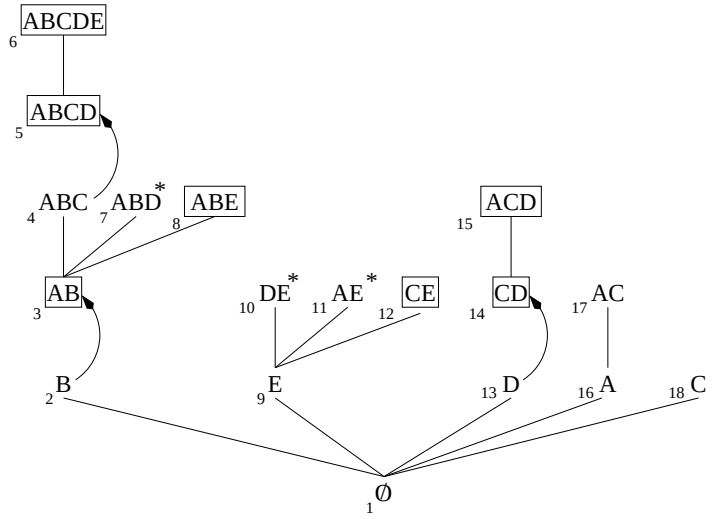


FIG. 3.3 – Parcours du treillis par Closet lors d'une extraction des itemsets fermés 1-fréquents.

Algorithme Charm

L'algorithme Charm [ZH02] utilise le même parcours que Closet. Les différences entre les deux algorithmes sont essentiellement au niveau des structures de données. Charm utilise une représentation *verticale* de la base de données. Cela signifie qu'avec chaque itemset S, Charm stocke la liste des numéros des transactions qui supportent S (cette liste est appelée la *tid list*). L'intérêt de cette représentation est que pour connaître la liste des numéros des transactions qui supportent un sur ensemble T de S, il suffit de faire l'intersection des listes de numéros de S et de U où U est un itemset tel que $S \cup U = T$. En fait, seule l'ancienne version de Charm stockait ces listes. La nouvelle version de Charm (qui est présentée dans [ZH02]) stocke avec un itemset S la liste des numéros des transactions qui supportent le père de S et qui ne supportent pas S (ces listes sont appelées *diffsets*). Dans ce cas, pour obtenir la liste du fils T de S, il suffit de calculer la liste différence des listes de S et U au lieu de faire leur intersection. Cette optimisation permet, selon les auteurs, de diminuer considérablement la taille des listes et donc également la complexité (le calcul d'une différence ou d'une intersection est plus rapide sur des listes courtes).

Une autre différence avec Closet est que la seconde optimisation de Closet n'est pas appliquée de la même manière. Dans Charm cette optimisation est remplacée par la suivante : si S est un itemset et que l'un des frères T de S exploré avant S est supporté par les mêmes transactions que S alors les fils de S ne sont pas explorés. En pratique cette optimisation est très proche de cette de Closet, elle est seulement plus faible (tout itemset sur lequel cette optimisation est appliquée serait passible de la seconde optimisation avec Closet).

Exemple 25 *La figure 3.4 présente l'extraction des itemsets fermés 1-fréquents par* Charm. *On remarque que la seule différence avec le parcours effectué par* Closet *est dans l'application de la seconde optimisation. Alors que dans le cas de* Closet, *les fils des itemsets* `ABD`, `DE` *et* `AE` *n'étaient pas explorés, ici seuls les fils de* `ABD` *ne sont pas explorés.*

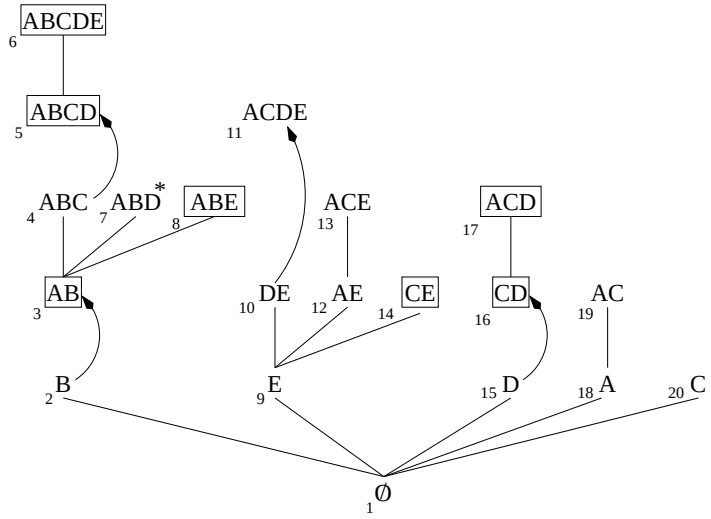


FIG. 3.4 – Parcours du treillis par Charm lors d'une extraction des itemsets fermés 1-fréquents.

Discussion

Les algorithmes d'extraction des itemsets fréquents comme Apriori sont inopérant lorsque les données sont fortement corrélées. En effet, dans ce cas il y a énormément d'itemsets fréquents même pour des seuils de fréquence élevés. Dans ce cas, l'extraction est souvent infaisable pour les seuils de fréquence qui intéressent l'utilisateur.

Si nous comparons maintenant les algorithmes que nous avons présentés, il apparaît que les algorithmes Close et Pascal sont très similaires. Ils extraient tous les deux les itemsets libres pour ensuite calculer leurs fermetures et obtenir ainsi tous les fermés.

Par contre, Closet et Charm ne procèdent pas de la même manière. Tout d'abord, ils font une recherche en profondeur d'abord en utilisant des structures de données très efficaces. Ensuite, ils explorent le treillis en recherchant directement les itemsets fermés. Pour accélérer le parcours, ils utilisent deux optimisations. La première revient à un calcul de fermeture, de façon à sauter directement d'un itemset à sa fermeture. Cette optimisation est en fait très similaire à celle qui consiste à ne considérer que des itemsets libres pour Close ou Pascal. En effet, on peut associer à chaque itemset exploré par Close ou Pascal un itemset libre. Sur les figures 3.3 et 3.4 il suffit de retirer des itemsets les items qui ont été rajoutés lors d'un saut.

Exemple 26 *Si on fait cette opération sur les itemsets explorés par* Closet *sur la figure 3.3, on obtient, dans l'ordre de parcours, les itemsets* ∅, B, BC, BCE, BD, BE, E, DE, AE, CE, D, AD, A, AC *et* C. *Comme on peut le constater sur la figure 3.2, ce sont tous des itemsets libres. Par contre, on constate que l'itemset* ACE *qui est libre, ne figure pas dans cette liste.*

Par contre, la liste des itemsets ainsi obtenus ne contient pas tous les itemsets libres. Cela vient de la seconde optimisation. Cette optimisation reviendrait, dans l'algorithme Close, à élaguer certains itemsets libres. La manière dont ces itemsets sont élagués dans Closet et Charm ne compromet pas la complétude de ces derniers. Par contre, utiliser le même type d'optimisation dans Close n'est pas forcément possible. Il faut en effet s'assurer qu'on ne va pas ainsi élaguer tous les itemsets libres d'une classe d'équivalence car dans ce cas il ne serait plus possible de générer le fermé de cette classe. Dans l'annexe A nous présentons une méthode originale pour adapter cet élagage à l'algorithme Close.

Finalement, les algorithmes les plus efficaces que nous avons présentés sont sans conteste Closet et Charm. La première version de Charm utilisait des *tid lists*. La taille de ces dernières et donc le coût de leur manipulation faisait que Closet était alors plus efficace, d'après les auteurs de Closet [PHM00]. La nouvelle version de Charm utilise des *diffsets* à la place de ces *tid lists*. Cette nouvelle version est plus efficace que Closet, d'après les auteurs de Charm [ZH02].

3.1.3 Représentations condensées utilisant les fermés

Les représentations condensées utilisant des fermés que nous introduisons ici ont été présentées pour la première fois dans [BB00]. La collection $\mathsf{FreqFerm}(\gamma, bd)$ des itemsets fermés γ-fréquents est incluse dans la collection des itemsets γ-fréquents. Lorsque les données sont fortement corrélées, la collection $\mathsf{FreqFerm}(\gamma, bd)$ peut être plusieurs ordres de grandeur plus petite que la collection des itemsets fréquents. Elle est donc particulièrement intéressante en tant que représentation condensée des itemsets fréquents. Nous verrons plus loin que la collection $\mathsf{FreqFerm}(\gamma, bd)$ peut aussi être utilisée pour produire des règles d'association non redondantes.

Il existe un lien entre la collection $\mathsf{FreqFerm}(\gamma, bd)$ et la collection des itemsets fréquents maximaux $\mathcal{B}d^+(\mathsf{Th}(\mathcal{C}_{\gamma\text{-}\mathsf{freq}}))$ que nous avons vue dans le chapitre 1. Cette collection des itemsets fréquents maximaux permet de caractériser les itemsets fréquents, ce sont tous les sous ensembles des itemsets fréquents maximaux, mais ne fourni pas une bonne approximation de leur fréquence. La proposition suivante nous donne le lien entre les fermés et ces ensembles maximaux.

Proposition 17 *La frontière positive $\mathcal{B}d^+(\mathsf{Th}(\mathcal{C}_{\gamma\text{-}\mathsf{freq}}))$ des itemsets γ-fréquents est incluse dans la collection $\mathsf{FreqFerm}(\gamma, bd)$ des itemsets fermés γ-fréquents. Réciproquement, si F est fermé, il est dans la frontière positive $\mathcal{B}d^+(\mathsf{Th}(\mathcal{C}_{\mu\text{-}\mathsf{freq}}))$ des itemsets μ-fréquents avec $\mu = \mathsf{freq}(F)$.*

Preuve : La démonstration de cette proposition est immédiate. En effet, un itemset de la frontière $\mathcal{B}d^+(\mathsf{Th}(\mathcal{C}_{\gamma\text{-}\mathsf{freq}}))$ ne peut pas avoir la même fréquence que l'un de ses sur ensemble strict. Il est donc fermé. □

Nous allons maintenant voir comment régénérer la collection $\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq})$ des itemsets fréquents et leurs fréquences à partir de la collection $\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{clos}}, \mathsf{freq})$ des itemsets fréquents fermés et de leurs fréquences.

Proposition 18 ([PBTL99b])
- *L'ensemble des itemsets γ-fréquents est exactement l'ensemble des sous ensembles des itemsets fermés γ-fréquents ;*
- *soit S un itemset γ-fréquent, alors la fréquence de S est donnée par :*

$$\mathsf{freq}(S) = \max_{\leq} \{\mathsf{freq}(F) \mid S \subseteq F \wedge F \in \mathsf{FreqFerm}(\gamma, bd)\}.$$

Preuve : Le premier point est conséquence de la proposition 17. Soit S un itemset fréquent. On pose $\mathcal{F} = \{\mathsf{freq}(F) \mid S \subseteq F \wedge F \in \mathsf{FreqFerm}(\gamma, bd)\}$. Par décroissance de la fréquence, il est clair que $\mathsf{freq}(S) \geq \max_{\leq}\mathcal{F}$. Ensuite, comme S est fréquent, $\mathsf{ferm}(S)$ l'est aussi et comme $S \subseteq \mathsf{ferm}(S)$, $\mathsf{ferm}(S) \in \mathcal{F}$. Par conséquent $\mathsf{freq}(S) \leq \max_{\leq}\mathcal{F}$. □

Notre première représentation condensée exacte de $\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq})$ est donc

$$\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{clos}}, \mathsf{freq}). \tag{3.1}$$

La régénération de $\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq})$ se fait en générant tous les sous ensembles des itemsets fermés et en calculant leur fréquence comme indiqué dans la proposition 18 page ci-contre.

Exemple 27 *Dans la base de données bd_1 de la figure 3.1 page 83, il y a 10 itemsets fermés 2-fréquents :*

$$\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{clos}}, \mathsf{freq}) = \{(\emptyset, 6), (\mathtt{A}, 4), (\mathtt{C}, 5), (\mathtt{E}, 3), (\mathtt{AB}, 3), (\mathtt{CD}, 4), (\mathtt{CE}, 2), (\mathtt{ABE}, 2), (\mathtt{ACD}, 3), (\mathtt{ABCD}, 2)\}.$$

Les itemsets 2-fréquents sont les sous ensembles de ces fermés. Calculons par exemple la fréquence de B *en utilisant le résultat de la proposition 18. Les itemsets fermés 2-fréquents qui contiennent* B *sont* AB*,* ABE *et* ABCD*. Leurs fréquences sont respectivement de* 3*,* 2 *et* 2*. La fréquence de* B *est donc* 3*.*

3.1.4 Représentations condensées utilisant les libres

Les représentations condensées utilisant les libres que nous introduisons ici ont été présentées pour la première fois dans [BBR00].

La collection des itemsets libres fréquents $\mathsf{FreqLibre}(\gamma, bd)$ peut aussi servir de base à une représentation condensée des itemsets fréquents. La proposition suivante permet de calculer la fréquence d'un itemset fréquent à partir de $\mathsf{Res}(bd, \mathcal{C}_{\mathsf{libre}} \wedge \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq})$.

Proposition 19 ([BBR00]) *Soit S un itemset γ-fréquent, alors la fréquence de S est donnée par :*

$$\mathsf{freq}(S) = \min_{\leq} \{\mathsf{freq}(L) \mid L \subseteq S \wedge L \in \mathsf{FreqLibre}(\gamma, bd)\}.$$

Preuve : Le sens $\leq$ découle encore de la décroissance de la fréquence. Montrons le sens $\geq$. Si S est libre, comme il est fréquent, il appartient à $\mathsf{FreqLibre}(\gamma, bd)$ et on a bien $\geq$. S'il n'est pas libre, alors il existe un sous ensemble libre L de S dans la même classe d'équivalence que S (par définition des itemsets libres). Donc $L \in \mathsf{FreqLibre}(\gamma, bd)$ et l'inégalité est démontrée. □

Cependant, pour pouvoir appliquer cette proposition pour calculer la fréquence de S, il faut déjà savoir que S est γ-fréquent. Or la collection $\mathsf{FreqLibre}(\gamma, bd)$ ne permet pas de le déterminer, c'est sa frontière négative qui va nous permettre de le savoir. Un itemset S appartient à $\mathcal{B}d^-(\mathsf{FreqLibre}(\gamma, bd))$ si tous ses sous ensembles sont dans $\mathsf{FreqLibre}(\gamma, bd) = \mathsf{Th}(\mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{libre}})$ et si S n'y est pas. L'itemset S peut ne pas y appartenir soit parce qu'il n'est pas fréquent, soit parce qu'il n'est pas libre (ou pour ces deux raisons à la fois).

Proposition 20 ([BBR00]) *On note $\mathsf{InFreqLibre}(\gamma, bd) = \mathsf{Th}(\mathcal{C}_{\mathsf{libre}}) \cap \mathcal{B}d^-(\mathsf{Th}(\mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{libre}}))$ la collection des itemsets de la frontière qui sont libres et infréquents. Alors S est un itemset γ-fréquent si et seulement si S n'est pas un sur ensemble d'un élément de $\mathsf{InFreqLibre}(\gamma, bd)$.*

Preuve : Les itemsets de $\mathsf{InFreqLibre}(\gamma, bd)$ ne sont pas γ-fréquents. Donc si S est γ-fréquent, il ne peut pas être un sur ensemble de l'un d'eux. Réciproquement, si S n'est pas γ-fréquent alors il existe un itemset libre L dans la classe de S inclus dans S (si S est libre alors $L = S$). Comme L n'est pas fréquent, il existe un sous ensemble T de L infréquent minimal (i.e., tous les sous ensembles de T sont fréquents). Comme L est libre, T aussi. Donc T est dans $\mathsf{InFreqLibre}(\gamma, bd)$. De plus, $T \subseteq L \subseteq S$. □

La paire

$$(\mathsf{Res}(bd, \mathcal{C}_{\mathsf{libre}} \wedge \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq}), \mathsf{InFreqLibre}(\gamma, bd)) \tag{3.2}$$

est donc bien une représentation condensée exacte de $\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq})$. La collection des itemsets libres infréquents $\mathsf{InFreqLibre}(\gamma, bd)$ permet de connaître quels itemsets sont fréquents et $\mathsf{Res}(bd, \mathcal{C}_{\mathsf{libre}} \wedge \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq})$ permet de calculer leurs fréquences.

Exemple 28 *Dans la base de données bd_1 de la figure 3.1 page 83, il y a 13 itemsets libres fréquents.*

$$\mathsf{Res}(bd, \mathcal{C}_{\mathsf{libre}} \wedge \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq}) =$$
$$\{(\emptyset, 6), (\mathtt{A}, 4), (\mathtt{B}, 3), (\mathtt{C}, 5), (\mathtt{D}, 4), (\mathtt{E}, 3), (\mathtt{AE}, 2), (\mathtt{BE}, 2), (\mathtt{BC}, 2), (\mathtt{BD}, 2), (\mathtt{CE}, 2), (\mathtt{AC}, 3), (\mathtt{AD}, 3)\}\,.$$

La collection des itemsets libres infréquents contient 3 itemsets.

$$\mathsf{InFreqLibre}(\gamma, bd) = \{\mathtt{DE}, \mathtt{ACE}, \mathtt{BCE}\}\,.$$

Comme ADE *est un sur ensemble de* $\mathtt{DE} \in \mathsf{InFreqLibre}(\gamma, bd)$*, il n'est pas fréquent. Par contre,* ABC *n'est pas dans ce cas, il est donc fréquent. Pour calculer sa fréquence, il faut considérer ses sous ensembles libres fréquents. Celui qui a la fréquence la plus faible est* BC*, ce qui donne la fréquence de* ABC*.*

Il existe une dernière représentation condensée que l'on peut construire à partir des libres :

$$\mathsf{Res}(bd, \mathcal{C}_{\mathsf{libre}} \wedge \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, (\mathsf{freq}, \mathsf{ferm})). \tag{3.3}$$

Dans ce cas, en plus de la fréquence des itemsets libres, on stocke aussi leur fermeture. La connaissance de cette fermeture permet de se passer de la collection $\mathsf{InFreqLibre}(\gamma, bd)$.

Proposition 21 *Soit S un itemset, S est fréquent si et seulement si il existe un itemset fréquent libre L tel que $L \subseteq S \subseteq \mathsf{ferm}(L)$. Dans ce cas, la fréquence de S est égale à celle de L.*

Preuve : Cette proposition se démontre sans difficultés. Soit S un itemset, il existe un itemset libre L dans la même classe d'équivalence que S et qui est contenu dans S, alors il est clair que $S \subseteq \mathsf{ferm}(L)$ et que S et L ont même fréquence. La propriété s'en déduit immédiatement. □

Exemple 29 *Dans la base de données bd_1 de la figure 3.1 page 83,*

$$\mathsf{Res}(bd, \mathcal{C}_{\mathsf{libre}} \wedge \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, (\mathsf{freq}, \mathsf{ferm})) = \{(\emptyset, 6), (\mathtt{A}, 4), (\mathtt{B}^{\mathtt{A}}, 3), (\mathtt{C}, 5),$$
$$(\mathtt{D}^{\mathtt{C}}, 4), (\mathtt{E}, 3), (\mathtt{AE}^{\mathtt{B}}, 2), (\mathtt{BE}^{\mathtt{A}}, 2), (\mathtt{BC}^{\mathtt{AD}}, 2), (\mathtt{BD}^{\mathtt{AC}}, 2), (\mathtt{CE}, 2), (\mathtt{AC}^{\mathtt{D}}, 3), (\mathtt{AD}^{\mathtt{C}}, 3)\}.$$

On note $\mathtt{BC}^{\mathtt{AD}}$ *pour indiquer que* BC *est libre et que sa fermeture est* ABCD *(on met en exposant* $\mathsf{ferm}(\mathtt{BC}) \setminus \mathtt{BC}$*).*

3.1.5 Règles non redondantes

Les itemsets fermés et libres ont été utilisés dans [PBTL99a, PBTL99b, PBTL99c] pour extraire des règles d'association non redondantes et pour construire une représentation condensée des règles d'association.

Une fois que les itemsets libres ou fermés ont été extraits, nous avons vu qu'il est possible de régénérer les itemsets fréquents. Ceux-ci peuvent ensuite servir à générer des règles d'association. Cependant, le nombre de règles générées est souvent très important. De plus, de nombreuses règles parmi celles-ci sont inintéressantes [TKR+95, BPT+00] car elles n'apportent pas d'information supplémentaire par rapport à d'autres règles.

Exemple 30 *Dans la base de données bd_1, les règles suivantes ont une confiance de 1 et un support de* 0.33*.*

$$\texttt{ABD} \Rightarrow \texttt{C}$$
$$\texttt{BCD} \Rightarrow \texttt{A}$$
$$\texttt{BD} \Rightarrow \texttt{AC}$$

Les deux premières règles sont redondantes par rapport à la dernière. En effet, elles peuvent être déduites de celle-ci.

La thèse de N. Pasquier [Pas00] présente une technique utilisant les libres et les fermés pour extraire une base de règles d'association non redondantes que nous présentons succinctement. Les règles qui appartiennent à cette base de règles sont appelées les *règles informatives réduites* dans [BTP+02].

Les règles exactes (i.e., de confiance 1) et les autres règles, appelées règles approximatives, sont traitées séparément. En ce qui concerne les règles exactes, seules les règles ayant un corps minimal et une tête maximale (pour l'inclusion) sont générées. En effet, toutes les autres règles exactes peuvent être générées à partir de ces règles informatives exactes en utilisant deux types de transformation : soit en retirant un item de la tête de la règle ; soit en faisant passer un item de la tête de la règle vers le corps. Par exemple, avec la règle $\texttt{BC} \Rightarrow \texttt{AD}$, on obtient les règles $\texttt{BC} \Rightarrow \texttt{D}$, $\texttt{BC} \Rightarrow \texttt{A}$, $\texttt{ABC} \Rightarrow \texttt{D}$ et $\texttt{BCD} \Rightarrow \texttt{A}$. La génération de ces règles est très simple à partir des fermés et des libres. L'ensemble des règles informatives exacte est exactement l'ensemble des règles de la forme $L \Rightarrow (\mathsf{ferm}(L) \setminus L)$ où L est un itemset libre.

Les règles informatives approximatives réduites peuvent aussi être générées à partir des libres et des fermés. Ce sont les règles de la forme $L \Rightarrow (F \setminus L)$ où L est un libre et F est un fermé contenant L et couvrant $\mathsf{ferm}(L)$, c'est-à-dire $\mathsf{ferm}(L) \subset F$ et il n'existe pas de fermé entre $\mathsf{ferm}(L)$ et F.

L'ensemble des règles informatives réduites de fréquence supérieure à γ et de confiance supérieure à θ est une représentation condensée de l'ensemble des règles valides. Ces dernières peuvent en effet être générées, avec leur fréquence et leur confiance, à partir des règles informatives réduites. Se référer à [Pas00] pour les détails.

Exemple 31 *Dans la base de données bd_1 de la figure 3.1 page 83, il y a 16 règles informatives exactes (autant que d'itemsets libres), par exemple* $\texttt{BC} \Rightarrow \texttt{AD}$, $\texttt{BD} \Rightarrow \texttt{AC}$ *ou* $\texttt{AC} \Rightarrow \texttt{D}$.

Les règles $\texttt{BC} \Rightarrow \texttt{ADE}$ *et* $\texttt{D} \Rightarrow \texttt{AC}$ *sont des règles informatives approximatives réduites. Par contre, la règle* $\texttt{B} \Rightarrow \texttt{ACDE}$ *ne l'est pas. En effet, elle est de la forme* $\texttt{B} \Rightarrow (\texttt{ABCDE} \setminus \texttt{B})$ *où* ABCDE *est un fermé qui contient bien* B *mais qui ne couvre pas* $\mathsf{ferm}(\texttt{B}) = \texttt{AB}$ *(voir la figure 3.2 page 84).*

3.1.6 Les itemsets δ-libres

Nous allons maintenant présenter une autre représentation condensée basée sur une généralisation des itemsets libres : les itemsets δ-libres [BBR00].

Si un itemset S n'est pas libre alors cela signifie qu'il existe une règle exacte $X \Rightarrow (S \setminus X)$. Cette règle indique que la fréquence de S est la même que celle de X et que par conséquent il est inutile de la recalculer. Ainsi, il n'est nécessaire de calculer la fréquence que des itemsets dont la fréquence ne peut pas être déduite par une telle règle exacte, c'est-à-dire les itemsets libres.

On peut généraliser ce principe pour calculer la fréquence d'encore moins d'itemsets en utilisant des règles qui ne sont pas exactes mais dont la fréquence est très proche de 1, les règles δ-fortes.

Définition 27 (règle δ-forte) *Soit δ un entier positif, une règle d'association est* δ-forte *dans une base de données binaire bd si elle admet au plus δ exceptions dans bd.*

Nous rappelons qu'une exception à une règle $X \Rightarrow Y$ est une transaction de la base de données qui contient X mais pas Y. On peut maintenant introduire la définition des itemsets δ-libres.

Définition 28 (δ-libre [BBR00], $\mathcal{C}_{\delta\text{-libre}}$) *Soit δ un entier positif, un itemset S est* δ-libre *dans une base de données binaire bd si il n'existe pas de règle δ-forte $X \Rightarrow Y$, $Y \neq \emptyset$ dans bd telle que $X \cup Y \subseteq S$. On définit ensuite la contrainte $\mathcal{C}_{\delta\text{-libre}}$ de la manière suivante*

$$\mathcal{C}_{\delta\text{-libre}}(S) = (S \text{ est } \delta\text{-libre}).$$

On constate que cette définition est une généralisation de la caractérisation des ensembles libres par le point (iv) de la proposition 15 page 83 si on remplace "règle exacte" par "règle δ-forte".

On peut aussi caractériser les itemsets δ-libres au moyen de la δ-fermeture de manière similaires aux itemsets libres (proposition 15).

Définition 29 (δ-fermeture) *Soit δ un entier positif. La* δ-fermeture *d'un itemset S (notée $\mathsf{ferm}_\delta(S)$) est définie par :*

$$\mathsf{ferm}_\delta(S) = \{I \in \mathsf{Items} \mid \mathsf{freq}_a(S) - \mathsf{freq}_a(S \cup \{I\}) \leq \delta\}$$

(freq_a est la fréquence absolue).

Proposition 22 *Les propositions suivantes sont équivalentes :*
(i) S est un itemset δ-libre ;
(ii) si $T \subset S$, alors $\mathsf{freq}(T) - \mathsf{freq}(S) > \delta$;
(iii) S n'est pas inclus dans la δ-fermeture d'un de ses sous ensembles stricts.

Si δ vaut 0, alors la δ-fermeture est exactement la fermeture et les δ-libres sont exactement les libres. Si $\delta > \mu$, alors tout itemset δ-libre est également μ-libre. Si on note $\mathsf{FreqLibre}_\delta(\gamma, bd)$ l'ensemble des itemsets δ-libres γ-fréquents, alors cela signifie que pour tout γ,

$$\mathsf{FreqLibre}_\delta(\gamma, bd) \subseteq \mathsf{FreqLibre}_\mu(\gamma, bd).$$

Exemple 32 *Dans la base de données bd_1 de la figure 3.1 page 83, il y a 13 itemsets libres 2-fréquents (voir l'exemple 28). Lorsque δ est différent de 0, on a :*

$$\begin{aligned} \mathsf{FreqLibre}_1(\gamma, bd) &= \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \emptyset\}, \\ \mathsf{FreqLibre}_2(\gamma, bd) &= \{\mathtt{B}, \mathtt{E}, \emptyset\}, \\ \textit{et pour } \delta > 2,\ \mathsf{FreqLibre}_\delta(\gamma, bd) &= \{\emptyset\}. \end{aligned}$$

Tout comme la contrainte $\mathcal{C}_{\mathsf{libre}}$, la contrainte $\mathcal{C}_{\delta\text{-libre}}$ est anti-monotone, ce qui permet de l'utiliser dans un algorithme similaire à Close, l'algorithme Min-Ex [BBR00]. Comme Close, l'algorithme Min-Ex fait un parcours niveau par niveau du treillis des itemsets. Lors de la passe sur les données, il calcule la δ-fermeture de chaque candidat en plus de la fréquence. Il utilise ensuite celle-ci pour déterminer quels sont les itemsets qui ne sont pas δ-libres en utilisant la caractérisation (iii) de la proposition 22. La différence entre Min-Ex et Close est donc limitée au fait que lors de la passe sur les données, Min-Ex calcule la δ-fermeture au lieu de la fermeture.

Tout comme pour les itemsets libres, on peut se baser sur les itemsets δ-libres pour concevoir une représentation condensée. Dans ce cas, il est aussi nécessaire d'utiliser la frontière négative des itemsets δ-libres. On note $\mathsf{InFreqLibre}^\delta(\gamma, bd)$ la collection des itemsets δ-libres non γ-fréquents dans bd et dont tous les sous ensembles sont dans $\mathsf{FreqLibre}_\delta(\gamma, bd)$, $\mathsf{InFreqLibre}^\delta(\gamma, bd) = \mathsf{Th}(\mathcal{C}_{\delta\text{-libre}}) \cap \mathcal{B}d^-(\mathsf{Th}(\mathcal{C}_{\gamma\text{-freq}} \wedge \mathcal{C}_{\delta\text{-libre}}))$. On obtient alors une représentation condensée approximative des itemsets γ-fréquents. Cela signifie que la fréquence des itemsets n'est connue qu'avec une incertitude bornée.

Proposition 23 *Soit S un itemset et bd une base de données binaire, si S est un sur ensemble d'un élément de $\mathsf{InFreqLibre}^\delta(\gamma, bd)$ alors S n'est pas fréquent. Sinon soit T un itemset δ-libre dans bd défini par*

$$\mathsf{freq}(T, bd) = \min_{\leq} \{\mathsf{freq}(L, bd) \mid L \subseteq S \wedge L \in \mathsf{FreqLibre}_\delta(\gamma, bd)\},$$

alors l'inégalité suivante est vraie (on considère des fréquences absolues) :

$$\mathsf{freq}(T, bd) \geq \mathsf{freq}(S, bd) \geq \mathsf{freq}(T, bd) - \delta(|S| - |T|),$$

.

Si δ vaut 0, on retrouve les propositions 20 et 19, ce qui montre que les itemsets libres sont un cas particulier des itemsets δ-libres pour $\delta = 0$.

Cette proposition démontre (la preuve est dans [BBR00]) que la paire

$$\Big(\mathsf{Res}(\mathcal{C}_{\gamma\text{-freq}} \wedge \mathcal{C}_{\delta\text{-libre}}, \mathsf{freq}), \mathsf{InFreqLibre}^\delta(\gamma, bd)\Big) \tag{3.4}$$

est une représentation condensée approximative de $\mathsf{Res}(\mathcal{C}_{\gamma\text{-freq}}, \mathsf{freq})$. L'incertitude sur la fréquence des itemsets fait qu'il n'est pas toujours possible de décider si un itemset est fréquent ou non. Par exemple, si le seuil de fréquence est de 0.2 et qu'un itemset a une fréquence de 0.19 avec une incertitude de 0.02, il n'est pas possible de décider s'il est ou non fréquent. Il est nécessaire de générer aussi ces itemsets "incertains" afin d'être sûr d'obtenir un sur ensemble des itemsets fréquents.

L'erreur commise sur les fréquences des itemsets régénérés est donc bornée par $\frac{\delta}{n}(|S|-|T|)$. Le paramètre δ prend des valeurs allant de 0 à quelque centaines, et dans des bases de

données réelles, le nombre de transactions peut être très élevé (de quelques milliers à plusieurs millions). Par exemple, dans le cas d'une base contenant 10^5 lignes et avec une valeur de δ de quelques centaines, l'erreur maximale restera généralement en dessous de 1% tout en permettant une amélioration considérable de l'efficacité de l'extraction par rapport à Close. L'utilisateur est évidemment libre de fixer ce paramètre comme il le désire pour mettre l'accent sur l'efficacité de l'extraction ou la précision des résultats. Il a été montré expérimentalement dans [BBR00] que l'erreur est en pratique beaucoup plus faible. Cela est dû au fait que l'erreur indiqué dans la proposition est une borne supérieure, en moyenne cette erreur est plus faible, surtout si δ est grand.

On pourrait imaginer une représentation condensée similaire à (3.3) en utilisant les itemsets δ-libres : $\mathsf{Res}(\mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\delta\text{-}\mathsf{libre}}, (\mathsf{freq}, \mathsf{ferm}_\delta))$. Le fait que cette collection soit vraiment une représentation condensée de $\mathsf{Res}(\mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq})$ est un problème ouvert [BB00]. La différence principale avec la fermeture vient de ce que la δ-fermeture n'est pas idempotente, i.e., en général $\mathsf{ferm}_\delta(S) \neq \mathsf{ferm}_\delta(\mathsf{ferm}_\delta(S))$.

3.1.7 Les itemsets $\vee$-*libres*

Les itemsets $\vee$-*libres* sont une autre généralisation des itemsets libres qui permet de constituer une autre représentation condensée exacte des itemsets fréquents appelée DBC (*disjunction-bordered condensation*). Nous ne ferrons que l'aborder ici, le lecteur intéressé pourra se reporter à [BR01].

Pour définir les itemsets *ou*-libres, nous devons d'abord définir les règles disjonctives simples.

Définition 30 (Règles disjonctives simples) *Une* règle disjonctive simple *basée sur un itemset S est une règle de la forme $X \Rightarrow A \vee B$, où $X \subset S$ et A et B sont des items appartenant à $S \setminus X$. Cette règle sera dite valide dans une base de données binaire bd si toute transaction supportant X supporte aussi soit l'item A soit l'item B.*

Définition 31 (itemset *ou*-libre) *Un itemset S est dit ou-*libre *dans une base de données binaire bd si il n'existe pas de règle disjonctive simple basée sur S qui soit valide dans bd.*

On remarque que la définition des règles disjonctives simples n'impose pas que les items A et B soient différents. La collection des itemsets *ou*-libres est donc un sous ensemble de la collection des itemsets libres. En effet, si un itemset S n'est pas libre, cela signifie qu'il existe une règle exacte $X \Rightarrow Y$ basée sur S valide dans les données. Alors il existe aussi une règle disjonctive simple $X \Rightarrow A \vee A$ avec $A \in Y$ basée sur S valide dans les données. Ainsi S n'est pas non plus un itemset *ou*-libres.

Le fait d'être un itemset *ou*-libre est encore une propriété anti-monotone. Cela signifie qu'il est encore possible d'extraire efficacement ces derniers. Dans [BR01], les auteurs proposent deux algorithmes pour les extraire : un algorithme niveau par niveau basé sur Close et un algorithme en profondeur d'abord inspiré de Closet.

Les auteurs montrent ensuite comment obtenir une représentation condensée appelée DBC. Celle-ci est constituée des itemsets *ou*-libres et des itemsets fréquents de la frontière négative de la collection des itemsets *ou*-libres. Cette représentation condensée est exacte, ce qui signifie qu'il est possible de régénérer tous les itemsets fréquents et leur fréquence à partir de celle-ci. De plus, les auteurs montrent qu'elle est plus petite que la collection des itemsets

fréquents et qu'elle est expérimentalement généralement plus petite que la collection des clos et que la représentation condensée basée sur les libres.

Dans un article récent [CG02], les itemsets *ou*-libres sont encore généralisés afin de n'extraire que les itemsets dont la fréquence ne peut pas être exactement déduite de la fréquence d'autres itemsets. Par contre, cette représentation condensée, bien qu'elle soit plus petite que les *ou*-libres et plus difficile à extraire. Comme nous nous intéressons surtout à l'optimisation de requêtes en utilisant des représentations condensées, nous n'aborderons pas cette autre représentation condensée. Dans cet article, les auteurs montrent d'ailleurs que les itemsets *ou*-libres sont un bon compromis entre la taille de la représentation condensée et l'efficacité de son extraction.

3.1.8 Discussion

Nous avons présenté dans cette section diverses représentations condensées des itemsets fréquents. Certaines sont exactes, ce qui signifie qu'on peut régénérer les itemsets fréquents et leur fréquence exactement et d'autres sont approximatives, ce qui signifie que les fréquences sont régénérées avec une certaine erreur.

Les différentes représentations que nous avons vues sont :
- les fermés :
$$\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{clos}}, \mathsf{freq}); \tag{3.1}$$
- deux représentations exactes utilisant les libres :
$$(\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{libre}}, \mathsf{freq}), \mathsf{InFreqLibre}(\gamma, bd)) \tag{3.2}$$
$$\text{et } \mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{libre}}, (\mathsf{freq}, \mathsf{ferm})); \tag{3.3}$$
- une représentation approximative utilisant les δ-libres :
$$(\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\delta\text{-}\mathsf{libre}}, \mathsf{freq}), \mathsf{InFreqLibre}^{\delta}(\gamma, bd)); \tag{3.4}$$
- et une représentation exacte utilisant les itemsets *ou*-libres.

On peut également rajouter les itemsets fréquents maximaux que nous avons vus dans le premier chapitre.

La taille de ces diverses représentations condensées est variable et dépend fortement des données, en particulier du niveau de corrélation de celles-ci. Il existe cependant plusieurs résultats permettant de les comparer qui sont rappelés dans la proposition suivante. Ces résultats sont tirés des différents articles qui traitent de ces représentations condensées.

Proposition 24 *On note :*
- $\mathsf{Freq}(\gamma, bd) = \mathsf{Th}(\mathcal{C}_{\gamma\text{-}\mathsf{freq}})$ *la collection des itemsets γ-fréquents ;*
- $\mathsf{FreqMax}(\gamma, bd) = \mathcal{B}d^{+}(\mathsf{Freq}(\gamma, bd))$ *la collection des itemsets γ-fréquents maximaux ;*
- $\mathsf{FreqFerm}(\gamma, bd) = \mathsf{Th}(\mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{clos}})$ *la collection des itemsets γ-fréquents fermés ;*
- $\mathsf{FreqLibre}(\gamma, bd) = \mathsf{Th}(\mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{libre}})$ *la collection des itemsets γ-fréquents libres ;*
- $\mathsf{FreqLibre}_{\delta}(\gamma, bd) = \mathsf{Th}(\mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\delta\text{-}\mathsf{libre}})$ *la collection des itemsets γ-fréquents δ-libres ;*
- $\mathsf{FreqDLibre}(\gamma, bd)$ *la collection des itemsets γ-fréquents ou-libres.*

Alors les propriétés suivantes sont vraies

$$\mathsf{FreqMax}(\gamma, bd) \subseteq \mathsf{FreqFerm}(\gamma, bd) \subseteq \mathsf{Freq}(\gamma, bd),$$
$$|\mathsf{FreqFerm}(\gamma, bd)| \leq |\mathsf{FreqLibre}(\gamma, bd)|,$$
$$\mathsf{FreqLibre}_{\delta}(\gamma, bd) \subseteq \mathsf{FreqLibre}_{\mu}(\gamma, bd) \subseteq \mathsf{FreqLibre}(\gamma, bd) \subseteq \mathsf{Freq}(\gamma, bd) \text{ si } \mu \leq \delta,$$
$$\mathsf{FreqDLibre}(\gamma, bd) \subseteq \mathsf{FreqLibre}(\gamma, bd).$$

Les δ-libres sont très intéressants car le paramètre δ permet de faire un compromis entre le niveau de précision souhaité (plus δ est petit, plus l'erreur est faible) et l'efficacité et la taille de la représentation (plus δ est grand et plus l'extraction et rapide et la représentation petite).

Les représentations condensées sont très utiles lorsqu'il s'agit d'extraire des itemsets ou des règles d'association dans des données très corrélées. En effet, dans ce cas, il est courant que l'extraction de tous les itemsets fréquents ne soit pas faisable en raison de leur trop grand nombre. Les représentations condensées utilisent les redondances présentes dans les données fortement corrélées pour éviter de faire des calculs de fréquence coûteux. Les résultats présentés dans les articles qui leur sont consacrés montrent que le gain obtenu en les utilisant peut être extrêmement important et peut atteindre plusieurs ordres de grandeurs. Ces études montrent également que dans le cas où les données sont faiblement corrélées, ces méthodes ne sont généralement pas plus lentes (ou alors de manière peu importante) que les algorithmes d'extraction directe des itemsets fréquents. Par conséquent, il parait indispensable d'utiliser ces représentations condensées pour optimiser l'extraction des itemsets et des règles d'association.

Dans ce cas, on utilise une étape de plus pour l'évaluation des requêtes inductives. Une représentation condensée des itemsets fréquents est extraite puis ensuite tous les itemsets fréquents sont générés à partir de cette représentation condensée. Lorsque des contraintes autres que la contrainte de fréquence sont utilisées dans la requête, elles sont alors testées sur les itemsets générés dans une dernière étape.

Clairement, il faut essayer d'utiliser les contraintes pendant l'étape d'extraction de la représentation condensée. Ainsi, on optimisera encore plus cette étape et on combinera donc les avantages des représentations condensées et de l'utilisation active des contraintes.

3.2 Extraction des representations condensées sous contraintes

Dans cette section, nous allons présenter notre nouvel algorithme d'extraction des représentations condensées sous contraintes. Nous considérerons les représentations condensées à base d'itemsets fermés et d'itemsets δ-libres.

L'extraction des δ-libres et des fermés peut être vue comme une extraction d'itemsets sous contraintes à l'aide de l'algorithme générique du chapitre 2. En fait, l'algorithme Close est une instanciation l'algorithme générique utilisé avec la contrainte $\mathcal{C}_{\gamma\text{-freq}} \wedge \mathcal{C}_{\text{libre}}$. De même, l'algorithme Min-Ex est une instanciation de l'algorithme générique avec la contrainte $\mathcal{C}_{\gamma\text{-freq}} \wedge \mathcal{C}_{\delta\text{-libre}}$. Une fois cette remarque faite, il semble intéressant d'essayer d'utiliser cet algorithme générique pour extraire des représentations condensées sous contraintes. C'est ce que nous faisons dans cette section en traitant d'abord le cas plus simple des contraintes anti-monotones. Pour pousser les contraintes monotones, nous introduirons une nouvelle représentation condensée, les itemsets libres contextuels.

3.2.1 Cas des contraintes anti-monotones

Supposons que l'on veuille évaluer la requête $\sigma = (bd, \mathcal{C}_{\mathsf{am}}, \mathsf{freq})$ où $\mathcal{C}_{\mathsf{am}}$ est une contrainte anti-monotone quelconque (par exemple $\mathcal{C}_{\mathsf{am}}(S) = \mathcal{C}_{\gamma\text{-freq}}(S, bd) \wedge (\mathtt{B} \in S)$). La stratégie d'évaluation que nous proposons ici consiste à passer par une représentation condensée de $\mathsf{Res}(\sigma)$ puis ensuite de régénérer $\mathsf{Res}(\sigma)$ à partir de celle-ci. Nous allons considérer plusieurs représentations condensées possibles en utilisant les itemsets fermés ou les itemsets δ-libres.

Nous commençons par étudier une représentation composées d'itemsets δ-libres. Considérons la requête $\sigma' = (bd, \mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\delta\text{-libre}}, \mathsf{freq})$. Nous allons montrer que la paire composée d'une part de $\mathsf{Res}(\sigma')$ et d'autre part de $\mathcal{B}d^{-}(\mathsf{Th}(\sigma')) \cap \mathsf{Th}(bd, \mathcal{C}_{\delta\text{-libre}})$ et une représentation condensée de $\mathsf{Res}(\sigma)$, c'est-à-dire que $\mathsf{Res}(\sigma)$ peut être régénéré à partir de cette représentation condensée sans accéder à la base de données bd. On aura donc une représentation condensée correspondant aux représentations (3.2) et (3.4) lorsqu'on considère une contrainte anti-monotone quelconque et plus seulement la contrainte de fréquence.

Proposition 25 *Soit $\sigma = (bd, \mathcal{C}_{\mathsf{am}}, \mathsf{freq})$ où $\mathcal{C}_{\mathsf{am}}$ est une contrainte anti-monotone et $\sigma' = (bd, \mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\delta\text{-libre}}, \mathsf{freq})$. Soit S un itemset, si il existe $T \in \mathcal{B}d^{-}(\mathsf{Th}(\sigma')) \cap \mathsf{Th}(bd, \mathcal{C}_{\delta\text{-libre}})$ tel que $T \subseteq S$ alors S n'est pas dans $\mathsf{Res}(\sigma)$. Sinon soit T un itemset δ-libre dans bd défini par*

$$\mathsf{freq}(T, bd) = \min_{\leq} \left\{ \mathsf{freq}(L, bd) \mid L \subseteq S \wedge L \in \mathsf{Th}(\sigma') \right\},$$

alors l'inégalité suivante est vraie (si on considère des fréquences relatives) :

$$\mathsf{freq}(T, bd) \geq \mathsf{freq}(S, bd) \geq \mathsf{freq}(T, bd) - \frac{\delta}{n}(|S| - |T|),$$

où n est le nombre de transactions de la base de données bd. En particulier, si $\delta = 0$, $\mathsf{freq}(S, bd) = \mathsf{freq}(T, bd)$.

Étant donné un itemset S, on est donc capable avec $\mathsf{Res}(\sigma')$ et $\mathcal{B}d^{-}(\mathsf{Th}(\sigma')) \cap \mathsf{Th}(bd, \mathcal{C}_{\mathsf{libre}})$ de déterminer si $S \in \mathsf{Th}(\sigma)$ et dans ce cas de calculer une approximation de la fréquence de S ou, si $\delta = 0$, la valeur exacte de cette fréquence. En fait, dans le cas où δ est non nul, la fréquence de S n'est connue qu'approximativement. Par conséquent, il n'est pas toujours possible de déterminer si S appartient bien à $\mathsf{Th}(\sigma)$.

Dans le cas $\delta = 0$, on peut aussi construire une représentation condensée correspondant à la représentation (3.3), il suffit de prendre $\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-freq}} \wedge \mathcal{C}_{\mathsf{libre}}, (\mathsf{freq}, \mathsf{ferm}))$.

Par contre, pour avoir une représentation à base de fermés, la solution consistant à prendre $\mathsf{Res}(bd, \mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{clos}}, \mathsf{freq})$ ne fonctionne pas. En effet, les itemsets fermés satisfaisant $\mathcal{C}_{\mathsf{am}}$ ne sont pas nécessairement suffisants pour régénérer tous les itemsets satisfaisant $\mathcal{C}_{\mathsf{am}}$. En effet, si un itemset fermé F ne satisfait pas la contrainte $\mathcal{C}_{\mathsf{am}}$, alors aucun des itemsets de sa classe d'équivalence ne pourra être régénéré. Or certains de ceux-ci peuvent satisfaire $\mathcal{C}_{\mathsf{am}}$.

Exemple 33 *Considérons la requête inductive $\sigma = (bd_1, 2^{\mathsf{Items}}, \mathcal{C}_{2\text{-freq}} \wedge \mathcal{C}, \mathsf{freq})$ où la contrainte $\mathcal{C}$ est définie par $\mathcal{C}(S) = (S \subseteq \mathtt{ABC})$. Dans ce cas, les itemsets fermés qui satisfont la contrainte $\mathcal{C}_{2\text{-freq}} \wedge \mathcal{C}$ sont* `A`, `AB`, `C` *et l'ensemble vide. Ces itemsets fermés permettent, en utilisant la propriété 18 page 90, de calculer la fréquence de leurs sous ensembles, i.e., des itemsets $\emptyset$,* `A`, `B`, `C` *et* `AB`. *Cependant, les fréquences de* `AC`, `BC` *et* `ABC` *qui satisfont la contrainte $\mathcal{C}_{2\text{-freq}} \wedge \mathcal{C}$ ne peuvent pas être calculées.*

Il ne suffit pas de prendre les itemsets fermés qui satisfont $\mathcal{C}_{\mathsf{am}}$, il faut en fait prendre toutes les fermetures des itemsets libres qui satisfont $\mathcal{C}_{\mathsf{am}}$. La représentation à base de fermés est donnée dans la proposition suivante :

Proposition 26 *Soit $\mathcal{C}_{\mathsf{am}}$ une contrainte anti-monotone. Alors la collection des itemsets fermés $\mathcal{F}$ (et de leur fréquence)*

$$\mathcal{F} = \{(\mathsf{ferm}(L), \mathsf{freq}(L)) \mid L \in \mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{libre}})\}$$

est une représentation condensée de $\mathcal{S} = \mathsf{Res}(bd, \mathcal{C}_{\mathsf{am}}, \mathsf{freq})$. En particulier, on peut calculer la fréquence d'un itemset de $\mathcal{S}$ à partir de la collection $\mathcal{F}$ en utilisant la proposition 18 page 90.

Dans ce cas, certains des itemsets fermés ne satisfont pas la contrainte $\mathcal{C}_{\mathsf{am}}$ mais ils sont nécessaire pour pouvoir calculer la fréquence de certains de leurs sous ensembles qui eux satisfont la contrainte $\mathcal{C}_{\mathsf{am}}$.

3.2.2 Cas des contraintes monotones

On veut maintenant évaluer une requête $\sigma = (bd, \mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}}, \mathsf{freq})$ utilisant une contrainte anti-monotone $\mathcal{C}_{\mathsf{am}}$ et une requête monotone $\mathcal{C}_{\mathsf{m}}$. Comme nous allons utiliser l'algorithme générique du chapitre 2, nous faisons ici encore une distinction entre les contraintes anti-monotones syntaxiques et non syntaxiques. Nous supposons donc que la contrainte $\mathcal{C}_{\mathsf{am}}$ s'écrit comme une conjonction $\mathcal{C}_{\mathsf{am}} = \mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_{\mathsf{am}_{ns}}$ d'une contrainte syntaxique $\mathcal{C}_{\mathsf{am}_s}$ et d'une contrainte non syntaxique $\mathcal{C}_{\mathsf{am}_{ns}}$.

Si on veut utiliser l'algorithme générique pour extraire des itemsets libres ou δ-libres qui satisfont la conjonction des contraintes $\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}}$, deux problèmes se posent :

- d'une part, ces itemsets libres (ou δ-libres) ne permettent pas de régénérer tous les itemsets de $\mathsf{Res}(\sigma)$. On se trouve dans le même type de problème qu'avec les fermés lorsqu'il y a une contrainte anti-monotone ;
- d'autre part, pour pouvoir savoir si un itemset est libre, il faut connaître la fermeture de ses sous ensembles. Or, si ceux-ci ne satisfont pas la contrainte monotone, on ne connaîtra pas leur fermeture.

Pour résoudre ces problèmes, il faudrait considérer aussi des itemsets libres qui ne satisfont pas la contrainte monotone, mais c'est justement cela que l'on veut éviter en poussant la contrainte monotone. Ces deux problèmes viennent du fait que l'on ne connait rien des itemsets qui ne satisfont pas $\mathcal{C}_{\mathsf{m}}$ et que l'on ne veut pas les traiter. Il faut donc redéfinir la notion d'itemset libre (ou δ-libre) en se limitant aux itemsets qui satisfont la contrainte monotone $\mathcal{C}_{\mathsf{m}}$. Pour cela, nous introduisons la notion d'itemset δ-libre contextuel.

Définition 32 *Soit $\mathcal{C}_{\mathsf{m}}$ une contrainte monotone et bd une base de données binaire. Un itemset S est un itemset δ-libre contextuel dans bd relativement à la contrainte $\mathcal{C}_{\mathsf{m}}$ si il n'est inclus dans la δ-fermeture d'aucun de ses sous ensembles stricts satisfaisant $\mathcal{C}_{\mathsf{m}}$. On définit ensuite la contrainte $\mathcal{C}_{\delta-\mathsf{libre\text{-}c}}$ comme étant satisfaite par les itemsets δ-libres contextuels. Dans le cas où δ vaut 0, on notera cette contrainte $\mathcal{C}_{\mathsf{libre\text{-}c}}$.*

On constate qu'avec cette définition, il suffit de connaître la δ-fermeture des sous ensembles de S qui satisfont $\mathcal{C}_{\mathsf{m}}$ pour pouvoir déterminer si S est δ-libre contextuel. Dans le cas $\delta = 0$, les itemsets libres contextuels sont les itemsets minimaux des restrictions à $\mathsf{Th}(\mathcal{C}_{\mathsf{m}})$ des classes d'équivalence définies par la relation $\sim_f$.

Exemple 34 *Soit $\mathcal{C}_{\mathsf{m}}$ la contrainte monotone définie par $\mathcal{C}_{\mathsf{m}}(S) = (\mathtt{A} \in S)$. La figure 3.5 montre les restrictions à $\mathsf{Th}(\mathcal{C}_{\mathsf{m}})$ des classes d'équivalence définies par $\sim_f$ dans les données de la figure 3.1. Les itemsets libres contextuels sont les itemsets minimaux de ces restrictions, c'est-à-dire :*

$$\mathsf{Th}(bd_1, \mathcal{C}_{\mathsf{libre\text{-}c}} \wedge \mathcal{C}_{\mathsf{m}}) = \{\mathtt{A}, \mathtt{AB}, \mathtt{AC}, \mathtt{AD}, \mathtt{AE}, \mathtt{ABC}, \mathtt{ABD}, \mathtt{ABE}, \mathtt{ACD}, \mathtt{ACE}, \mathtt{ADE}\}\,.$$

Ceci nous permet de régler le premier problème. Il faut ensuite vérifier que l'on va bien être capable de régénérer tous les itemsets de $\mathsf{Th}(\sigma)$ et leurs fréquences à partir de $\mathsf{Th}(\sigma')$, $\sigma' = (\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}}, \mathsf{freq})$. Pour cela, nous allons adapter la proposition 23 page 95 en nous plaçant dans l'espace $\mathcal{E} = \mathsf{Th}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_{\mathsf{m}})$ à la place de 2^{Items}. Nous définissons la frontière négative d'une collection d'itemsets dans une partie de 2^{Items}.

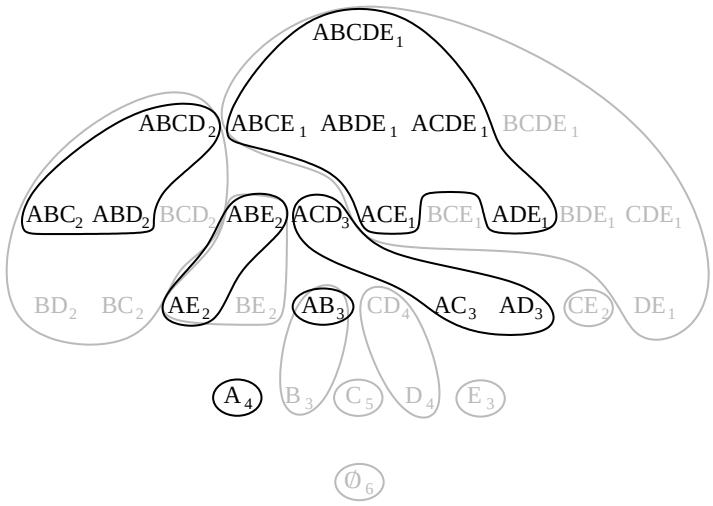


FIG. 3.5 – Les restrictions à $\mathsf{Th}(\mathcal{C}_\mathsf{m})$ des classes d'équivalence définies par $\sim_f$ avec $\mathcal{C}_\mathsf{m}(S) = (\mathtt{A} \in S)$.

Définition 33 (frontière négative dans $\mathcal{E}$) *La frontière négative d'une collection d'itemsets $\mathcal{S}$ dans une partie $\mathcal{E}$ de 2^{Items} est définie par :*

$$\mathcal{B}d^-_\mathcal{E}(\mathcal{S}) = \min_\subseteq(\mathcal{E} \setminus \mathcal{S}).$$

Lorsque $\mathcal{E} = 2^{\mathsf{Items}}$, on retrouve la définition 14 page 33 de la frontière négative.

Proposition 27 *Soit $\sigma = (bd, \mathcal{C}_\mathsf{am} \wedge \mathcal{C}_\mathsf{m}, \mathsf{freq})$ une requête inductive où $\mathcal{C}_\mathsf{am} = \mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_{\mathsf{am}_{ns}}$ est une contrainte anti-monotone conjonction d'une contrainte syntaxique et d'une contrainte non syntaxique, $\mathcal{C}_\mathsf{m}$ est une contrainte monotone et $\sigma' = (bd, \mathcal{C}_\mathsf{am} \wedge \mathcal{C}_\mathsf{m} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}}, \mathsf{freq})$. Soit S un itemset de $\mathcal{E} = \mathsf{Th}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_\mathsf{m})$, s'il existe $T \in \mathcal{B}d^-_\mathcal{E}(\mathsf{Th}(\mathcal{C}_{\mathsf{am}_{ns}} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}})) \cap \mathsf{Th}(\mathcal{C}_{\delta-\mathsf{libre\text{-}c}})$ tel que $T \subseteq S$ alors S n'est pas dans $\mathsf{Res}(\sigma)$. Sinon soit T un itemset δ-libre contextuel dans bd relativement à $\mathcal{C}_\mathsf{m}$ défini par*

$$\mathsf{freq}(T, bd) = \min_\leq \left\{ \mathsf{freq}(L, bd) \mid L \subseteq S \wedge L \in \mathsf{Th}(\sigma') \right\},$$

alors l'inégalité suivante est vraie (si on considère des fréquences relatives) :

$$\mathsf{freq}(T, bd) \geq \mathsf{freq}(S, bd) \geq \mathsf{freq}(T, bd) - \frac{\delta}{n}(|S| - |T|),$$

où n est le nombre de transactions de la base de données bd. En particulier, lorsque δ est nul, $\mathsf{freq}(S, bd) = \mathsf{freq}(T, bd)$.

Pour démontrer cette proposition, nous allons utiliser un lemme :

Lemme 1 *Si S est un itemset minimal pour l'inclusion dans $\mathcal{E} = \mathsf{Th}(\mathcal{C}_\mathsf{m} \wedge \mathcal{C}_\mathsf{am})$ où $\mathcal{C}_\mathsf{am}$ est une contrainte anti-monotone et $\mathcal{C}_\mathsf{m}$ une contrainte monotone, alors S est un itemset δ-libre contextuel relativement à $\mathcal{C}_\mathsf{m}$.*

Preuve : Soit S un itemset minimal dans $\mathcal{E}$, alors aucun sous ensemble de S ne peut satisfaire $\mathcal{C}_\mathsf{m}$ et donc, par définition des δ-libres contextuels, S est δ-libre contextuel. En effet, sinon il existe $T \subseteq S$ qui satisfait $\mathcal{C}_\mathsf{m}$. Comme S est dans $\mathcal{E}$, il satisfait $\mathcal{C}_\mathsf{am}$ et donc T aussi. Par conséquent T est dans $\mathcal{E}$ et donc S n'est pas minimal dans $\mathcal{E}$ ce qui est absurde. □

Maintenant, la preuve de la proposition :

Preuve : On note $\mathcal{F}_1(S)$ la formule
"il existe $T \in \mathcal{B}d^-_{\mathcal{E}}(\mathsf{Th}(\mathcal{C}_{\mathsf{am}_{ns}} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}})) \cap \mathsf{Th}(\mathcal{C}_{\delta-\mathsf{libre\text{-}c}})$ tel que $T \subseteq S$",
$\mathcal{F}_2(S,T)$ la formule "T est δ-libre contextuel et

$$\mathsf{freq}(T, bd) = \min_{\leq} \left\{ \mathsf{freq}(L, bd) \mid L \subseteq S \wedge L \in \mathsf{Th}(\sigma') \right\} ",$$

et $\mathcal{I}(S,T)$ la formule

$$\mathsf{freq}(T, bd) \geq \mathsf{freq}(S, bd) \geq \mathsf{freq}(T, bd) - \frac{\delta}{n}(|S| - |T|).$$

On définit la propriété $\mathcal{P}(S)$ par "si $\mathcal{F}_1(S)$ alors $S \notin \mathsf{Th}(\sigma)$, sinon il existe T tel que $\mathcal{F}_2(S,T)$ et l'inégalité $\mathcal{I}(S,T)$ est vraie". La proposition s'énonce alors "si $S \in \mathcal{E}$ alors $\mathcal{P}(S)$". Nous montrons la proposition par récurrence sur $|S|$.

- Soit S minimal dans $\mathcal{E}$, montrons que $\mathcal{P}(S)$. Si $\mathcal{F}_1(S)$ est vraie, cela signifie que S ne satisfait pas $\mathcal{C}_{\mathsf{am}_{ns}}$ et donc S n'est pas dans $\mathsf{Th}(\sigma)$. Si $\mathcal{F}_1(S)$ est fausse alors $\mathcal{F}_2(S,S)$ est vraie (car d'après le lemme S est δ-libre contextuel) et donc $\mathcal{I}(S,S)$ aussi.
- Soit $S \in \mathcal{E}$ tel que tous les sous ensembles de S satisfassent la proposition. Montrons que S la satisfait également. Si $\mathcal{F}_1(S)$ est vraie, alors $\mathcal{C}_{\mathsf{am}_{ns}}(S)$ est faux et donc $S \notin \mathsf{Th}(\sigma)$. Si $\mathcal{F}_1(S)$ est fausse et S est δ-libre contextuel, alors $S \in \mathsf{Th}(\sigma')$ et donc $\mathcal{F}_2(S,S)$ et $\mathcal{I}(S,S)$ sont vraies. Si maintenant $\mathcal{F}_1(S)$ est fausse et S n'est pas δ-libre contextuel alors il existe un item $i \in S$ tel que la règle $S \setminus \{i\} \Rightarrow i$ ait moins de δ exceptions. Nous appliquons l'hypothèse de récurrence à $S \setminus \{i\}$. Comme $\mathcal{F}_1(S)$ est fausse alors $\mathcal{F}_1(S \setminus i)$ aussi. Par conséquent, il existe T tel que $\mathcal{F}_2(S \setminus i, T)$ et $\mathcal{I}(S \setminus i, T)$. La règle $S \setminus \{i\} \Rightarrow i$ a moins de δ exceptions, par conséquent

$$\mathsf{freq}(S) \geq \mathsf{freq}(S \setminus i) - \delta/n \geq \mathsf{freq}(T) - \frac{\delta}{n}(|S| - |T|) - \frac{\delta}{n}$$

et donc $\mathcal{I}(S,T)$ est vraie. Il est possible que $\mathcal{F}_2(S,T)$ ne soit pas vraie, mais dans ce cas, si $\mathcal{F}_2(S,T')$ est vraie alors $\mathsf{freq}(T) \geq \mathsf{freq}(T')$ et donc $\mathcal{I}(S,T')$ est vraie.

□

Pour que cette proposition soit applicable, il faut que l'itemset S soit dans $\mathcal{E}$. Si la contrainte $\mathcal{C}_{\mathsf{m}}$ est syntaxique, ceci peut se vérifier sans accès aux données, sinon il suffit de remarquer qu'un itemset de $\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}})$ est nécessairement un sur ensemble d'un itemset minimal de $\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}})$. Ces derniers sont des itemsets libres contextuels et sont donc dans la collection $\mathsf{Res}(\sigma')$. Ainsi, même si $\mathcal{C}_{\mathsf{m}}$ n'est pas une contrainte syntaxique, il est possible de tester si $S \in \mathcal{E}$ sans accès aux données, en utilisant $\mathsf{Res}(\sigma')$. La paire

$$\left(\mathsf{Res}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}}, \mathsf{freq}), \mathcal{B}d^-_{\mathcal{E}}(\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}})) \cap \mathsf{Th}(\mathcal{C}_{\delta-\mathsf{libre\text{-}c}})\right) \tag{3.5}$$

et donc bien une représentation condensée de $\mathsf{Res}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}}, \mathsf{freq})$ et elle correspond aux représentations (3.4) et (3.2) (si $\delta = 0$).

Dans le cas où $\delta = 0$, il existe une autre représentation condensée qui correspond à la représentation (3.3) :

$$\mathsf{Res}(bd, \mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\mathsf{libre\text{-}c}}, (\mathsf{freq}, \mathsf{ferm})). \tag{3.6}$$

Dans cette représentation, on garde les libres contextuels et leurs fermetures (en plus de leur fréquence). Ainsi, il n'est pas nécessaire de conserver la collection $\mathcal{B}d^-(\mathsf{Th}(\sigma')) \cap \mathsf{Th}(\mathcal{C}_{\delta-\mathsf{libre\text{-}c}})$,

il suffit de régénérer les itemsets S tels qu'il existe un libre contextuel L tel que $L \subseteq S \subseteq \mathsf{ferm}(L)$.

Il faut remarquer que les nouvelles représentations condensées que nous avons présentées ici permettent de régénérer un sur ensemble de $\mathsf{Res}(\sigma)$. Il faut ensuite éliminer les itemsets qui ne satisfont pas $\mathcal{C}_{\mathsf{am}}$ (ceci peut se faire sans accès à la base de données car leur fréquence est déjà connue). Dans le cas ou $\delta \neq 0$, il est possible qu'à cause de l'incertitude sur les fréquences il ne soit pas possible de vérifier si un itemset satisfait $\mathcal{C}_{\mathsf{am}}$. Dans ce cas, il faut garder ces itemsets pour pouvoir garantir la complétude.

Nous présentons maintenant notre algorithme CoCo qui extrait ces représentations condensées sous contraintes. Cet algorithme est basé sur notre algorithme générique du chapitre 2 (section 2.6.2 page 70). Les entiers n_min et n_max sont calculés de la même manière que dans l'algorithme générique. Comme dans l'algorithme Close, les itemsets sont stockés dans un enregistrement comportant trois champs :

- item pour stocker les items ;
- freq pour stocker la fréquence ;
- et ferm_δ pour la δ-fermeture. On stocke en fait une liste de couples (I, n_I) où I est un item de $\mathsf{ferm}_\delta(S) \setminus S$ et n_I indique le nombre de lignes de la base de donnée qui supportent S et pas I. Par définition de la δ-fermeture, n_I est nécessairement inférieur ou égal à δ.

Algorithme 6 (Algorithme CoCo)

Entrée : Une contrainte de la forme $\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_{\mathsf{am}_{ns}} \wedge \mathcal{C}_{\mathsf{m}}$, un entier δ
Sortie : $\mathsf{Res}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}}, (\mathsf{freq}, \mathsf{ferm}_\delta))$ et $\mathcal{B}d^-_{\mathcal{E}}(\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}})) \cap \mathsf{Th}(\mathcal{C}_{\delta-\mathsf{libre\text{-}c}})$

1 $k := n\_min$
2 $\mathcal{F}_1 := \{i \in \mathsf{Items} \mid \mathcal{C}_{\mathsf{am}}(i)\}$
3 $\mathsf{Cand} := \mathbf{gen\_cand\_coco}(k, \emptyset)$
4 $\mathsf{Cand} := \{S \in \mathsf{Cand} \mid \mathcal{C}_{\mathsf{am}_s}(S) = vrai\}$ /* test de $\mathcal{C}_{\mathsf{am}_s}$ */
5 **tant que** $\mathsf{Cand} \neq \emptyset$ **ou** $k \leq n\_max$ **faire**
6 $\quad \mathsf{Cand} := \mathbf{passe\_bd\_coco}(\mathsf{Cand}, bd)$ /* calcul de $S.\mathsf{ferm}_\delta$ et $S.\mathsf{freq}$ pour $S \in \mathsf{Cand}$ */
7 $\quad \mathcal{B}_k := \{S \in \mathsf{Cand} \mid \mathcal{C}_{\mathsf{am}_{ns}}(S) = faux\}$ /* test de $\mathcal{C}_{\mathsf{am}_{ns}}$*/
8 $\quad \mathcal{S}_k := \mathsf{Cand} \setminus \mathcal{B}_k$
9 $\quad \mathsf{Cand} := \mathbf{gen\_cand\_coco}(k+1, \mathcal{S}_k)$ /* génération des candidats */
10 $\quad \mathsf{Cand} := \{S \in \mathsf{Cand} \mid \mathcal{C}_{\mathsf{am}_s}(S) = vrai\}$ /* test de $\mathcal{C}_{\mathsf{am}_s}$ */
11 $\quad \mathsf{Cand} := \mathbf{élagage\_coco}(\mathsf{Cand}, \mathcal{S}_k)$ /* élagage et test de $\mathcal{C}_{\delta-\mathsf{libre\text{-}c}}$ */
12 $\quad k := k + 1$
13 **fait**
14 **retourne** $\bigcup_{0 \leq i \leq k} \mathcal{S}_i$ et $\bigcup_{0 \leq i \leq k} \mathcal{B}_i$

La fréquence et la δ-fermeture des itemsets sont calculées lors d'une passe sur les données. Pour calculer la δ-fermeture d'un itemset S, on calcule le nombre de lignes de la base de données qui supportent S et pas I, pour chaque item I qui n'est pas dans S. Dès que ce nombre devient strictement supérieur à δ, alors l'item I ne peut pas être dans la δ-fermeture de S, il est donc supprimé de la liste $S.\mathsf{ferm}_\delta$. À la fin de la passe sur les données, les items qui sont encore dans cette liste sont exactement les items de la δ-fermeture de S.

passe_bd_coco(Cand, bd)

```
1  pour toute transaction t ∈ bd faire
2     pour tout itemset candidat S ∈ Cand tel que S.item ⊆ t faire
3        S.freq := S.freq + 1
4        si S.freq = 1 alors
5           pour tout item I ∈ t \ S.item faire
6              S.ferm_δ := S.ferm_δ # (I, 0)        /* (I, 0) est ajouté à la liste S.ferm_δ */
7           fait
8        sinon
9           pour tout couple (I, n_I) dans la liste S.ferm_δ faire
10             si I ∉ t alors
11                si n_I = δ alors
12                   retirer (I, n_I) de la liste S.ferm_δ
13                sinon
14                   n_I := n_I + 1
15                finsi
16             finsi
17          fait
18       finsi
19    fait
20 fait
```

La fonction de génération des candidats **gen_cand_coco** est très similaire à la fonction utilisée dans l'algorithme générique. La seule différence est à la ligne 8 : les itemsets ont un champ de plus pour stocker la δ-fermeture qui est initialisé avec la liste vide.

```
gen_cand_coco(k, S) /* fonction de génération des candidats du niveau k */
1  C := ∅
2  pour tout S ∈ S faire
3     pour tout i ∈ F_1 faire
4        C := C ∪ {S.item ∪ {i}}
5     fait
6  fait
7  C := C ∪ (Bd⁻(Th(¬C_m)) ∩ Items_k)
8  retourne {(S, 0, [ ]) | S ∈ C}
```

Tous les itemsets produits par cette fonction satisfont la contrainte $\mathcal{C}_{\mathsf{m}}$. Ensuite, à la ligne 10 de l'algorithme CoCo, les itemsets qui ne satisfont pas $\mathcal{C}_{\mathsf{am}_s}$ sont éliminés. Après ce filtrage, Cand ne contient plus que des itemsets appartenant à $\mathcal{E} = \mathsf{Th}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_{\mathsf{m}})$. L'élagage et le test de la contrainte $\mathcal{C}_{\delta-\mathsf{libre\text{-}c}}$ sont faits par la fonction **élagage_coco**.

```
élagage_coco(Cand, S_k)        /* Cand contient les candidats du niveau k+1 et
     S_k l'ensemble des itemsets du niveau k satisfaisant la contrainte C_am ∧ C_m ∧ C_δ−libre-c */
1  pour tout S ∈ Cand faire
2     pour tout sous ensemble T de S.item de taille k faire
3        si C_m(T) = vrai et (T ∉ S_k.item ou S.item ⊆ ferm_δ(T)) alors
4           Cand := Cand \ {S}
5     fait
7  fait
8  retourne Cand
```

Après cette fonction, Cand contient donc des itemsets libres contextuels qui satisfont $\mathcal{C}_{\mathsf{am}_s}$ et dont tous les sous ensembles qui satisfont $\mathcal{C}_{\mathsf{m}}$ sont dans $\mathsf{Th}(\sigma')$, c'est-à-dire exactement les itemsets libres contextuels minimums de la partie de $\mathsf{Th}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_{\mathsf{m}})$ qui n'a pas encore été explorée.

La propriété suivante donne la correction et la complétude de notre algorithme.

Proposition 28 *L'algorithme* CoCo *est correct et complet, i.e., il calcule bien*

$$\mathsf{Res}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}}, (\mathsf{freq}, \mathsf{ferm}_\delta)) \text{ et}$$
$$\mathcal{B}d^-_{\mathcal{E}}(\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}})) \cap \mathsf{Th}(\mathcal{C}_{\delta-\mathsf{libre\text{-}c}}).$$

Preuve : Nous avons déjà montré dans le chapitre 2 que l'algorithme générique est correct et complet. Il reste donc à montrer que la collection $\bigcup_{0\leq i\leq k}\mathcal{B}_i$ retournée par l'algorithme est bien égale à $\mathcal{B}d^-_{\mathcal{E}}(\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}})) \cap \mathsf{Th}(\mathcal{C}_{\delta-\mathsf{libre\text{-}c}})$. Nous avons vu plus haut que l'ensemble des candidats, après la ligne 10 dans l'algorithme CoCo, contient les itemsets libres contextuels minimums de la partie de $\mathsf{Th}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_{\mathsf{m}})$ qui n'a pas encore été explorée. Par conséquent, ceux de ses itemsets qui satisfont $\mathcal{C}_{\mathsf{am}_{ns}}$ sont dans $\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}})$ et les autres dans $\mathcal{B}d^-_{\mathcal{E}}(\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}}))$. Comme les itemsets de $\mathcal{B}d^-_{\mathcal{E}}(\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}}))$ qui ne satisfont pas $\mathcal{C}_{\mathsf{libre\text{-}c}}$ ont déjà été éliminés par la fonction **élagage_coco**, on a bien $\bigcup_{0\leq i\leq k}\mathcal{B}_i = \mathcal{B}d^-_{\mathcal{E}}(\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}})) \cap \mathsf{Th}(\mathcal{C}_{\delta-\mathsf{libre\text{-}c}})$. □

La sortie de l'algorithme CoCo permet donc bien de construire les deux représentations condensées

$$\big(\mathsf{Res}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}}, \mathsf{freq}), \mathcal{B}d^-_{\mathcal{E}}(\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}})) \cap \mathsf{Th}(\mathcal{C}_{\delta-\mathsf{libre\text{-}c}})\big)$$
$$\text{et, quand } \delta = 0,\ \mathsf{Res}(bd, \mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\mathsf{libre\text{-}c}}, (\mathsf{freq}, \mathsf{ferm})).$$

3.3 Nouvelles stratégies d'évaluation de requêtes inductives

Ces nouvelles représentations condensées nous permettent de proposer de nouvelles stratégies d'évaluation pour les requêtes inductives portant sur les itemsets et utilisant des contraintes monotones et anti-monotones.

Pour évaluer une requête $\sigma = (bd, 2^{\mathsf{Items}}, \mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}}, \mathsf{freq}, [0, 1])$ nous proposons la stratégie suivante :

- calculer une représentation condensée de $\mathsf{Res}(\sigma)$. Ce calcul se fait en utilisant activement les contraintes anti-monotones et monotones ;
- régénérer $\mathsf{Res}(\sigma)$ à partir de la représentation condensée.

On peut utiliser une des trois représentations condensées que nous avons proposées. Les libres et la frontière (3.5), les libres et leurs fermetures (3.6) et les δ-libres et la frontière (3.5). Les deux premières sont extraites de la même manière et sont très proches. Par conséquent le choix de l'une ou de l'autre n'est pas très important. Nous verrons dans le chapitre suivant que la seconde est plus simple à utiliser lorsque l'on veut s'en servir comme cache.

La troisième représentation est approximative. Cela signifie que la fréquence des itemsets n'est connue qu'à une précision donnée. En revanche, cette représentation est plus efficace à extraire. Ainsi, elle permet de donner des résultats approximatifs dans des données et à des seuils de fréquence ou aucune autre technique n'est envisageable. Il vaut parfois mieux avoir des résultats imprécis que de ne rien avoir. Avec le paramètre δ, l'utilisateur peut faire

un compromis entre l'efficacité de l'extraction (avec un δ grand) et une bonne précision des résultats (avec un δ petit).

Les limitations de ces stratégies sont liées aux représentations condensées utilisées. Celles-ci sont conçues pour répondre à des requêtes portant sur la fréquence des itemsets. Ainsi, si l'utilisateur a besoin d'évaluer d'autres mesures sur les données et que celles-ci ne peuvent pas être réduites à des calculs de fréquence, alors ces stratégies ne sont pas applicables. Cependant, de nombreuses mesures sont calculables à partir de la fréquence des itemsets. C'est en particulier le cas de la plupart des mesures sur les règles d'association (confiance, conviction, intérêt, etc.).

3.4 Extraction de règles avec négations

Dans cette section, nous nous intéressons à l'extraction de règles d'association avec négations. Les règles d'association traditionnelles ne permettent d'exprimer que des relations positives entre les attributs, i.e., si la règle $\mathtt{AB} \Rightarrow \mathtt{D}$ (avec sa fréquence et sa confiance) exprime une relation entre la présence des attributs A et B et la présence de l'attribut D.

L'objectif des règles avec négations est d'enrichir les règles avec des informations concernant l'absence de certains attributs. Par exemple, la règle $\mathtt{AB}\neg\mathtt{C} \Rightarrow \mathtt{D}$ est une règle avec négations. Sa confiance exprime la probabilité conditionnelle que D soit vrai lorsque A, B sont vrais et que C est faux. Pour définir plus précisément les règles avec négations, nous introduisons tout d'abord les itemsets généralisés.

3.4.1 Itemsets généralisés et règles avec négations

Un itemset généralisé est un itemset qui peut contenir des attributs positifs et des attributs négatifs. Plus formellement :

Soit *bd* une base de données binaire de schéma $\mathsf{Items}^+ = \mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots$ On peut compléter cette base de données en lui rajoutant des colonnes $\overline{\mathtt{A}}, \overline{\mathtt{B}}, \overline{\mathtt{C}}, \ldots$ Chaque transaction t est complétée sur ces colonnes en donnant à $t(\overline{\mathtt{X}})$ la valeur 1 si $t(\mathtt{X})$ vaut 0 et 0 dans le cas contraire. Les items $\mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots$ sont appelés les items positifs et $\mathsf{Items}^- = \left\{\overline{\mathtt{A}}, \overline{\mathtt{B}}, \overline{\mathtt{C}}, \ldots\right\}$ les items négatifs. Un itemset généralisé est alors un sous ensemble de $\mathsf{Items} = \mathsf{Items}^+ \cup \mathsf{Items}^-$. La fréquence absolue d'un itemset généralisé est définie comme celle d'un itemset classique : c'est le nombre de transactions complétées qui supportent l'itemset.

Exemple 35 *Dire que l'itemset généralisé* $\mathtt{AB}\overline{\mathtt{C}}\mathtt{D}$ *a une fréquence relative de* 0.1 *signifie que dans* 10% *des transactions complétées, les attributs* A, B, $\overline{\mathtt{C}}$ *et* D *sont vrai, ce qui signifie également que dans 10% des transactions les attributs* A, B *et* D *sont vrai et l'attribut* C *est faux.*

On peut ensuite définir des règles d'association avec négations de manière immédiate : une règle d'association avec négations est une règle d'association $X \Rightarrow Y$ où X et Y sont des itemsets généralisés tels que $Y \neq \emptyset$ et $X \cap Y = \emptyset$.

Exemple 36 *Dire que la règle d'association* $\mathtt{AB}\overline{\mathtt{C}} \Rightarrow \mathtt{D}$ *a une fréquence de 0.1 et une confiance de 0.7 signifie que :*

- *l'itemset généralisé* $\mathtt{AB}\overline{\mathtt{C}}\mathtt{D}$ *a une fréquence relative de* 0.1 *;*
- *dans 70% des transactions où* A *et* B *sont vrais et* C *est faux, alors* D *est vrai.*

Il faut remarquer qu'une telle règle est très intéressante si sa confiance est très supérieure à la confiance de la règle $\mathtt{AB} \Rightarrow \mathtt{D}$. *L'information "*$\mathtt{C}$ *est faux" apporte dans ce cas une plus value intéressante par rapport à détermination de* $\mathtt{D}$.

Notre définition des règles avec négations autorise un nombre quelconque d'attributs négatifs dans la tête ou le corps de règle. On peut même avoir des règles ne comportant que des attributs négatifs. Les règles avec négations sont construites à partir des itemsets généralisées exactement de la même manière que les règles d'association à partir des itemsets. Dans la suite, nous nous concentrerons donc essentiellement sur l'extraction des itemsets généralisés.

Le fait de compléter les transactions avec des attributs négatifs augmente généralement considérablement la densité de la base de donnée et ceci est vrai en particulier pour les bases de données de faible densité (i.e., le nombre d'attributs vrais par transaction est faible par rapport au nombre total d'attributs). La densité de la base de données complétée est en effet de 0.5 (dans chaque transaction et pour chaque attribut $\mathtt{A}$, soit $\mathtt{A}$ est vrai soit $\overline{\mathtt{A}}$ est vrai). De plus, le nombre d'attributs a doublé par rapport à la base de données originale.

Tout ceci a pour effet de considérablement augmenter la complexité des extractions. Même les contextes traditionnellement "faciles", i.e., de très faible densité, deviennent très difficiles lorsqu'on s'intéresse à des règles avec négations. En effet, si un attribut a une fréquence très faible (ce qui est souvent le cas), la négation de cet attribut a une fréquence très forte. Cela signifie que la plupart des attributs négatifs ont des fréquences très fortes et par conséquent il y a énormément d'itemsets généralisés fréquents comportant beaucoup d'attributs négatifs.

Pour lutter contre cette complexité, les approches proposées précédemment consistent à restreindre très fortement l'ensemble des règles avec négations considérées de manière a rendre l'extraction possible [SON98]. Le principal inconvénient de ces approches est que l'utilisateur n'a pas réellement le contrôle des règles qui sont extraites. Leur forme syntaxique est figée dans l'algorithme. Notre approche [BBJ00], consiste plutôt à ne pas restreindre a priori l'ensemble des règles mais à laisser le choix à l'utilisateur de préciser par le biais de contraintes les règles qu'il souhaite extraire. De plus, nous utilisons les représentations condensées pour améliorer les temps d'extractions.

Dans les sections suivantes nous détaillons un exemple d'extraction d'itemsets généralisés dans deux bases de données. Nous montrons en particulier que l'utilisation de notre algorithme CoCo est indispensable pour obtenir des extraction en un temps raisonnable.

3.4.2 Les bases de données binaires

Nos expériences d'extraction d'itemsets généralisés ont été faites sur deux bases de données binaires. La première est la base de données *mushroom*. Cette base de données contient la description de différents champignons. Elle contient 8124 lignes décrivant chacune une espèce de champignon. Chaque ligne contient 23 attributs discrets (couleur du chapeau, du pied, odeur, présence ou non d'un anneau, etc.) qui sont encodés en utilisant 128 attributs binaires. Cela conduit à une matrice binaire contenant 8124 lignes et 128 colonnes ou chaque ligne contient 23 fois la valeur "1".

La seconde base de données est une base provenant de l'INSEE et qui recense les services disponibles pour les entreprises dans les communes françaises. Elle contient 37000 lignes, chacune décrivant une commune, et 59 colonnes, chaque colonne indiquant la présence de certains services pour les entreprises dans la commune concernée. Ces services sont par exemple, entreprises de nettoyage, de location de matériel de chantier, location de véhicules, agences

immobilières, services bancaires, etc. Cette base de données est très creuse puisque le nombre moyen de services est de 4 par commune.

Ces deux bases de données ont été complétées en rajoutant les attributs négatifs. Finalement, on obtient deux bases de données :

- pour mushroom, 8124 lignes, 256 colonnes et 128 valeurs vraies par ligne ;
- pour la base INSEE, 37000 lignes, 118 colonnes et 59 valeurs vraies par ligne.

3.4.3 Différentes stratégies

L'extraction de tous les itemsets généralisés est souvent impossible et inutile du point de vue de l'utilisateur. De nombreux itemsets généralisés ne concernent que des items négatifs et sont donc peu intéressants. Nous supposons plutôt que l'utilisateur s'intéresse à des itemsets qui contiennent au moins quelques items positifs. Par la suite, ces itemsets généralisés peuvent être utilisés pour produire des règles d'association avec négations. Le fait d'intégrer les items négatifs peut permettre d'améliorer la qualité de ces règles et de découvrir de nouvelles tendances.

Dans les expérimentations, nous utilisons donc des contraintes qui imposent que les itemsets généralisés ne contiennent pas que des items négatifs. Plus précisément, nous définissons les contraintes $\mathcal{C}_{\mathsf{am}p\mathsf{p}}$ qui sont satisfaites par les itemsets généralisés contenant au moins p items positifs. Ces contraintes sont clairement monotones.

Il faut cependant remarquer que rien dans nos algorithmes n'impose l'utilisation de ces contraintes. L'utilisateur est libre de préciser les itemsets généralisés qui l'intéressent au moyen de contraintes quelconques dès lors qu'elles sont utilisables par notre algorithme CoCo (étant donné la complexité des extraction, ces contraintes doivent cependant être suffisamment sélectives). Les contraintes $\mathcal{C}_{\mathsf{am}p\mathsf{p}}$ ne sont qu'un exemple que nous avons choisi car il semble raisonnablement réaliste.

Dans nos expériences, nous comparons différentes stratégies pour extraire des itemsets généralisés contenant au moins 3 items positifs.

- Stratégie I : ni la contrainte $\mathcal{C}_{\mathsf{am3p}}$ ni les représentations condensées ne sont utilisées pendant l'extraction. Nous utilisons en fait ici l'algorithme Apriori pour extraire tous les itemsets généralisés fréquents et ensuite la sortie d'Apriori est filtrée pour ne retenir que ceux qui satisfont $\mathcal{C}_{\mathsf{am3p}}$.
- Stratégie II : l'algorithme générique est utilisé pour pousser la contrainte $\mathcal{C}_{\mathsf{am3p}}$ pendant l'extraction (phase de génération des candidats), les représentations condensées ne sont pas utilisées. Contrairement à la stratégie I, la contrainte $\mathcal{C}_{\mathsf{am3p}}$ et utilisée activement pendant l'extraction et non pas en post-traitement.
- Stratégie III : les représentations condensées sont utilisées pour extraire les itemsets généralisés fréquents. La contrainte $\mathcal{C}_{\mathsf{am3p}}$ n'est pas utilisée pendant l'extraction. Il y a deux sous stratégies selon la représentation condensée utilisée :
  - stratégie IIIa : on extrait les itemsets libres (représentation condensée (3.2)) ;
  - stratégie IIIb : on extrait les itemsets δ-libres (représentation condensée (3.4)).
- Stratégie IV : C'est notre nouvelle stratégie. Nous utilisons notre algorithme CoCo pour pousser la contrainte $\mathcal{C}_{\mathsf{am3p}}$ pendant l'extraction des représentations condensées, il y a là encore deux sous stratégies :
  - stratégie IVa : on extrait les itemsets libres contextuels qui satisfont $\mathcal{C}_{\mathsf{am3p}}$ (représentation condensée (3.5) avec $\delta = 0$) ;
  - stratégie IVb : on extrait les itemsets δ-libres contextuels qui satisfont $\mathcal{C}_{\mathsf{am3p}}$ (représentation condensée (3.5) avec $\delta \neq 0$).

Il faut remarquer que les résultats de ces stratégies ne sont pas identiques : les stratégies I et II calculent tous les itemsets qui satisfont $\mathcal{C}_{\gamma\text{-freq}} \wedge \mathcal{C}_{\mathsf{am3p}}$, les stratégies IIIa et IIIb calculent des représentations condensées des itemsets fréquents et les stratégies IVa et IVb des représentation condensées des itemsets qui satisfont $\mathcal{C}_{\gamma\text{-freq}} \wedge \mathcal{C}_{\mathsf{am3p}}$. Cependant, si on se place du point de vue de l'utilisateur qui s'intéresse aux itemsets qui satisfont $\mathcal{C}_{\gamma\text{-freq}} \wedge \mathcal{C}_{\mathsf{am3p}}$, les sorties de ces différentes stratégies contiennent la même information (avec une incertitude pour les stratégies IIIb et IVb qui utilisent des δ-libres contextuels), on peut donc parfaitement les comparer.

3.4.4 Résultats expérimentaux

Ces expériences ont été effectuées sur un PC équipé d'un processeur Pentium III à 500 MHz avec 768 Mo de mémoire. Pour ces expériences nous avons utilisé les implémentations d'Artur Bykowski.

Les courbes de la figure 3.6 donnent les temps de calcul pour les différentes stratégies en fonction du seuil de fréquence. Sur le jeux de données mushroom, on constate que l'utilisation des représentations condensées est indispensable. En effet, les stratégies qui ne les utilisent pas (stratégies I et II) sont totalement inutilisables, ce qui signifie que même avec un seuil de fréquence de 99% les extractions ne finissaient pas en un temps raisonnable. Les stratégies IIIa et IIIb sont faisables mais ne permettent pas d'atteindre des seuils de fréquences très bas, et ceci même si on augmente la valeur de δ jusqu'à 162. Sur ces données, les stratégies IVa et IVb permettent d'atteindre des seuils de fréquence beaucoup plus bas, même si cela est fait au détriment d'une perte de précision lorsqu'on utilise les stratégies IVb.

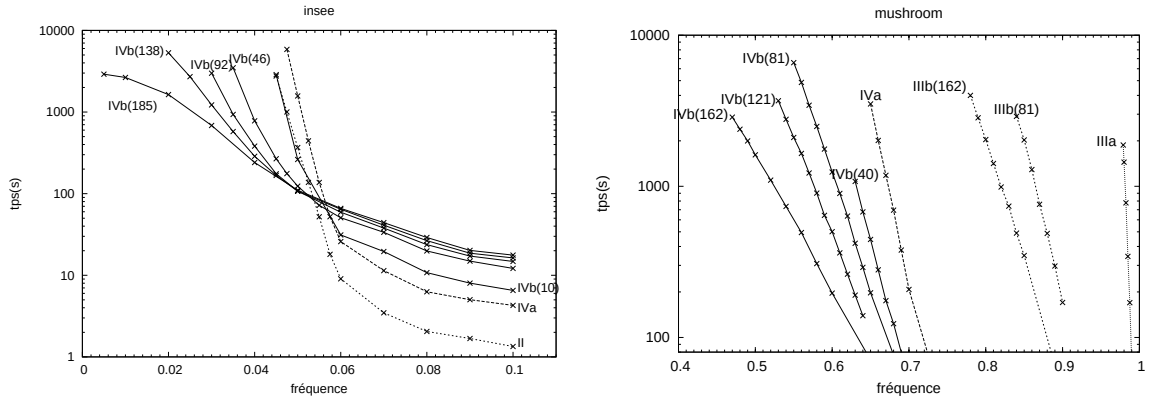


FIG. 3.6 – Extraction d'itemsets généralisés dans les données INSEE et mushroom, les valeurs de δ sont indiquées entre parenthèses.

Sur les données de l'INSEE, la situation est très différente. Le fait de pousser la contrainte $\mathcal{C}_{\mathsf{am3p}}$ a une très forte influence sur les résultats. Par contre, l'usage des représentations condensées n'apporte pas de gain pour les seuils de fréquences supérieurs à environ 0.06. En particulier, le fait d'utiliser les itemsets libres contextuels en plus de la contrainte $\mathcal{C}_{\mathsf{am3p}}$ (Stratégie IVa) n'améliore jamais les performances par rapport à la stratégie II. Quant aux stratégies IIIa et IIIb qui n'utilisent pas la contrainte $\mathcal{C}_{\mathsf{am3p}}$, elles ne permettent pas de descendre en dessous d'un seuil de 0.7 en un temps raisonnable, elles ne figurent donc pas sur le graphique. En fait, lorsqu'on regarde les sorties des algorithmes sur ces données, on s'aperçoit qu'il n'y a presque aucune règle exacte contenant au moins 3 attributs positifs.

Par conséquent, pratiquement tous les itemsets sont libres, ce qui signifie que la stratégie IVa manipule le même nombre d'itemsets que la stratégie II. Comme il faut en plus calculer les fermetures, la stratégie IVa est logiquement moins bonne que la II. Pour les seuils de fréquence les plus bas (moins de 0.06), les stratégies IVb reprennent le dessus sur la stratégie II. Elles permettent alors des gains d'efficacité très importants au prix d'un résultat moins précis.

Finalement, le fait d'ajouter les contraintes lors de l'extraction de représentations condensées ne change pas le comportement qualitatif, i.e., on retrouve une très bonne efficacité dans le cas de données fortement corrélées (de type mushroom) et de moins bons résultats dans des données faiblement corrélées (surtout lorsque $\delta = 0$) de type Insee. Ces expériences montrent que la nouvelle stratégie que nous avons présentée permet de faire des extractions en un temps raisonnable à des seuils de fréquence qui ne peuvent pas être atteints par les autres stratégies existantes.

3.4.5 Étude expérimentale de l'erreur en fonction de δ

L'utilisation des δ-libres contextuels permet un gain de performance significatif. Ce gain se fait au prix d'une imprécision lors de la régénération des fréquences des itemsets qui ne sont pas δ-libres contextuels. Nous allons ici étudier cette erreur en fonction du paramètre δ.

Pour pouvoir étudier cette erreur de manière précise, il faudrait calculer directement la fréquence de tous les itemsets et la comparer avec l'approximation que l'on peut en faire en utilisant les δ-libres. Cependant, dans les expériences que nous avons menées, calculer la fréquence de ces itemsets directement n'est pas possible étant donné leur nombre. On se trouve dans un cas ou les seules techniques disponibles sont les extractions des δ-libres. Il va donc falloir estimer l'erreur en se basant uniquement sur les δ-libres. Pour cela, nous allons tout d'abord montrer que l'on peut obtenir une meilleure approximation de l'erreur que celle donnée dans la proposition 23 (c'est à dire δ multiplié par la différence de taille entre le δ-libre et l'itemset dont on veut régénérer la fréquence). Pour cela, nous allons exploiter de manière plus complète la sortie de l'algorithme.

Nous avons vu que l'algorithme CoCo calcule la δ-fermeture de chaque itemset δ-libre. De plus, pour chaque item I de la δ-fermeture, l'algorithme calcule également la différence $n_I = \mathsf{freq}(S) - \mathsf{freq}(S \cup \{I\})$ entre la fréquence de l'itemset libre S et $S \cup \{I\}$.

Soit S un itemset libre, I_1, I_2, ... I_k les éléments de sa δ-fermeture et n_1, n_2, ... n_k les entiers définis par $n_i = \mathsf{freq}(S) - \mathsf{freq}(S \cup \{I\}_i)$. On suppose de plus que les items I_i sont ordonnés par ordre décroissant de la valeur de n_i. Considérons maintenant un itemset T contenant S et inclus dans la δ-fermeture de S. L'itemset T est donc la réunion de S et de certains des items I_i :

$$T = S \cup \{I_{i_1} I_{i_2} ... I_{i_l}\}$$

On suppose ici encore que les I_{i_j} sont ordonnés par valeur décroissante de n_{i_j}. On montre alors facilement par récurrence sur l que la fréquence de T est dans l'intervalle $[\mathsf{freq}(S) - n_{i_1}, \mathsf{freq}(S) - \sum_{1 \leq j \leq l} n_{i_j}]$. La longueur de cet intervalle vaut donc $\sum_{2 \leq j \leq l} n_{i_j}$ si $l \geq 2$ et 0 sinon. Nous définissons alors l'erreur absolue sur la fréquence de T comme la largeur de cet intervalle.

Si on considère maintenant les 2^k itemsets contenant S et inclus dans sa δ-fermeture, la

somme des erreurs absolues faites sur tous ces itemsets vaut :

$$Err = \sum_{\substack{2 \leq l \leq k \\ 1 \leq i_1 < i_2 < ... < i_l \leq k}} \left(\sum_{2 \leq j \leq l} n_{i_j} \right).$$

On montre alors que cette somme est égale à :

$$Err = 2^{l-2}(n_2 + n_3 + ... + n_l) + 2^{l-3}(n_3 + n_4 + ... + n_l) + ... + 2(n_{l-1} + n_l) + n_l.$$

Si on calcule ensuite la somme de ces erreurs pour tous les itemsets δ-libres fréquents S et que l'on divise cette somme par la somme des 2^k, on obtient une approximation de l'erreur moyenne absolue sur les itemsets fréquents. Cette valeur n'est qu'une approximation pour deux raisons :

- d'une part, un itemset T peut être pris en compte plusieurs fois dans ce calcul si il existe plusieurs itemsets libres S tels que T contienne S et soit inclus dans sa δ-fermeture ;
- d'autre part, le fait que l'on prenne ainsi bien en compte tous les itemsets fréquents T lorsque qu'on fait ce calcul avec tous les libres fréquents est encore une conjecture [BB00]. Plus précisément, savoir si, pour tout itemset T, il existe un itemset δ-libre S tel T contienne S et soit inclus dans la δ-fermeture de S est un problème ouvert.

Nous avons représenté sur la figure 3.7 cette erreur absolue moyenne calculée à partir de tous les itemsets libres contextuels fréquents en fonction de δ et du seuil de fréquence sur les deux jeux de données Mushroom et Insee. On constate bien sur ces graphiques que, dans ces deux jeux de données, l'erreur moyenne augmente lorsque δ augmente.

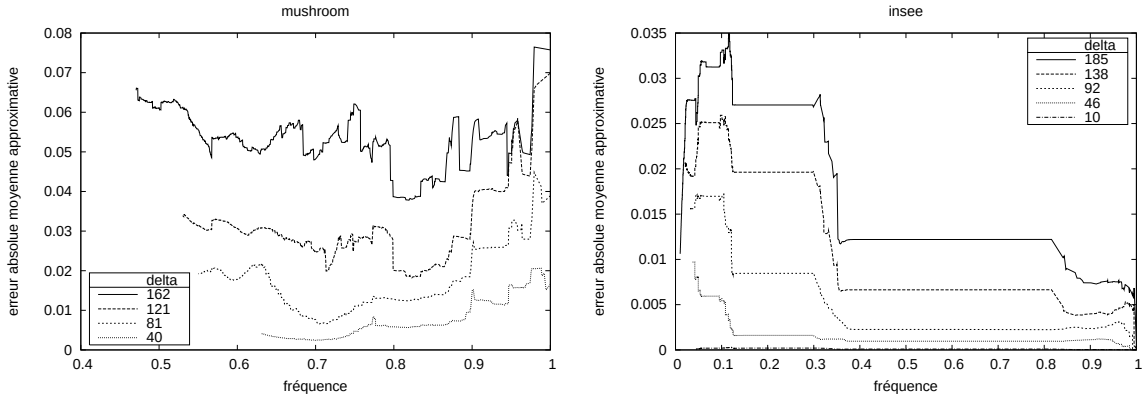


FIG. 3.7 – Erreur absolue moyenne approximative en fonction de δ et du seuil de fréquence.

3.5 Application à l'étude de données d'expression de gènes

Dans cette section, nous présentons une application de l'extraction des règles d'association à l'analyse de données d'expression de gènes que nous avons mené en collaboration avec des biologistes de l'équipe Signalisation et identités cellulaires dirigée par Olivier Gandrillon, Centre de Génétique Moléculaire et Cellulaire CNRS UMR 5534, université Lyon I. La récupération des données, leur préparation et les extractions ont été faites par une stagiaire en biologie, Celine Becquet, et un post-doc en informatique, Sylvain Blachon. L'interprétation biologique des résultats a été mené par toute l'équipe d'Olivier Gandrillon. Ces expériences ont donné lieu à une publication dans une revue internationnale de biologie [BBJ+02].

3.5.1 Un peu de biologie

Nous commençons par quelques rappels de biologie. Nous simplifions volontairement par souci de clarté.

Une des activité de toute cellule vivante est la production de protéines. Ces protéines sont des chaînes linéaires d'acides aminés pris parmi la vingtaine d'acides aminés existants. Elles sont produites par les ribosomes qui assemblent ces acides en décodant l'ordre de leur assemblage sur un ARN messager. Ces ARN messagers sont produits dans le noyau de la cellule à partir des gènes portés par l'ADN.

Le type des protéines produites et leur quantité varie au cours du temps en fonction de l'activité de la cellule. Les biologistes appellent *le niveau d'expression d'un gène* la mesure de l'intensité de la production de protéines codées par ce gène. Il faut noter qu'il n'existe pas à notre connaissance de consensus au sein des biologistes sur une définition plus précise du niveau d'expression d'un gène (il n'existe d'ailleurs pas de consensus sur la définition d'un gène).

Les biologistes ont montré que les niveaux d'expression des gènes influent les uns sur les autres. Par exemple, l'expression d'un gène A peut avoir pour effet, outre la production de protéines de type A, de diminuer ou au contraire d'augmenter le niveau d'expression d'un gène B. Celui-ci peut à son tour avoir une influence sur un gène C qui influe sur le gène A ... Ces interactions entre gènes peuvent être représentées par un graphe dans lequel les gènes sont les sommets et les arêtes décrivent les interactions entre les gènes. La construction de ce graphe et par conséquent la compréhension des interactions entre les gènes est l'un des nouveau objectifs de la recherche en biologie cellulaire. La connaissance de ces interactions permettra en effet de mieux comprendre le rôle de chacun des gènes[1] et par conséquent le fonctionnement de la cellule.

Les expérimentations que nous avons menées en collaboration avec les biologistes avaient pour but de montrer l'utilité des techniques d'extraction de connaissances que nous avons développées pour l'analyse de données d'expression de gènes.

3.5.2 Données brutes

Nous avons utilisé pour nos expériences des données dites d'expression de gènes. Ces données indiquent le niveau d'expression de certains gènes pour une cellule donnée dans une situation biologique donnée.

Il existe deux techniques principales pour mesurer ce niveau d'expression : les puces à ADN et la technique SAGE (Serial Analysis of Gene Expression). Dans ces deux techniques, on estime le niveau d'expression d'un gène en évaluant le nombre d'ARN messagers correspondant à ce gène présents dans la cellule à un instant donné. Dans le cas des puces à ADN, il faut choisir au préalable les gènes dont on veut mesurer l'expression, alors que pour la technique SAGE, on obtient le niveau d'expression de tous les gènes.

Dans notre cas, les données utilisées ont été produites en utilisant la technique SAGE. Celle-ci comporte plusieurs étapes : on commence par prélever des ARN messagers dans une cellule. Ensuite on séquence ceux-ci sur une dizaine de bases à partir d'une position particulière. On obtient alors, pour chaque ARN, une chaîne de caractères de longueur 10 sur l'alphabet A,C,T,G appelée "tag". Les tags identiques sont comptabilisés et on obtient

[1] Les grands programmes de séquençage du génome ont permis de décoder les sequences de nombreux gènes, mais la fonction de la majorité d'entre-eux reste inconnue.

finalement une liste de tags avec, pour chacun d'eux, le nombre de fois ou il a été séquencé. Cette technique fait donc un échantillonnage de l'ensemble des ARN contenus dans la cellule au moment du prélèvement.

Chaque tag correspond à un ou plusieurs gènes (il est en effet possible que deux ARN différents aient un même tag, on dira alors que le tag est ambigu) et des bases de données permettent de retrouver le ou les gènes correspondant à un tag et éventuellement leurs fonctions lorsqu'elles sont connues. La mesure du nombre de tags correspondant à un gène permet donc d'obtenir une approximation du niveau d'expression de celui-ci (la qualité de cette approximation dépend évidement du nombre total de tags séquencés).

Les données brutes utilisées proviennent d'un ensemble de 93 fichiers disponibles sur le web et concernent 93 expériences sur des cellules humaines de différents organes (cellules saines ou cancéreuses). Pour chacune des 93 expériences, nous disposions d'une liste de paires (t_i, nb_i) où t_i est un tag (chaîne de 10 caractères sur l'alphabet A,C,T,G) et nb_i un entier désignant le nombre de fois où ce tag a été séquencé. Le nombre total de tags séquencés dans une expérience (i.e., la somme sur i des nb_i) variait entre quelques milliers et environ 90000 (le nombre d'ARN dans une cellule à un instant donnée est de l'ordre de 300000).

3.5.3 Pré-traitements

Normalisation. Comme le nombre des tags séquencés variait énormément entre les expériences, il n'était pas possible de comparer directement les valeurs de nb_i entre ces expériences. Il fallait d'abord passer par une étape de normalisation. Celle-ci a été faite en divisant les valeurs de nb_i par la somme des nb_i (i.e., le nombre total de tags séquencés dans une expérience) pour chacune des expériences puis en multipliant cette valeur par 300000 (estimation du nombre d'ARN total par cellule). Les listes ainsi obtenues contenaient donc, pour chaque tag, une estimation du nombre de copies de l'ARN correspondant par cellule (cf fig. 3.8).

tag	avant	après normalisation
GATACCACTA	12	72
CTAGGTTAAC	2	12
TGCAAACTGA	35	210
...	...	...
t_i	nb_i	$nb_i * 6$
...	...	...

FIG. 3.8 – Exemple de liste avant et après normalisation (si le nombre total de tags $12 + 2 + 35 + ...$ vaut 50000).

Sélection des données et construction d'une matrice. Les biologistes ont alors choisi dans un premier temps de ne conserver que les 74 listes contenant au moins 20000 tags séquencés et, dans ces listes, de ne s'intéresser qu'à 822 des tags différents disponibles. Ces tags ont été choisi car ils étaient non ambigus (chacun de ces tags correspondait à un et un seul gène) et les gènes correspondant appartenaient au "transcriptome minimal" (ensemble minimal théorique des gènes exprimés dans tous les types de cellules). À partir de ces 74 listes de 822 tags différents, une matrice de 74 lignes et 822 colonnes a été construite. Cette matrice contient a l'intersection d'une ligne i et d'une colonne j le nombre (estimé) de copies de l'ARN correspondant au gène j dans l'expérience i (cf fig. 3.9). C'est cette valeur qui est utilisée comme "niveau d'expression" du gène j dans l'expérience i.

	gène A	gène B	gène C	....
exp. 1	23	45	554	...
exp. 2	12	13	333	...
exp. 3	32	0	18	...
...	...	...	...	...

FIG. 3.9 – Exemple de matrice d'expression où chaque ligne représente une expérience et chaque colonne un gène (on peut ici remplacer les tags par les gènes correspondants car les tags sélectionnés sont tous non ambigus).

Binarisation de la matrice. Les techniques d'extractions que nous utilisons nécessitent que la matrice soit booléenne, c'est-à-dire qu'elle ne contiennent que les valeurs 0 et 1. La matrice obtenue lors de l'étape précédente comportait des valeurs de niveau d'expression comprises entre 0 et environ 400, il a donc fallu binariser ces valeurs. Les biologistes souhaitaient étudier les groupes de gènes simultanément sur-exprimés[2], nous avons donc choisi de donner à un gène la valeur 1 dans la matrice pour une expérience donnée si ce gène y est sur-exprimé. Les biologistes ont alors essayé différents types de binarisation (c'est-à-dire plusieurs définitions de la sur-expression d'un gène).

Dans tous les cas, cette binarisation s'est faite en calculant des seuils pour chacun des gènes. Toutes les valeurs d'expression supérieures au seuil étaient transformées en 1 (ce qui signifiait que le gène est sur-exprimé) et les valeurs inférieures en 0 (le gène n'est pas sur-exprimé). Trois méthodes différentes ont été expérimentées par les biologistes pour déterminer le seuil pour un gène :

- Max moins $x\%$: Pour un gène G donné, soit $E_{max}(G)$ la plus forte valeur d'expression pour ce gène dans les 74 expériences. Le seuil est alors défini par $(1 - \frac{x}{100})E_{max}(G)$ où x est un pourcentage.
- Moyenne : Soit $E_{min}(G)$ la plus faible valeur d'expression d'un gène. Le seuil est défini comme la moyenne de E_{min} et E_{max} : $(E_{min}(G) + E_{max}(G))/2$.
- $(100 - x)$ centile : Le seuil est la valeur du $(100 - x)$ centile (i.e., les $x\%$ des valeurs les plus élevées du niveau d'expression sont transformées en 1 et les autres en 0).

Ces différentes binarisations ont permis aux biologistes d'étudier dans quelle mesure les motifs extraits dans les données dépendaient de la technique de binarisation.

	gène A	gène B	gène C	....
exp. 1	0	1	1	...
exp. 2	0	1	0	...
exp. 3	1	0	0	...
...	...	...	...	...

FIG. 3.10 – Exemple de binarisation sur la matrice de la fig. 3.9 avec des seuils de 30 pour le gène A, 8 pour le gène B et 400 pour le gène C.

3.5.4 Extraction dans le contexte booléen

Trois matrices booléennes furent construites en utilisant les trois techniques de détermination du seuil (Max moins 12%, Moyenne et 95^e centile). Les ensembles libres, leurs fréquences et leurs fermetures ont alors été extraits sur ces trois matrices pour des seuils de fréquences

[2]Un gène est sur-exprimé dans une cellule si son niveau d'expression y est supérieur à la normale

variant entre 2% et 10%. Une couverture des règles de confiance 1 a ensuite été générée à partir de ces itemsets libres et les biologistes ont alors étudié ces règles. Ces règles exprimaient des relations fortes entre les gènes. Par exemple, la règle $\texttt{AE} \Rightarrow \texttt{CD}, 24$ indique que lorsque les gènes `A` et `E` sont sur-exprimés dans une des 74 expériences, alors les gènes `C` et `D` sont également sur-exprimés dans ces mêmes expériences. De plus, cette règle a une fréquence de 24 ce qui signifie que ces gènes sont sur-exprimés dans 24 des 74 expériences.

3.5.5 Post-traitement et interprétation des résultats

Les règles ont été post-traités afin de permettre un examen visuel des résultats. Ce post-traitement consistait à afficher les gènes apparaissant dans les règles avec des couleurs qui dépendaient de la fonction des protéines codées par les gènes. Ce codage utilisait 7 couleurs. Six couleurs pour désigner les 6 grandes familles de fonctions des protéines et une couleur pour les gènes dont la fonction était inconnue. Cette représentation des règles en utilisant un codage par couleur avait l'avantage de permettre d'examiner d'un coup d'oeil un grand nombre de règles.

Les règles obtenues après coloriage exhibaient une grande homogénéité en terme de couleur, ce qui signifie que les gènes liés par une règle d'association partageaient une fonction similaire. Ce premier résultat semblait montrer que les motifs extrait avaient réellement un sens au niveau biologique. Ceci était plutôt encourageant et indiquait que la technique de l'extraction de motifs valait la peine d'être expérimentée par des biologistes pour l'étude des données d'expression de gènes.

Les biologistes ont ensuite sélectionné un ensemble de règles qu'ils ont étudié plus en détail. Tout d'abord, ils se sont concentrés sur les plus longues règles. Ils ont extrait 47 règles associant 13 gènes différents. Celles-ci étaient très homogènes et associaient principalement des gènes impliqués dans la synthèse de protéines. Seuls deux gènes sur les treize n'étaient pas dans cette catégorie. Le premier appartenait à la catégorie des gènes impliqués dans la transcription et le second était de fonction inconnue. En recherchant les tags associés dans une base de données, ils se sont rendu compte que le premier tag avait été incorrectement associé à un gène. Il correspondait en fait à un gène participant à la synthèse de protéines. Quand au second, après vérification, il était toujours de fonction inconnue. Les biologistes ont cependant estimé qu'il était raisonnable de conjecturer que le gène inconnu était également impliqué dans la synthèse de protéines.

Les biologistes ont aussi étudié un certain nombre d'autres règles qui ont permis de confirmer des relations déjà connues ou seulement supposées entre des gènes. Il ont aussi découvert des relations qui n'avait pas à leur connaissance été observées auparavant mais qui leur semblaient biologiquement plausibles et en tout cas intéressantes.

Enfin, les biologistes ont comparé les règles extraites avec une autre technique de fouille de données couramment utilisée en biologie : le clustering hiérarchique. Dans cette technique, les gènes sont classées en différents groupes en fonction des expériences dans lesquels ils sont sur-exprimés. Par rapport aux règles d'association, la sortie d'un algorithme de clustering hiérarchique est beaucoup plus petite (un arbre dont les feuilles sont les gènes contre un ensemble pouvant être très grand de règles d'association), par contre, un gène ne peut apparaître que dans un seul groupe alors que dans le cas des règles d'association, un gène peut apparaître dans de très nombreuses règles. Cette caractéristique des règles d'association a beaucoup intéressé les biologistes. En effet, les réseaux de régulation des gènes (qui décrivent leurs interactions) sont probablement très complexes et ne peuvent pas être correctement décrits en faisant simplement un partitionnement des gènes. Les règles d'association permettent

sans doute d'avoir une description plus fine, car chaque gène peut apparaître dans un nombre quelconque de règles. Ainsi, un gène qui aurait un effet régulateur sur plusieurs autres gènes de fonctions différentes peut apparaître dans plusieurs règles. En revanche, dans le cas du clustering, il n'apparaîtrait que dans un seul groupe, celui sur lequel il a le plus d'influence.

Lors de leurs expériences, les biologistes ont cependant noté que les gènes qui apparaissaient dans un même groupe étaient aussi généralement liés par plusieurs règles d'association. Ces deux techniques ne donnent donc pas des résultats totalement déconnectés. Cependant, il n'y avait pas d'homogénéité (en terme de fonction des gènes) entre les gènes d'un même cluster comme cela avait été observé pour les règles. Un groupe regroupait fréquemment des gènes ayant des fonctions très différentes, alors que les règles concernaient souvent des gènes ayant des fonctions proches. Le clustering ne permet donc probablement pas de faire des hypothèses sur la fonction d'un gène inconnu comme les biologistes avaient pu le faire en utilisant les règles d'association (voir plus haut).

Finalement, la technique d'extraction de règles d'association dans des matrices d'expression de gènes semble très prometteuse. Les règles extraites sont en effet très homogènes en ce qui concerne la fonction des gènes y apparaissant, ce qui est biologiquement satisfaisant, et de plus, les règles font apparaître des relations entre des gènes qui avaient déjà été observées ou qui semblent intéressante aux yeux des biologistes.

3.5.6 Perspectives

Plusieurs améliorations sont en cours d'étude. Tout d'abord, les extractions n'ont été faites que sur un ensemble très restreint de gènes (environ 800). De nouvelles extractions sont en cours en utilisant un plus grand nombre de gènes, si possible en utilisant tous les tags non ambigus (i.e., qui correspondent à un et un seul gène), c'est-à-dire plusieurs milliers de tags. Ceci ne va pas sans poser de gros problèmes de complexité : on augmente ainsi considérablement le nombre de colonnes de la matrice et par conséquent la taille de l'espace de recherche. Si l'utilisation des itemsets libres avait permis des extractions dans le cas restreint à 822 gènes, les premières expériences montrent que cela ne sera plus suffisant dans le cas ou la matrice contient plusieurs milliers de gènes. Dans ce cas, deux possibilités sont envisageables : l'utilisation des contraintes et l'extraction sur la matrice transposée. Nous avons déjà vu que l'utilisation de contraintes spécifiées par l'utilisateur peut considérablement améliorer les performances de l'extraction. Ici, les biologistes seront sans doute intéressés par des contraintes portant sur la fonction des gènes (pour étudier plus en détails certaines familles de gènes ou certains gènes particuliers) ou l'homogénéité des règles (rechercher des règles très homogènes ou au contraire impliquant des gènes de fonctions très différentes). La deuxième possibilité est l'utilisation de la matrice transposée. La matrice sur laquelle nous avons fait les extractions est très dissymétrique, elle avait peu de lignes (moins d'une centaine) et beaucoup de colonnes (de 800 à plusieurs milliers). Cette situation n'est pas très favorable pour l'utilisation de nos techniques. En effet, la taille de l'espace de recherche est exponentielle par rapport au nombre de colonnes alors que la complexité par rapport au nombre de lignes est linéaire. Dans ce cas, faire l'extraction sur la matrice transposée est très intéressant, on se retrouve en effet avec une matrice qui ne contient plus qu'une centaine de colonnes. Les règles extraites concernent alors des relations entre les expériences, par exemple : lorsqu'un gène est sur-exprimé dans les expériences 3 et 4 alors il est aussi sur-exprimé dans l'expérience 6. Mais il est également possible, en utilisant les treillis de concepts, de retrouver l'information sur les gènes à partir de ces extractions sur la matrice transposée. Des travaux très récents [RC03] ont montré tout l'intéret de cette approche.

Au delà de l'extraction proprement dite, deux types d'améliorations sont envisageables au niveau du post-traitement. Tout d'abord, le codage des fonctions des gènes en utilisant les couleurs est pour l'instant très grossier. Il n'utilise en effet que 7 couleurs. L'utilisation de classifications beaucoup plus précises des gènes est en cours d'étude. Enfin, de nombreuses mesures de la qualité des règles autres que la confiance et la fréquence pourraient être utilisées.

3.6 Conclusion

Dans ce chapitre, nous avons présenté un algorithme CoCo combinant les représentations condensées utilisant les δ-libres contextuels avec l'utilisation active des contraintes monotones et anti-monotones pour l'évaluation de requêtes inductives. Ceci nous a permis de proposer des nouvelles stratégies d'évaluation de requêtes inductives. Nos expérimentations montrent tout l'intérêt de celles-ci lorsque les données sont fortement corrélées.

Chapitre 4

Optimisation d'une séquence de requêtes

Les travaux exposés dans ce chapitre ont été présentés en partie dans [JB02c].

4.1 Introduction

Le processus d'extraction de connaissances dans des données (ECD) est, comme nous l'avons déjà vu dans le chapitre 1, un processus interactif et itératif. Cela signifie que l'utilisateur va demander l'évaluation non pas d'une seule requête mais d'une séquence de requêtes. Dans les deux chapitres précédents, nous avons présenté des stratégies d'évaluation pour optimiser *une* requête d'extraction d'itemsets. Ici, nous nous posons le problème de l'optimisation de l'évaluation d'une *séquence* de requêtes.

Pour l'optimisation de cette séquence, nous nous basons sur l'hypothèse que les requêtes de l'utilisateur vont partager un certain degré de similarité. En effet, lorsque l'utilisateur définit une nouvelle requête, il procédera naturellement en modifiant sa dernière requête afin d'affiner sa recherche ou, au contraire, de l'élargir. On peut donc supposer qu'une partie des résultats de sa nouvelle requête auront déjà été calculés lors de l'évaluation des requêtes précédentes. Étant donné le coût élevé des calculs nécessaires pour évaluer une requête, il est clair que le système doit éviter de refaire des calculs.

4.2 Comparaison de deux techniques d'évaluation d'une séquence de requêtes

Considérons une séquence de requêtes d'extraction d'itemsets $\sigma_i = (bd, 2^{\mathsf{Items}}, \mathcal{C}_i, \mathsf{freq})$ pour $1 \leq i \leq k$. Les techniques que nous avons vues dans les chapitres précédents nous permettent d'optimiser l'évaluation de chacune de ces requêtes en utilisant activement la contrainte $\mathcal{C}_i$ et les représentations condensées.

On peut également utiliser ici une stratégie de type "générer et tester". C'est-à-dire que l'on commence par calculer une collection d'itemsets fréquents $\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq})$ et on filtre ensuite cette collection en utilisant la contrainte $\mathcal{C}_i$ pour calculer la réponse de la requête σ_i.

Nous avons déjà vu que si on considère une seule requête, alors la stratégie générer et tester est beaucoup plus coûteuse que de pousser la contrainte. Par contre, si il faut évaluer

une séquence de requêtes, ce n'est plus le cas. Évidemment, on ne calculera la collection $\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq})$ qu'une seule fois et on l'utilisera pour l'évaluation de toutes les requêtes σ_i. Si il faut un temps t_{coll} pour calculer la collection $\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq})$ et un temps moyen de t_{sel} pour sélectionner dans cette collection les itemsets satisfaisant une contrainte $\mathcal{C}_i$, alors le temps total pour l'évaluation d'une séquence de k requêtes par la stratégie générer et tester est de $t_{coll} + k.t_{sel}$. Si le temps moyen d'évaluation d'une de ces requêtes en utilisant les optimisations que nous avons vues dans les deux chapitres précédents est de t_{opt}, alors le coût de la séquence complète est de $k.t_{opt}$.

Lorsqu'il n'y a qu'une requête, il est clair que t_{coll} est très supérieur à t_{opt}. Par conséquent la stratégie générer et tester est inefficace. Par contre, le coût t_{sel} d'une sélection dans la collection $\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq})$ est très inférieur à celui du calcul de la solution même en utilisant des optimisations : $t_{sel} << t_{opt}$. Or, le rapport du temps de calcul des deux stratégies est asymptotiquement égal à t_{sel}/t_{opt}. Donc, pour une séquence suffisamment longue, la stratégie générer et tester est plus efficace. De plus, l'utilisateur n'a pas besoin d'être présent lors du calcul de la collection $\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq})$. On peut imaginer que ce calcul se fasse automatiquement chaque nuit. Dans ce cas, le rapport de force est encore plus en faveur d'une stratégie générer et tester.

Il semble donc que l'optimisation d'une requête en utilisant les contraintes ou les représentations condensées ne soit finalement pas très utile. Ce type de critique sur l'utilisation des contraintes est fait par exemple dans [GdB99].

La technique générer et tester souffre pourtant de plusieurs inconvénients. Tout d'abord, il est évident qu'il faut que le résultat de la requête soit inclus dans la collection $\mathsf{Res}(bd, \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq})$. Or, il est tout à fait possible que l'utilisateur soit intéressé par des motifs peu fréquents. Dans ce cas, il faudra recalculer une nouvelle collection $\mathsf{Res}(bd, \mathcal{C}_{\mu\text{-}\mathsf{freq}}, \mathsf{freq})$ avec un seuil de fréquence μ plus faible. Toutefois, il est possible que cette nouvelle collection ne soit pas calculable en un temps raisonnable alors même que l'utilisateur n'est intéressé que par un petit nombre des itemsets peu fréquents (c'est-à-dire qu'une extraction sous contraintes serait faisable). La taille de cette collection peut aussi devenir trop importante lorsque le seuil de fréquence est bas. Ici encore, il est possible que l'utilisateur n'ait jamais besoin de toute cette collection.

Enfin, si la base de données binaire *bd* change entre les requêtes, alors il faut recalculer la collection $\mathsf{Res}(bd', \mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq})$ sur la base de données modifiée bd'. Si la base de données binaire *bd* est définie comme une vue sur une base de données relationnelle, on peut distinguer deux types de modifications :

- la base relationnelle est modifiée par ajout, suppression ou modification de tuples ;
- la vue définissant la base de données binaire est modifiée, par exemple car l'utilisateur s'est rendu compte que son choix de binarisation pour un attribut n'est pas bon.

Dans le premier cas, il existe des algorithmes pour mettre à jour de manière incrémentale les collections de motifs calculés. En revanche, dans le second cas, il faut recalculer entièrement la collection. Or ces changements peuvent être fréquents, en particulier si l'utilisateur est en train de régler différents paramètres (comme pour la binarisation) qui influent sur la base de données binaire.

Finalement, une stratégie générer et tester n'est donc pas suffisamment souple pour pouvoir être utilisée en pratique. L'idéal serait d'avoir une stratégie qui allie la souplesse de l'extraction sous contraintes avec l'efficacité de l'utilisation d'une collection de motifs déjà matérialisée. Pour cela, il faut concevoir un algorithme d'extraction utilisant les techniques d'optimisation que nous avons vues dans les précédents chapitres et pouvant aussi utiliser les résultats des requêtes précédentes afin de ne pas recalculer des motifs déjà matérialisés.

4.3 Travaux existants

Plusieurs travaux ont déjà été publiés sur le problème du traitement de plusieurs requêtes. Les travaux que nous allons rapidement présenter ici traitent en général soit des règles d'association soit des séquences.

Dans [BP99], E. Baralis et G. Psaila étudient le cas des règles d'association. Ils considèrent des requêtes écrites en utilisant le langage MINE RULE [MPC96] que nous avons présenté dans la section 1.1.5 page 25. Le résultat d'une requête MINE RULE et une collection de règles et, pour chaque règle, une valeur de fréquence et de confiance. Étant donné deux requêtes portant sur les règles d'association, les auteurs identifient trois types de relations entre ces requêtes.

- L'équivalence : deux requêtes sont équivalentes si les résultats de ces deux requêtes sont égaux quel que soit la base de données.
- L'inclusion : une requête Q_1 est incluse dans Q_2 si toute règle présente dans le résultat de Q_1 est aussi présente dans le résultat de Q_2 avec les mêmes valeurs de fréquence et de confiance et ceci quel que soit la base de données.
- La dominance : une requête Q_1 est dominée par Q_2 si toute règle r présente dans le résultat de Q_1 avec une fréquence f_1 et une confiance c_1 est aussi présente dans le résultat de Q_2 avec une fréquence $f_2 \geq f_1$ et une confiance $c_2 \geq c_1$.

Les auteurs présentent ensuite un algorithme pour mettre à jour une collection de règles lorsqu'une de ces propriétés est détectée sur deux requêtes. Cet algorithme est très simple. En effet, dans le cas d'équivalence, il n'y a rien à faire ; dans le cas d'inclusion, il suffit de filtrer le résultat et dans le cas de dominance, il faut refaire les calculs de fréquence et de confiance en une passe sur les données et ensuite filtrer l'ensemble des règles. L'apport principal de cet article est d'étudier le langage MINE RULE afin de détecter ces relations entre les requêtes MINE RULE. Pour cela, les auteurs considèrent deux requêtes différentes (seuils de fréquence différents, cardinaux maximal et minimal de la tête et du corps de la règle différents, contraintes de sélection différentes ...) et étudient les relations liant ces deux requêtes.

Des travaux similaires ont également été faits dans le domaine de l'extraction de séquences [Woj01]. L'auteur définit également des relations d'équivalence, d'inclusion et de domination entre des paires de requêtes et étudie dans quels cas ces relations sont vraies sur des paires de requêtes d'extraction de séquences.

Dans [MWZ00], T. Morzy, M. Wojciechowski et M. Zakrzewicz proposent d'utiliser des collections de motifs comme des vues dans les bases de données relationnelles. Leur langage de requête est une extension de SQL pour traiter les itemsets dans lequel une requête est constituée de trois éléments : un seuil de fréquence, une partie définissant les données dans lesquelles les itemsets sont extraits et une partie définissant les contraintes syntaxiques que doivent satisfaire les itemsets. Ils introduisent alors des vues composées d'itemsets et définies par une requête et ils étudient comment utiliser ces vues pour optimiser l'extraction d'une nouvelle requête. Ils ne proposent pas de nouvel algorithme mais montrent plutôt quel algorithme existant utiliser selon la relation entre la vue et la nouvelle requête. Selon cette relation, ils identifient 4 types d'algorithmes utilisables :

- extraction complète si aucun autre algorithme n'est applicable ;
- extraction incrémentale si la base de données de la nouvelle requête contient des tuples supplémentaires par rapport à celle utilisée dans la vue (cela correspond au cas où les contraintes de sélection des données sont affaiblies) ;
- extraction complémentaire si les contraintes sur les itemsets sont plus faibles que dans

la requête définissant la vue (ils proposent alors d'utiliser l'algorithme de [NDD99] mais l'algorithme que nous allons présenter serait aussi utilisable dans ce cas) ;
- filtrage si les contraintes sur les itemsets sont plus restrictives dans la nouvelle requête que dans la requête définissant la vue.

S. Parthasarathy et al. montrent dans [PZOD99] comment mettre à jour le résultat d'une requête d'extraction de séquences sans avoir à refaire tous les calculs lorsque l'utilisateur modifie sa requête ou lorsque la base de données est mise à jour par l'insertion de nouveaux tuples. Dans cet article, une seule collection de séquences est maintenue et mise à jour. Par conséquent, seul le résultat de la requête immédiatement précédente est utilisable. Les résultats des requêtes plus anciennes ne sont pas conservés.

Enfin, différents travaux portent sur l'utilisation d'un cache dans lequel sont stockés les résultats des requêtes précédentes. Dans [NDD99], B. Nag et al. montrent comment utiliser un tel cache. Leur cache contient les itemsets et leurs fréquences. Ce cache est construit par leur algorithme (une extension de CAP) lorsqu'il calcule la solution de requêtes. Les auteurs décrivent plusieurs techniques de gestion du cache pour éviter que celui-ci ne grossisse trop.
- Avec la stratégie *no replacement* aucune gestion du cache n'est faite. Tous les itemsets sont ajoutés dans le cache jusqu'à ce que celui-ci atteigne sa taille limite. Ensuite, plus aucun itemset n'est ajouté.
- Avec la stratégie *simple replacement*, lorsque le cache est plein, les itemsets dont la fréquence est la plus faible sont éliminés et remplacés par des itemsets de fréquence plus grande. Les auteurs considèrent en effet que garder des itemsets avec une grande fréquence est plus intéressant car ils sont plus souvent utilisés et le calcul de leur fréquence est plus coûteux.
- Enfin, la stratégie *benefit replacement* est plus efficace, mais elle est limitée au cas très restrictif où les requêtes ne contiennent qu'une contrainte de fréquence. Cela signifie que la seule différence entre les requêtes est la valeur du seuil de fréquence.

Dans l'algorithme proposé par [GdB00], des collections d'itemsets et de règles d'association sont également stockées pour être réutilisées ensuite dans les évaluations des requêtes suivantes.

L'approche proposée dans [DGLS01] par C. T. Diop et al. n'utilise pas de cache à proprement parler. Les auteurs proposent un modèle permettant à l'utilisateur de combiner et de manipuler des tâches d'extractions déjà calculées en utilisant des opérations analogues à celles de l'algèbre relationnelle. Ils montrent également comment utiliser les résultats de ces tâches pour optimiser le calcul des réponses aux nouvelles tâches.

4.3.1 Discussion

Une partie de ces travaux se consacrent à la découverte de relations simples, comme l'inclusion, entre les résultats des requêtes. Par conséquent, ils sont limités au cas où une telle relation existe effectivement entre les requêtes.

Nos travaux se rapprochent plus de ceux de Nag et al. et de Goethals et al. dans lesquels les résultats des requêtes précédentes sont utilisés indépendamment des relations entre les requêtes. Par contre, dans ces travaux, les collections d'itemsets complètes sont stockées. Or, dans le cas de données corrélées, ces collections contiennent beaucoup de redondance et sont de plus très grandes et difficiles à extraire. Nous proposons donc d'utiliser des représentations condensées comme un cache pour accélérer les extractions et limiter la taille du cache. Indépendamment de nos travaux, A. Giacometti et al. [GLD02] ont également fait une proposition similaire bien qu'ils se focalisent plus sur la représentation condensée d'un ensemble de

requêtes et moins sur l'optimisation de leur évaluation. Nous comparerons nos deux approche plus loin.

4.4 Description de notre approche

Nous avons vu dans le chapitre précédent que l'évaluation d'une requête sur les itemsets ou les règles d'association peut se décomposer en plusieurs étapes :

- calcul d'une représentation condensée des itemsets satisfaisant la contrainte ;
- régénération de tous les itemsets satisfaisant la contrainte ;
- génération éventuelle des règles d'association à partir des itemsets.

Il y a des variantes, comme par exemple la génération directe des règles à partir des représentations condensées. En tous cas, l'étape la plus coûteuse reste la première. De plus, si on considère la taille des collections de motifs, celle-ci est croissante de l'étape 1 à l'étape 2 et de l'étape 2 à l'étape 3. Ainsi, si on veut stocker des résultats des requêtes, il semble judicieux de stocker les résultats obtenus à la fin de la première étape, c'est-à-dire les représentations condensées des itemsets. Ainsi, toute l'information qui a été calculée est conservée et la régénération des collections des itemsets et des règles est très rapide. Par rapport aux travaux précédents, nous utilisons une représentation condensée ce qui permet de diminuer la taille des collections stockées. Nous pouvons donc conserver plus d'information sur des données plus denses pour la même taille de stockage.

De plus, comme nous l'avons vu dans le chapitre précédent, le fait que nous utilisions les représentations condensées dans la première étape de l'extraction améliore les performances dans les données corrélées et permet d'extraire à des seuils de fréquence impossibles à atteindre autrement.

Dans certains des travaux que nous avons vus dans la section précédente, les résultats de plusieurs requêtes sont stockés de manière indépendante. Cette technique n'est pas très économique. En effet, les informations concernant les itemsets qui appartiennent à plusieurs collections sont dupliquées. C'est pour cette raison que nous avons choisi de stocker les informations relatives à toutes les requêtes dans une seule collection.

Cette collection est utilisée par notre algorithme comme un cache. À chaque fois que l'algorithme a besoin de la fréquence et de la fermeture d'un itemset, il commence par chercher cette information dans le cache, puis, si elle ne s'y trouve pas, cette information est calculée à partir de la base de données binaire. Nous ne faisons donc pas d'hypothèse sur les différentes requêtes et en particulier nous ne requérons pas qu'il y ait une relation d'inclusion (ou tout autre type de relation) entre les requêtes.

Dans la suite, nous considérons donc une suite de requêtes du type $\sigma_i = (bd, 2^{\mathsf{Items}}, \mathcal{C}_{\mathsf{am}_i} \wedge \mathcal{C}_{\mathsf{m}_i}, \mathsf{freq})$, $1 \leq i \leq n$ où les contraintes $\mathcal{C}_{\mathsf{am}_i}$ sont anti-monotones et les $\mathcal{C}_{\mathsf{m}_i}$ monotones.

4.5 Algorithme **iCoCo**

Comme nous voulons utiliser un algorithme d'extraction de représentations condensées sous contraintes, il faut que le cache permette d'inférer la fréquence et la fermeture des itemsets. De plus, il doit contenir des collections d'itemsets issues de différentes requêtes. Le plus simple est alors d'utiliser une représentation condensée du type (3.6) qui contient des libres contextuels, leur fréquence et leur fermeture. Ainsi, notre cache est une collection de triplets de la forme $(S, \mathsf{freq}(S), \mathsf{ferm}(S) \setminus S)$ où S est un itemset.

Pour utiliser ce cache, nous présentons ici une variante iCoCo (i comme *interactif*) de l'algorithme CoCo d'extraction de représentations condensées sous contraintes présenté dans le chapitre 3. L'algorithme iCoCo utilise les mêmes fonctions de génération des candidats, d'élagage et de passe sur les données que l'algorithme CoCo. Les deux différences principales entre les algorithmes CoCo et iCoCo sont d'une part l'utilisation du cache et d'autre part le fait que l'algorithme iCoCo ne considère que des représentations condensées exactes (il n'utilise pas les δ-libres avec $\delta \neq 0$). Par conséquent, lors de la passe sur les données, l'algorithme iCoCo calcule des fermetures au lieu des δ-fermetures pour CoCo. Il utilise donc la même fonction de passe sur les données que l'algorithme Close. Dans l'algorithme ci-dessous, les lignes qui ont été modifiées ou ajoutées par rapport à l'algorithme CoCo sont marquées d'une astérisque. Nous rappelons que $\mathcal{C}_{\mathsf{am}} = \mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_{\mathsf{am}_{ns}}$ est une contrainte anti-monotone conjonction d'une contrainte syntaxique et d'une contrainte non syntaxique, $\mathcal{C}_{\mathsf{m}}$ est une contrainte monotone et $\mathcal{E} = \mathsf{Th}(\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_{\mathsf{m}})$.

Algorithme 7 (Algorithme iCoCo)

Entrée : Une contrainte de la forme $\mathcal{C}_{\mathsf{am}_s} \wedge \mathcal{C}_{\mathsf{am}_{ns}} \wedge \mathcal{C}_{\mathsf{m}}$ et un cache C
Sortie : $\mathsf{Res}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\text{libre-c}}, (\mathsf{freq}, \mathsf{ferm}))$, $\mathcal{B}d^{-}_{\mathcal{E}}(\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\text{libre-c}})) \cap \mathsf{Th}(\mathcal{C}_{\text{libre-c}})$ et un cache C mis à jour

1 $k := n\_min$
2 $\mathcal{F}_1 := \{i \in \mathsf{Items} \mid \mathcal{C}_{\mathsf{am}}(i)\}$
3 $\mathsf{Cand} := \mathbf{gen\_cand\_coco}(k, \emptyset)$
4 $\mathsf{Cand} := \{S \in \mathsf{Cand} \mid \mathcal{C}_{\mathsf{am}_s}(S) = vrai\}$ /* test de $\mathcal{C}_{\mathsf{am}_s}$ */
5 **tant que** $\mathsf{Cand} \neq \emptyset$ **ou** $k \leq n\_max$ **faire**
6* $(\mathsf{Cand}_{dc}, \mathsf{Cand}_{hc}) := \mathbf{passe\_cache}(\mathsf{Cand}, C)$ /* passe sur le cache */
7* $\mathsf{Cand}_{hc} := \mathbf{passe\_bd\_close}(\mathsf{Cand}_{hc}, bd)$ /* passe sur les données */
8* $C := \mathbf{insertion}(\mathsf{Cand}_{hc}, C)$ /* mise à jour du cache */
9* $\mathsf{Cand} = \mathsf{Cand}_{dc} \cup \mathsf{Cand}_{hc}$
10 $\mathcal{B}_k := \{S \in \mathsf{Cand} \mid \mathcal{C}_{\mathsf{am}_{ns}}(S) = faux\}$ /* test de $\mathcal{C}_{\mathsf{am}_{ns}}$ */
11 $\mathcal{S}_k := \mathsf{Cand} \setminus \mathcal{B}_k$
12 $\mathsf{Cand} := \mathbf{gen\_cand\_coco}(k+1, \mathcal{S}_k)$ /* génération des candidats */
13 $\mathsf{Cand} := \{S \in \mathsf{Cand} \mid \mathcal{C}_{\mathsf{am}_s}(S) = vrai\}$ /* test de $\mathcal{C}_{\mathsf{am}_s}$ */
14 $\mathsf{Cand} := \mathbf{élagage\_coco}(\mathsf{Cand}, \mathcal{S}_k)$ /* élagage et test de $\mathcal{C}_{\delta-\text{libre-c}}$ */
15 $k := k + 1$
16 **fait**
17 **retourne** $\bigcup_{0 \leq i \leq k} \mathcal{S}_i$ et $\bigcup_{0 \leq i \leq k} \mathcal{B}_i$

Le cache est utilisé juste avant la passe sur les données. Pour chaque itemset candidat S, on recherche dans le cache un triplet $(T, \mathsf{freq}(T), \mathsf{ferm}(T) \setminus T)$ tel que $T \subseteq S \subseteq \mathsf{ferm}(T)$. Si un tel itemset T existe dans le cache, alors, d'après les propriétés de la fermeture, on sait que S et T sont dans la même classe d'équivalence et par conséquent $\mathsf{freq}(S) = \mathsf{freq}(T)$ et $\mathsf{ferm}(S) = \mathsf{ferm}(T)$. Un triplet $(T, \mathsf{freq}(T), \mathsf{ferm}(T) \setminus T)$ contenu dans le cache permet donc de calculer la fréquence et la fermeture de tous les itemsets contenant T et inclus dans $\mathsf{ferm}(T)$. Les itemsets dont la fréquence et la fermeture ont pu être inférées à partir du cache sont mis dans Cand_{dc} (dans cache) les autres dans Cand_{hc} (hors cache).

passe_cache(Cand, C)
1 $\mathsf{Cand}_{dc} := \emptyset$ $\mathsf{Cand}_{hc} := \emptyset$
2 **pour tout** $S \in \mathsf{Cand}$ **faire**

3 **si** il existe $(T, f, F) \in C$ tel que $T \subseteq S.\mathsf{item} \subseteq T \cup F$ **alors**
4 $S.\mathsf{freq} := f$
5 $S.\mathsf{ferm} := F$
6 $\mathsf{Cand}_{dc} := \mathsf{Cand}_{dc} \cup \{S\}$
7 **sinon**
8 $\mathsf{Cand}_{hc} := \mathsf{Cand}_{hc} \cup \{S\}$
9 **finsi**
10 **fait**
11 **retourne** $(\mathsf{Cand}_{dc}, \mathsf{Cand}_{hc})$

Ensuite, la fréquence et la fermeture des candidats de Cand_{hc} pour lesquels elles sont toujours inconnues sont calculées de manière classique en faisant une passe sur les données. Ces candidats, dont le cache ne permettait pas de calculer la fréquence et la fermeture, sont alors insérés dans le cache avec leur fréquence et leur fermeture.

Si le cache était initialement vide alors, à la fin de l'évaluation de la première requête σ_1, il contient tous les itemsets libres contextuels dont l'algorithme a calculé la fermeture et la fréquence. C'est-à-dire que le cache contient tous les triplets $(S, \mathsf{freq}(S), \mathsf{ferm}(S) \setminus S)$ pour chaque itemset S appartenant à

$$\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\mathsf{libre\text{-}c}}) \cup \left[\mathcal{B}d^{-}_{\mathcal{E}}(\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\mathsf{libre\text{-}c}})) \cap \mathsf{Th}(\mathcal{C}_{\mathsf{libre\text{-}c}})\right].$$

Lorsqu'une nouvelle requête σ_2 est évaluée par l'algorithme, l'insertion des nouveaux itemsets dans le cache pourrait se faire très simplement en y ajoutant les triplets $(S, \mathsf{freq}(S), \mathsf{ferm}(S) \setminus S)$ pour chacun des itemsets S. Cependant, en procédant ainsi, on risque d'introduire une redondance inutile dans le cache comme le montre l'exemple suivant.

Exemple 37 *Dans cet exemple, nous étudions les itemsets de la classe d'équivalence de* ABCD *dans la base de données* bd_1 *(voir la figure 3.1), c'est-à-dire les itemsets*

$$\{\mathtt{BC}, \mathtt{BD}, \mathtt{ABC}, \mathtt{ABD}, \mathtt{BCD}, \mathtt{ABCD}\}.$$

Considérons d'abord la requête $\sigma_1 = (bd_1, \mathcal{C}_{\text{2-freq}} \wedge \mathcal{C}_{\mathsf{m}}, \mathsf{freq})$ *où* $\mathcal{C}_{\mathsf{m}}(S) = (\mathtt{A} \in S)$*. Dans la classe d'équivalence de* ABCD*, les itemsets libres contextuels qui satisfont cette contrainte sont* ABC *et* ABD*. Par conséquent, le cache contient, à la fin de l'évaluation de la requête* σ_1*, les triplets* $(\mathtt{ABC}, \mathtt{D}, 2)$ *et* $(\mathtt{ABD}, \mathtt{C}, 2)$*. Le premier de ces triplets permet d'inférer la fermeture et la fréquence des itemsets* ABC *et* ABCD *et le second la fermeture et la fréquence de* ABD *et* ABCD*.*

Soit maintenant la requête $\sigma_2 = (bd_1, \mathcal{C}_{\text{2-freq}} \wedge \mathcal{C}'_{\mathsf{m}}, \mathsf{freq})$ *où* $\mathcal{C}_{\mathsf{m}}(S) = (\mathtt{B} \in S)$*, les itemsets libres contextuels relativement à* $\mathcal{C}'_{\mathsf{m}}$ *qui satisfont* $\mathcal{C}_{\text{2-freq}} \wedge \mathcal{C}'_{\mathsf{m}}$ *sont* BD *et* BC*. L'algorithme insère donc dans le cache les triplets* $(\mathtt{BD}, \mathtt{AC}, 2)$ *et* $(\mathtt{BC}, \mathtt{AD}, 2)$*. Ceux-ci permettent d'inférer la fermeture et la fréquence de tous les itemsets de la classe d'équivalence de* ABCD*. Par conséquent, il y a redondance avec les triplets* $(\mathtt{ABC}, \mathtt{D}, 2)$ *et* $(\mathtt{ABD}, \mathtt{C}, 2)$*. Ces derniers peuvent en fait être supprimés du cache sans perte d'information.*

Le cache contient des triplets $(S, \mathsf{freq}(S), \mathsf{ferm}(S) \setminus S)$ où l'itemset S est un libre contextuel relativement à l'une des contraintes $\mathcal{C}_{\mathsf{m}_i}$ utilisée dans l'une des requêtes. Ce triplet contient l'information sur la fréquence et la fermeture des itemsets inclus dans $\mathsf{ferm}(S)$ et contenant S. Lorsque de nouveaux triplets sont insérés dans le cache, il est possible que l'un d'entre eux généralise le triplet $(S, \mathsf{freq}(S), \mathsf{ferm}(S) \setminus S)$, i.e., qu'il contienne plus d'information. Dans ce cas, le triplet $(S, \mathsf{freq}(S), \mathsf{ferm}(S) \setminus S)$ peut être retiré du cache et remplacé par le nouveau.

Proposition 29 *Soit* $t = (S, \mathsf{freq}(S), \mathsf{ferm}(S) \setminus S)$ *un triplet dans le cache et* $t' = (T, \mathsf{freq}(T), \mathsf{ferm}(T) \setminus T)$ *un triplet à insérer dans le cache, alors le triplet* t *peut être effacé du cache et remplacé par le triplet* t' *sans perte d'information (par rapport à notre algorithme) si et seulement si* $T \subseteq S$ *et* $\mathsf{ferm}(T) = \mathsf{ferm}(S)$.

Preuve : si $T \subseteq S$ et $\mathsf{ferm}(T) = \mathsf{ferm}(S)$ alors tout itemset U tel que $S \subseteq U \subseteq \mathsf{ferm}(S)$ vérifie également $T \subseteq U \subseteq \mathsf{ferm}(T)$. Par conséquent, si la fréquence et la fermeture d'un itemset peuvent être inférées à partir de t, elles peuvent aussi l'être à partir de t'. Donc, en remplaçant t par t' dans le cache, il n'y a aucune perte d'information. Réciproquement, si la fréquence et la fermeture d'un itemset qui peuvent être inférées à partir de t peuvent aussi l'être à partir de t' en utilisant l'algorithme, alors cela signifie que $T \subseteq S$ et $\mathsf{ferm}(S) \subseteq \mathsf{ferm}(T)$. Comme la fermeture est croissante et que $T \subseteq S$, on a aussi $\mathsf{ferm}(T) \subseteq \mathsf{ferm}(S)$ et par conséquent $\mathsf{ferm}(S) = \mathsf{ferm}(T)$. □

Nous utilisons cette proposition dans la fonction d'insertion des candidats :

```
insertion(Cand, C)
1   pour tout S ∈ Cand faire
2      pour tout (T, f, F) ∈ C faire
3         si S.item ⊂ T et S.ferm = F alors
4            C := C \ {(T, f, F)}
5         finsi
6         C := C ∪ {(S.item, S.freq, S.ferm)}
7      fait
8   fait
9   retourne C
```

Pour que cette fonction n'ajoute pas dans le cache un triplet $(S, \mathsf{freq}(S), \mathsf{ferm}(S) \setminus S)$ inutile, il faut que le cache ne contiennent pas déjà un triplet $(T, \mathsf{freq}(T), \mathsf{ferm}(T) \setminus T)$ tel que $T \subseteq S$ et $\mathsf{ferm}(T) = \mathsf{ferm}(S)$. Dans notre cas, cela n'est pas possible car cela signifierait que la fréquence et la fermeture de S peuvent être inférées à partir du cache. Or il est clair que dans l'algorithme iCoCo, les triplets qui sont insérés dans le cache contiennent uniquement des itemsets dont la fréquence et la fermeture n'ont pu être inférées à partir du cache.

Lorsque les requêtes σ_i qui ont été évaluées par l'algorithme n'utilisent que des contraintes anti-monotones, il est possible d'optimiser l'algorithme. C'est ce cas plus restrictif qui est exposé dans notre article [JB02c]. Dans ce cas, l'algorithme ne calcule que des itemsets libres ce qui permet de faire plusieurs optimisations.

Premièrement, comme la fermeture est une fonction croissante, la fermeture d'un itemset contient la fermeture de tous ses sous ensembles. On peut donc écrire la fermeture d'un itemset S comme une réunion disjointe :

$$\mathsf{ferm}(S) = S \cup \left(\bigcup_{T \subset S} \mathsf{ferm}(T) \setminus S \right) \cup P.$$

L'ensemble P est appelé la fermeture propre de S et $\bigcup_{T \subset S} \mathsf{ferm}(T) \setminus S$ la fermeture héritée. Ainsi, il n'est pas nécessaire de stocker dans le cache la fermeture héritée. Elle peut être calculée au moment de la génération des candidats en faisant la réunion des fermetures des sous ensembles de S (évidemment, il ne faut considérer que les sous ensembles de S de taille $|S| - 1$). Pendant la passe sur les données, on peut ainsi se limiter au calcul de la

fermeture propre. Comme le calcul de cette fermeture est faite par des intersections, si on retire la fermeture héritée, les intersections se feront avec des listes plus courtes et seront donc plus efficaces. Cette optimisation n'est pas utilisable dans le cas général car on ne connaît pas forcement les fermetures de tous les sous ensembles de S. Il n'est donc pas possible de calculer sa fermeture hérité à partir de la fermeture de ses sous ensembles.

Deuxièmement, le cache sera nécessairement fermé vers le bas, i.e., si le triplet $(L, \mathsf{freq}(L), \mathsf{ferm}(L) \setminus L)$ est dans le cache alors les triplets $(S, \mathsf{freq}(S), \mathsf{ferm}(S) \setminus S)$ pour tous les sous ensembles S de L sont aussi dans le cache. Par conséquent, si un itemset S n'est pas dans le cache, alors aucun de ses sur ensembles ne peut y être (autrement dit, la propriété d'être dans le cache est anti-monotone). On peut utiliser cette propriété en ajoutant une marque sur les itemsets qui indique qu'un itemset est dans le cache. Lors de la génération des candidats, la marque de chaque itemset est initialisée avec la conjonction des marques de ses sous ensembles. Ensuite, lors de la passe sur le cache, seul les itemsets dont la marque est à vrai sont recherchés dans le cache (c'est-à-dire uniquement ceux dont tous les sous ensembles sont dans le cache).

Troisièmement, la gestion du cache est simplifiée. Comme le cache contient des itemsets libres et que l'algorithme ne manipule que des itemsets libres (par opposition à des libres contextuels), la fréquence et la fermeture d'un itemset libre L peuvent être inférées à partir du cache si et seulement si le triplet $(L, \mathsf{freq}(L), \mathsf{ferm}(L) \setminus L)$ est dans le cache. La fonction **passe_cache** est donc simplifiée. La fonction d'insertion des triplets dans le cache est elle aussi simplifiée : il n'y a plus de problème de redondance. En effet, s'il existe deux itemsets $S \subset T$ ayant même fermeture alors T ne peut pas être libre (il est inclus dans la fermeture de S qui est un de ses sous ensembles stricts). Par conséquent, tous les triplets sont insérés dans le cache sans faire aucune vérification.

Pour utiliser ces optimisations, chaque enregistrement représentant un itemset contient maintenant 5 champs :
- un champ item pour stocker les items ;
- un champ freq pour la fréquence ;
- un champ hferm pour la fermeture héritée ;
- un champ pferm pour la fermeture propre ;
- et un champ dc pour indiquer si l'itemset est dans le cache.

Dans la fonction de génération des candidats **gen_cand**, les champs hferm et pferm sont initialisés avec l'ensemble vide et le champ dc est initialisé à vrai.

Les nouvelles fonctions sont alors :

élagage_coco2(Cand, $\mathcal{S}_k$)
1 **pour tout** $S \in$ Cand **faire**
2 **pour tout** sous ensemble T de S.item de taille k **faire**
3 **si** $T \notin \mathcal{S}_k$.item **ou** S.item $\subseteq \mathsf{ferm}(T)$ **alors**
4 Cand := Cand $\setminus \{S\}$
5 **sinon**
6 S.dc := S.dc $\wedge$ T.dc
7 S.hferm := S.hferm $\cup$ (T.hferm $\cup$ T.pferm) $\setminus$ S.item
8 **finsi**
9 **fait**
10 **fait**
11 **retourne** Cand

Cette nouvelle fonction d'élagage initialise les fermetures hérités et les marques.

passe_cache2(Cand, C)
1 **pour tout** itemset $S \in$ Cand tel que S.dc $= vrai$ **faire**
2 **si** $(S.\text{item}, f, F) \in C$ **alors**
3 $S.\text{freq} := f$
4 $S.\text{pferm} := F$
5 **finsi**
6 **fait**

passe_bd_close2(Cand, bd)
1 **pour toute** transaction $t \in bd$ **faire**
2 **pour tout** itemset candidat $S \in$ Cand **faire**
3 **si** $S.\text{item} \subseteq t$ **alors**
4 $S.\text{freq} := S.\text{freq} + 1$
5 **si** $S.\text{freq} = 1$ **alors** $S.\text{pferm} := t \setminus (S.\text{item} \cup S.\text{hferm})$
6 **sinon** $S.\text{pferm} := S.\text{pferm} \cap t$ **finsi**
7 **finsi**
8 **fait**
9 **fait**

Lorsque la première transaction qui contient S est rencontrée, la fermeture propre de S est initialisée avec l'ensemble des items de la transaction qui ne sont ni dans S ni dans sa fermeture héritée. Ainsi, cette liste est plus courte et donc les intersections qui sont faites ensuite sont plus rapides.

4.5.1 Contenu du cache

Dans cette section, nous étudions le contenu du cache à la fin d'une séquence de requêtes $\sigma_i = (bd, 2^{\mathsf{Items}}, \mathcal{C}_{\mathsf{am}_i} \wedge \mathcal{C}_{\mathsf{m}_i}, \mathsf{freq})$, $1 \leq i \leq n$ où les contraintes $\mathcal{C}_{\mathsf{am}_i}$ sont anti-monotones et les contraintes $\mathcal{C}_{\mathsf{m}_i}$ sont monotones.

Après l'évaluation de la première requête, le cache contient tous les itemsets dont la fréquence et la fermeture ont été calculées par l'algorithme, c'est-à-dire qu'il contient les triplets $(S, \mathsf{freq}(S), \mathsf{ferm}(S) \setminus S)$ pour tous les itemsets appartenant à la théorie $\mathsf{Th}(\mathcal{C}_{\mathsf{am}_1} \wedge \mathcal{C}_{\mathsf{m}_1} \wedge \mathcal{C}_{\mathsf{libre\text{-}c}})$ et pour certains itemsets appartenant à la frontière de cette théorie (voir plus haut). Lorsqu'une nouvelle requête σ_2 est évaluée, les triplets correspondant aux itemsets appartenant à la théorie $\mathsf{Th}(\mathcal{C}_{\mathsf{am}_2} \wedge \mathcal{C}_{\mathsf{m}_2} \wedge \mathcal{C}_{\mathsf{libre\text{-}c}})$ et à sa frontière sont ajoutés dans le cache. Comme nous l'avons vu, ces ajouts peuvent aussi être suivis de la suppression de certains triplets qui sont devenus redondants et donc inutiles.

Dans ce cas général, il est difficile de caractériser les éléments qui se trouvent dans le cache après l'évaluation de plusieurs requêtes. Pour simplifier le problème, nous supposons que les triplets correspondants aux itemsets de la frontière ne sont pas ajoutés dans le cache. Cela signifie que le cache contient exactement l'information relative aux itemsets qui satisfont la contrainte de l'une des requêtes. Remarquons qu'il est possible de modifier l'algorithme iCoCo pour qu'il n'insère effectivement dans le cache que ces itemsets. Il suffit pour cela de faire l'insertion des éléments de Cand dans le cache après le test de la contrainte. Ainsi, les itemsets de la frontière négative (qui ne satisfont pas la contrainte) ne sont pas insérés dans le cache. Dans ce cas, les triplets qui se trouvent dans le cache après l'évaluation d'une séquence de requêtes sont caractérisés par la proposition suivante.

Proposition 30 *Considérons le cache construit pendant l'évaluation d'une séquence de requête* $\sigma_i = (bd, 2^{\mathsf{Items}}, \mathcal{C}_{\mathsf{am}_i} \wedge \mathcal{C}_{\mathsf{m}_i}, \mathsf{freq})$, $1 \leq i \leq n$ *où les contraintes* $\mathcal{C}_{\mathsf{am}_i}$ *sont anti-monotones et* $\mathcal{C}_{\mathsf{m}_i}$ *monotones. On suppose que seuls les itemsets de* $\mathsf{Th}(\mathcal{C}_{\mathsf{am}_i} \wedge \mathcal{C}_{\mathsf{m}_i} \wedge \mathcal{C}_{\mathsf{libre\text{-}c}})$ *sont ajoutés dans le cache lors de l'évaluation de chaque requête. Soit* Cl *une classe d'équivalence pour la relation* $\sim_f$ *d'égalité des fermetures, alors, dans cette classe d'équivalence, le cache contient exactement les triplets de la forme* $(S, \mathsf{freq}(S), \mathsf{ferm}(S) \setminus S)$ *pour les itemsets* S *minimaux dans* Cl *qui satisfont la disjonction*

$$\bigvee_{1 \leq i \leq n} (\mathcal{C}_{\mathsf{am}_i} \wedge \mathcal{C}_{\mathsf{m}_i}).$$

Une des conséquences de cette proposition est que le contenu du cache ne dépend pas de l'ordre dans lequel les requêtes sont évaluées. De plus, le cache contient toute l'information nécessaire pour calculer la fréquence de tous les itemsets qui satisfont la contrainte de l'une des requêtes. Finalement, le contenu de notre cache est très proche des représentations condensées d'ensembles de requêtes présenté dans [GLD02]. Il y a cependant plusieurs différences :

- Pour notre cache, nous utilisons une représentation à base de libres contextuels plus leurs fermetures. Dans [GLD02], les représentations condensées utilisent soit des libres contextuels soit des "fermés contextuels" (i.e., les auteurs généralisent les fermés de la même manière que les libres sont généralisés en libres contextuels).
- L'information contenue dans leurs représentations condensées permet, lors de la régénération, de déterminer la liste des requêtes pour lesquelles un itemset est solution. Nous ne gardons pas cette information dans le cache (bien que cela soit possible).

Ces différences s'expliquent par des objectifs différents. Notre but est l'optimisation de séquences de requêtes, alors que dans [GLD02], l'objectif principal est une étude théorique de la généralisation des représentations condensées à un ensemble de requêtes.

Dans notre algorithme iCoCo, les itemsets de la frontière négative dont la fréquence et la fermeture sont calculées par l'algorithme sont aussi mis dans le cache. Par conséquent, le cache contient plus d'information que s'il ne contenait pas ces frontières. Du point de vue de l'optimisation d'une séquence de requêtes, cela est préférable car plus le cache contient d'information et moins l'évaluation des requêtes suivantes nécessitera d'accès à la base de données.

4.6 Expérimentations

Afin de valider notre approche, nous avons implémenté un premier prototype utilisant un cache. Cependant, ce prototype ne gère pour l'instant que les contraintes anti-monotones. Ce prototype est donc plutôt une version de l'algorithme Close avec un cache qu'une véritable implémentation de notre algorithme iCoCo. Nous avons utilisé toutes les optimisations décrites dans la section 4.5 dans le cas où la requête n'utilise que des contraintes anti-monotones.

L'objectif de ces expérimentations est double :

- nous voulons d'abord démontrer que faire des extractions de représentations condensées sous contraintes en utilisant un cache est utile, c'est-à-dire que le gain d'efficacité apporté par le cache est significatif ;
- ensuite, nous voulons vérifier que la taille de notre cache est réellement plus petite que la taille des collections d'itemsets complètes et donc que les caches utilisant les itemsets comme dans [NDD99].

4.6.1 Efficacité de l'utilisation du cache

Le cache stocke les fréquences et les fermetures d'itemsets qui ont été calculées par l'algorithme dans des extractions précédentes. Pour tester l'efficacité du cache, nous avons mesuré le gain apporté lorsqu'une partie des itemsets dont l'algorithme doit calculer la fréquence et la fermeture sont dans le cache.

L'étape la plus coûteuse de l'algorithme est le calcul de la fréquence et de la fermeture des itemsets candidats. Si l'algorithme n'a besoin de faire ce calcul que pour 50% des itemsets, on peut donc penser, en première approximation, que l'extraction se fera deux fois plus vite.

Pour une requête d'extraction σ, nous avons donc utilisé l'algorithme pour évaluer σ une fois sans utiliser de cache et ensuite en utilisant un cache contenant une proportion variable des itemsets utilisés par l'algorithme. Nous avons ensuite comparé les temps d'extraction et la proportion des itemsets qui étaient dans le cache. Plus formellement, si $\sigma = (\mathcal{C}_{\mathsf{am}}, \mathsf{freq})$ est une requête, l'algorithme doit calculer la fréquence et la fermeture des itemsets de

$$I(\sigma) = \mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{libre}}) \cup [\mathcal{B}d^{-}(\mathsf{Th}(\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{libre}})) \cap \mathsf{Th}(\mathcal{C}_{\mathsf{libre}})]$$

pour répondre à cette requête. Nous définissons donc la proportion de ces itemsets qui sont dans le cache par :

$$p(\sigma, C) = \frac{|I(\sigma) \cap C|}{|I(\sigma)|}.$$

Nous notons $t(\sigma)$ le temps d'évaluation de la requête σ lorsque l'extraction se fait sans utiliser de cache et $t(\sigma, C)$ le temps d'extraction avec un cache C. Nous définissons le gain d'efficacité par le rapport :

$$g(\sigma, C) = \frac{t(\sigma) - t(\sigma, C)}{t(\sigma)}.$$

Dans la figure 4.1, nous avons tracé le gain $g(\sigma, C)$ en fonction de $p(\sigma, C)$ pour un cache fixé et différentes valeurs de σ. Pour ces expériences, nous avons utilisé les jeux de données mushroom et connect-4 disponibles sur le site de l'université d'Irvine (Californie) `http://www.ics.uci.edu/~mlearn/MLRepository.html`. Le jeu de données mushroom contient 8124 lignes et 128 colonnes avec 23 attributs vrais par ligne. Le jeu de données connect-4 contient 67557 lignes et 129 colonnes avec 43 attributs vrais par ligne.

4.6.2 Taille du cache

Dans cette deuxième série d'expériences, nous avons comparé la taille de notre cache avec la taille de la collection complète des itemsets contenant la même quantité d'information. Notre cache est constitué de triplets de la forme $(S, \mathsf{freq}(S), \mathsf{pferm}(S))$. Si ces triplets sont stockés sans utiliser de compression particulière, chacun d'eux occupe une place égale à la taille de S plus la taille de $\mathsf{pferm}(S)$ plus un pour stocker la fréquence. Nous définissons donc la taille d'un cache C par :

$$\mathit{taille}(C) = \sum_{(S,f,F) \in C} (|S| + |F| + 1).$$

Pour une collection d'itemsets Coll, nous définissons sa taille par :

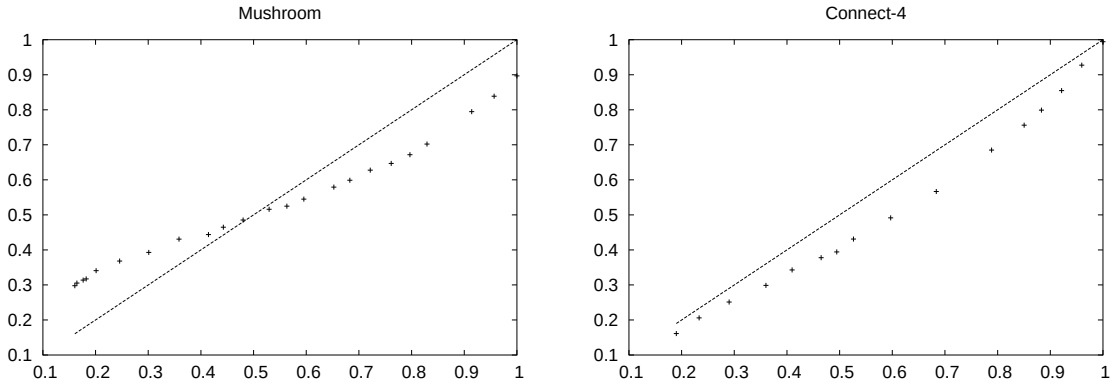


FIG. 4.1 – Gain d'efficacité en fonction de la proportion des itemsets dans le cache.

$$taille(Coll) = \sum_{S \in Coll} (|S| + 1).$$

Dans nos expériences, nous avons construit les caches en évaluant les résultats de requêtes du type $\sigma_\gamma = (\mathcal{C}_{\gamma\text{-}\mathsf{freq}}, \mathsf{freq})$ avec notre prototype. Ces caches contiennent donc les triplets $(S, \mathsf{freq}(S), \mathsf{pferm}(S)$ pour tous les itemsets appartenant à

$$I(\sigma_\gamma) = \mathsf{Th}(\mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{libre}}) \cup [\mathcal{B}d^-(\mathsf{Th}(\mathcal{C}_{\gamma\text{-}\mathsf{freq}} \wedge \mathcal{C}_{\mathsf{libre}})) \cap \mathsf{Th}(\mathcal{C}_{\mathsf{libre}})].$$

Nous les avons comparés aux collections d'itemsets qui seraient utilisées pour calculer la réponse à ces mêmes requêtes en n'utilisant pas les représentations condensées, c'est-à-dire les itemsets appartenant à la collection

$$\mathsf{Th}(\sigma_\gamma) \cup \mathcal{B}d^-(\sigma_\gamma).$$

La figure 4.2 montre les tailles de ces deux types de collections.

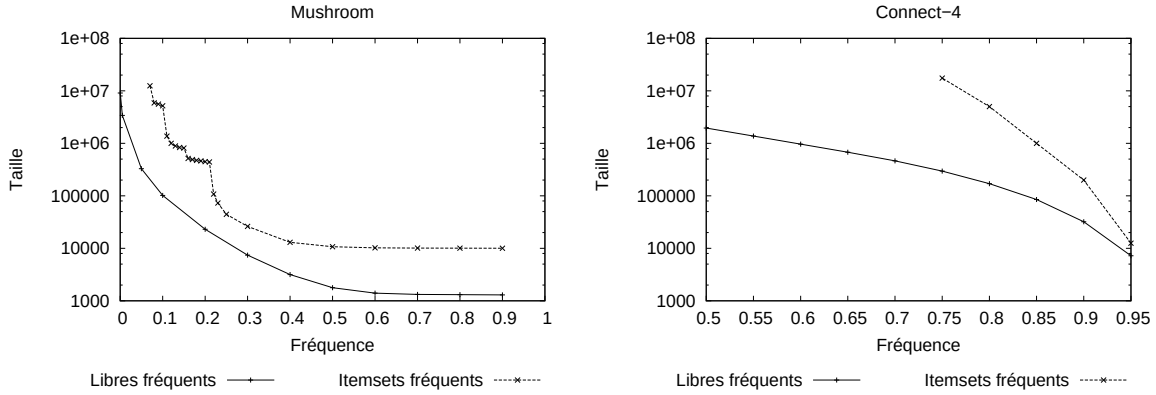


FIG. 4.2 – Comparaison des tailles entre notre cache et les collections d'itemsets pour les mêmes seuils de fréquence.

On constate bien que la taille de notre cache est très nettement inférieure. Cela signifie aussi que pour une taille équivalente, notre cache contient beaucoup plus d'information.

4.7 Nouvelles stratégies d'évaluation de requêtes inductives

Notre nouvel algorithme iCoCo utilisant un cache permet d'utiliser les stratégies déjà présentées dans le chapitre 3 pour l'évaluation d'une requête inductive. L'utilisation d'un cache permet cependant de concevoir de nouvelles stratégies d'évaluation même pour une seule requête. Si la requête à évaluer contient une contrainte avec des disjonctions, par exemple de la forme $(\mathcal{C}_{\mathsf{am}_1} \wedge \mathcal{C}_{\mathsf{m}_1}) \vee \cdots \vee (\mathcal{C}_{\mathsf{am}_n} \wedge \mathcal{C}_{\mathsf{m}_n})$, notre algorithme CoCo n'est pas capable de la traiter. La seule solution est de faire l'évaluation de chaque disjonction séparément, mais alors, il est possible que certains calculs soient faits plusieurs fois (si l'intersection des théories $\mathsf{Th}(\mathcal{C}_{\mathsf{am}_i} \wedge \mathcal{C}_{\mathsf{m}_i})$ n'est pas vide). En utilisant un cache, on assure que cela ne sera pas le cas car les résultats des évaluations des différents termes de la disjonction seront stockés dans le cache et seront donc utilisés pour les évaluations des termes suivants. Évaluer une requête avec des disjonctions se ramène donc à l'évaluation d'une séquence de requêtes.

Dans le cas d'une séquence de requêtes, cet algorithme offre aussi de nouvelles possibilités. Considérons une séquence de requêtes $\sigma_i = (bd, 2^{\mathsf{Items}}, \mathcal{C}_{\mathsf{am}_i} \wedge \mathcal{C}_{\mathsf{m}_i}, \mathsf{freq})$. Il y a alors deux stratégies principales pour évaluer cette séquence avec notre algorithme :

- utilisation d'une collection matérialisée ;
- extraction directe.

Dans le premier cas, on suppose que le cache contient une collection de triplets de la forme $(S, \mathsf{freq}(S), \mathsf{ferm}(S) \setminus S)$ qui a déjà été calculée. Par exemple, cette collection peut être matérialisée pendant la nuit en prévision d'une session d'extraction le lendemain, ou cela peut être le cache construit pendant une session précédente sur la même base de données binaire (éventuellement par un autre utilisateur). L'avantage d'avoir une collection matérialisée importante dès le début de l'extraction est que le temps d'évaluation des nouvelles requêtes peut être très bas, i.e., du même ordre de grandeur que les temps correspondants à des phases de sélection dans des collections de motifs matérialisées. Cependant, la collection matérialisée n'est plus une limite stricte. Les résultats des nouvelles requêtes peuvent sortir de cette collection sans pénaliser le temps d'extraction de manière exagérée.

Dans le second cas, le cache est initialement vide. Notre algorithme permet alors de le remplir au fur et à mesure que les requêtes sont évaluées, l'algorithme se servant du cache construit avec les requêtes précédentes pour évaluer toute nouvelle requête. Ainsi, les calculs faits pendant l'évaluation d'une requête n'ont pas a être recommencés et, le cache contenant de plus en plus d'information, le coût moyen d'évaluation d'une requête baisse progressivement. Finalement, ce coût descend au niveau de celui d'une simple sélection dans une collection matérialisée lorsque la taille du cache est devenue suffisamment importante.

Ces stratégies sont conçues pour le cas où les requêtes sont toutes des requêtes étendues, i.e., l'utilisateur demande toujours, en plus des itemsets, la valeur d'une fonction d'évaluation comme la fréquence. Bien évidemment, nos algorithmes ne sont pas adaptés pour calculer le résultat d'une requête simple. Dans ce cas, un algorithme de calcul direct des frontières est sans doute plus efficace (comme dans [dRK01]). Par contre, si l'utilisateur formule tantôt des requêtes simples tantôt des requêtes étendues, on peut voir le cache comme l'endroit où serait stocké l'information sur les valeurs de la fonction d'évaluation. Le système permettrait de manipuler les résultats des requêtes par l'intermédiaire de leurs frontières lorsque l'utilisateur n'utilise que des requêtes simples, certains travaux vont dans ce sens [dR02a] en proposant des techniques pour manipuler directement les résultats de requêtes (réunion, intersection ...). Dans le cas où des requêtes étendues sont utilisées, alors le système utiliserait un cache ou un calcul direct à partir de la base de données pour calculer la fonction d'évaluation lorsque cela est nécessaire.

Notre algorithme reste limité au cas où la base de données binaire ne change pas. Lorsqu'on se place à l'échelle d'une session d'extraction, cette hypothèse peut être raisonnable, i.e., il est probable qu'un utilisateur fera plusieurs extractions sur une base binaire *bd* avant de la changer. Dans le cas où cette base est modifiée, et si les modifications ne sont pas faites au niveau du schéma, il est possible de propager ces modifications à un cache bâtit sur cette base de données sans le recalculer totalement. Bien qu'il n'existe pas encore à notre connaissance de technique de mise à jour d'une collection d'itemsets libres ou fermés lors de la modification de la base de données, ces techniques existent dans le cas des itemsets ou des règles d'association [TBAR97, AEMT01, AFLM99, CHNW96]. Elles sont utilisées lorsque des tuples sont insérés ou supprimés dans la base de données et il est sans doute possible de les adapter au cas des représentations condensées. Dans le cas où la base de données change à chaque extraction sans qu'il soit possible d'utiliser cette technique, notre algorithme se ramène à une extraction sous contrainte simple, i.e., comme l'algorithme CoCo.

Conclusion

Contributions principales

Extraction d'itemsets sous contraintes. Dans ce mémoire, nous avons d'abord présenté, dans le premier chapitre, le cadre des bases de données inductives qui est de plus en plus utilisé dans la communauté fouille de données. Une base de donnée inductive est une extension des bases de données classiques qui contient, en plus des données, des collections de motifs définies en intention sur ces données. Un utilisateur peut donc interroger les données comme dans une base de données classique, mais il peut aussi utiliser des requêtes de sélection sur les motifs en spécifiant par le biais de contraintes les motifs qu'il souhaite extraire. Pour l'utilisateur tout ce passe comme si tous les motifs étaient disponibles, c'est au solveur d'utiliser efficacement les contraintes pour calculer efficacement les motifs. Ce cadre formel nous a permis de motiver la recherche d'algorithmes génériques pour l'extraction de motifs sous des contraintes les plus diverses possibles.

Dans le deuxième chapitre, nous avons présenté notre algorithme permettant l'extraction d'itemsets avec une contrainte qui est une conjonction d'une contrainte monotone et d'une contrainte anti-monotone [BJ00].

Cet algorithme nous a permis de présenter les algorithmes d'extraction des représentations condensées (δ-libres et fermés) sous un nouveau point de vue : ce sont respectivement des instanciations de notre algorithme avec les contraintes $\mathcal{C}_{\gamma\text{-freq}} \wedge \mathcal{C}_{\delta\text{-libre}}$ et $\mathcal{C}_{\gamma\text{-freq}} \wedge \mathcal{C}_{\mathsf{clos}}$. Cela nous a également permis de comprendre pourquoi les algorithmes d'extraction des clos calculent en fait d'abord les libres : c'est parce que la contrainte $\mathcal{C}_{\mathsf{libre}}$ est anti-monotone alors que $\mathcal{C}_{\mathsf{clos}}$ n'est ni anti-monotone ni monotone.

Extraction de représentations condensées sous contraintes. Dans le troisième chapitre, nous avons introduit notre nouvel algorithme d'extraction de représentations condensées sous contraintes, CoCo [BJ01b, JB02b]. Alors que les algorithmes existants se contentaient d'extraire des δ-libres ou des clos fréquents, notre algorithme permet l'extraction de δ-libres et de clos avec une contrainte qui est une conjonction d'une contrainte monotone et d'une contrainte anti-monotone. Pour résoudre les problèmes posés par l'ajout d'une contrainte monotone, nous avons introduit une généralisation des itemsets δ-libres, les δ-libres contextuels relativement à une contrainte monotone. Notre algorithme CoCo traite donc le cas des conjonctions de contraintes de la forme $\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\delta-\text{libre-c}}$. Nous avons ensuite présenté des nouvelles stratégies d'extraction d'itemsets ou de règles d'association sous contraintes utilisant cet algorithme [JB02b, JB02a, BJ01a].

Nous montrons dans nos expérimentations que l'utilisation des représentations condensées sous contraintes améliore considérablement les temps d'extraction si l'on compare avec des stratégies qui n'utilisent que les représentations condensées ou que les contraintes. De plus,

nous bénéficions de tous les avantages des représentations condensées : Tout d'abord, elles permettent de régénérer tous les itemsets rapidement (car sans accès aux données). Cela signifie qu'il est possible de régénérer efficacement tous les itemsets qui satisfont la conjonction $\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}}$ ainsi que leurs fréquences à partir de ceux qui satisfont $\mathcal{C}_{\mathsf{am}} \wedge \mathcal{C}_{\mathsf{m}} \wedge \mathcal{C}_{\delta-\mathsf{libre\text{-}c}}$. Ensuite, l'extraction de ces représentations condensées est très efficace dans les données corrélées ou une extraction directe des itemsets est trop coûteuse. Le paramètre δ des δ-libres permet d'ailleurs de faire un compromis entre vitesse d'extraction et précision dans la régénération des fréquences : plus δ est grand et plus l'extraction est rapide, au détriment de la précision de la fréquence des itemsets régénérés.

Application aux règles avec négations. Notre algorithme CoCo nous a permis de nous attaquer avec succès au problème de l'extraction des règles d'association avec négations. Les règles d'association traditionnelles traduisent des relations sur la présence de différents attributs. L'idée des règles avec négations est de permettre aux règles d'exprimer aussi des relation sur l'absence de certains attributs. Pour cela, on crée, pour chaque attribut `A`, un nouvel attribut "négatif" qui prend dans chaque ligne comme valeur la négation logique de la valeur de `A`. Les règles d'association dans cette base de données complétée peuvent alors exprimer l'absence d'un attribut, cela correspond à la présence de l'attribut négatif correspondant. Il faut bien remarquer que cette opération de complétion de la base de données augmente considérablement la densité de celle-ci. Si on voit la base de données comme une matrice binaire, la densité correspond à la proportion moyenne de "1" dans la matrice. Les bases de données dans lesquelles les itemsets et les règles sont extraits sont généralement peu denses, mais la base de données complétée avec les attributs négatifs a une densité de 1/2. Les bases de données denses posent beaucoup de problèmes car, dans ce cas, le nombre d'itemsets potentiellement fréquents est très grand même avec des seuils de fréquence faibles. L'utilisation de notre algorithme CoCo permet cependant d'extraire à des seuils de fréquence qui ne peuvent être atteints par les autres méthodes existantes. En effet, l'utilisation des représentations condensées permet l'extraction dans des données fortement corrélées et l'utilisation simultanée des contraintes permet de se limiter à des itemsets potentiellement plus intéressants. Les contraintes que nous avons utilisées dans nos expériences imposent que les itemsets extraits contiennent au moins un, deux ou trois attributs positifs. En effet, on peut considérer que les itemsets ou les règles qui ne concernent que des attributs négatifs sont moins intéressants que ceux qui contiennent aussi des attributs positifs et surtout ces contraintes permettent de fortement diminuer le nombre d'itemsets fréquents. Évidemment, ces contraintes ne sont pas les seules qui soient intéressantes. On pourrait par exemple s'intéresser à l'extraction de règles ne contenant des attributs négatifs que si l'ajout de ces derniers permet une augmentation significative de la confiance. La difficulté sera, là encore, de parvenir à "pousser" cette contrainte pour permettre une extraction efficace.

Application à l'analyse du transcriptome. L'application la plus intéressante de nos travaux concerne l'extraction d'itemsets et de règles dans des données d'expression de gènes lors d'une collaboration avec des biologistes. Ces dernières années, les biologistes ont consacré beaucoup de moyens au décryptage du génome de plusieurs animaux, plantes ou autres organismes. L'objectif de ces recherches est de mieux comprendre le fonctionnement des cellules. Cependant, même si le génome de nombreuses espèces est actuellement disponible, il faut encore l'interpréter. Les résultats du décryptage sont des longues chaînes composées des caractères A,C,T et G qui sont les initiales des noms des bases porteuses de l'information génétique dans l'ADN. Une fois que ces chaînes de caractères sont connues, encore faut-il

les comprendre. En particulier, il faut réussir à localiser les gènes dans l'ADN (on considère que pour les mammifères, moins de 5% de l'ADN est codant) et à comprendre leur rôle. En première approximation, on peut considérer que les gènes sont les "formules" qui permettent à la cellule de fabriquer les protéines dont elle a besoin. À un instant donné, on peut caractériser (toujours en simplifiant) l'état d'une cellule par les quantités de chaque type de protéine qu'elle produit : ce sont les niveaux d'expression des gènes. Les biologistes ont observé que le niveau d'expression de chaque gène a une influence sur le niveau d'expression d'autres gènes. Ces relations peuvent être représenté sous la forme d'un graphe dans lequel les sommets sont les gènes et les arêtes indiquent ces influences (qui peuvent être positives ou négatives). Il y a donc un double challenge : comprendre le rôle de chaque gène et parvenir à découvrir ce graphe des interactions entre gènes. Notre collaboration avec les biologistes a montré que nos techniques peuvent être utilisées pour aider à résoudre ces deux problèmes. Les données que nous avons utilisées indiquent les niveaux d'expression de différents gènes dans des cellules. Les lignes de notre base de données sont les cellules et les colonnes, les gènes. Nous avons alors fait des extractions d'itemsets libres et fermés dans cette base de données. Cette technique semble très prometteuse d'après les biologistes et ces travaux ont d'ailleurs donné lieu à une publication dans une revue de biologie [BBJ+02].

Optimisation des séquences de requêtes. Dans le premier chapitre, nous avons souligné le caractère itératif et interactif du processus d'extraction de connaissances. L'utilisateur demande en effet souvent l'évaluation d'une séquence de requête et non pas d'une seule requête. Étant donné le coût important de l'évaluation d'une seule requête, il faut absolument supporter cette interactivité pour que les temps de calcul soient acceptables. Dans le chapitre 4, nous avons donc présenté une version interactive iCoCo [JB02c] de notre algorithme CoCo. Pour gérer cette interactivité, notre algorithme utilise les résultats des requêtes précédentes en les stockant dans un cache. Pour ce cache, nous avons utilisé une fois de plus les représentations condensées. Par rapport aux autres algorithmes existants, l'utilisation des représentations condensées permet d'avoir un cache plusieurs ordres de grandeur plus petit dans le cas de données corrélées.

Tous ces travaux ont été effectués dans le cadre du projet européen cInQ [ec] sur la découverte de connaissances à l'aide de requêtes inductives. Nos travaux sont en particulier utilisés dans les rapports d1 [cc02b] (théorie des bases de données inductives), d2 [cc02c] (définition du domaine ITEMS (itemsets, épisodes et règles)), d6 [cc02d] (solveurs pour le domaine ITEMS) et d10 [cc02a] (stratégies d'évaluation des requêtes complexes).

Perspectives

Nous pensons qu'il est possible de distinguer deux perspectives principales à notre travail. Tout d'abord, il faut continuer l'étude théorique des bases de données inductives et de l'évaluation des requêtes inductives. Ensuite, il semble important de se fixer des objectifs applicatifs, i.e., de concevoir des bases de données inductives dédiées à des applications particulières.

Perspectives théoriques : bases de données inductives. Dans nos travaux, nous avons tenté de faire ressortir les principes utiles dans l'extraction d'itemsets pour certains types de contraintes, les contraintes monotones et anti-monotones. Or, dans certains cas, il est possible d'utiliser efficacement des contraintes qui ne sont ni monotones ni anti-monotones,

par exemple pour l'extraction de séquences satisfaisant une expression régulière [ALB02]. Enfin, il n'existe pas toujours de méthode efficace pour traiter certaines autres contraintes qui ne sont ni anti-monotones ni monotones, comme la contrainte de confiance minimum. D'une manière générale, il faut se pencher sur les relations entre l'espace de recherche (par exemple le treillis des itemsets), la manière dont il est exploré (par exemple, niveau par niveau) et les contraintes. De nombreuses recherches ont été menées dans cette direction dans le domaine de la programmation logique inductive, il serait sans doute profitable de les exploiter.

Une autre perspective concerne l'étude de solveurs génériques pour traiter des contraintes complexes (i.e., combinaisons de contraintes primitives). Par exemple, en supposant que l'on connaisse un algorithme efficace pour extraire les 10 règles ayant la meilleure confiance, comment le combiner avec un autre algorithme pour trouver les 10 règles ayant la meilleure confiance et satisfaisant une contrainte syntaxique. L'intérêt d'un solveur générique est qu'il permet une plus grande liberté de choix quant aux contraintes à "pousser" et donc quant à la stratégie à utiliser. Il faut remarquer que le choix de la stratégie la plus efficace est un problème ouvert même dans le cas ou l'on ne considère que des conjonctions de contraintes monotones et anti-monotones (il est toujours plus efficace de pousser toutes les contraintes anti-monotones, par contre ce n'est pas toujours le cas pour les contraintes monotones). Ce problème est rendu difficile principalement parce qu'il est difficile de connaître a priori la sélectivité d'une contrainte. Cette recherche de solveurs génériques et de stratégies d'évaluation est très importante dans le cadre d'une vision base de données inductive, c'est d'ailleurs l'un des thèmes de recherche du projet cInQ.

Dans ce cadre des bases de données inductives, nos travaux sur l'évaluation de séquences de requêtes ouvrent également des perspectives de recherche intéressantes. On retrouve là des problématiques proches des bases de données, en particulier en ce qui concerne la matérialisation des résultats. L'évaluation des requêtes inductives est généralement beaucoup plus coûteuse que pour une requête classique (e.g., relationnelle) et la taille du résultat peut aussi être beaucoup plus grande. Par conséquent, il faut trouver des techniques adaptées pour faire un compromis entre les temps d'extraction et le stockage des résultats. Il faut évidemment aussi pouvoir réutiliser des résultats matérialisés lors de l'évaluation d'une nouvelle requête. Nous avons montré que, dans le cas des itemsets, les représentations condensées peuvent être utilisées à ces fins. Néanmoins, il n'existe pas encore, à notre connaissance, de représentations condensées pour d'autres motifs que les itemsets, et ce, même dans le cas de motifs proches comme les séquences.

Finalement, on peut encore soulever le problème de la prise en compte de la connaissance du domaine. Est-il possible de la représenter et de l'utiliser efficacement ? En particulier, intégrer cette connaissance peut permettre d'améliorer la qualité des résultats en évitant de présenter à l'utilisateur des propriétés des données qui découlent manifestement de cette connaissance. Il nous semble en particulier intéressant d'étudier les liens entre les connaissances du domaine et les représentations condensées. En effet, dans le cas des itemsets fermés, on peut remarquer qu'une partie de cette connaissance, si elle peut se mettre sous la forme d'une règle de confiance 1, sera extraite par l'algorithme et ensuite utilisée pour élaguer l'espace de recherche.

Bases de données inductives dédiées. Parallèlement à cette recherche sur la théorie des bases de données inductive, il semble intéressant de développer des techniques spécifiquement dédiées à une application particuliére. Par exemple, nous avons présenté les travaux de [dRK01] dans lequel les auteurs décrivent une base de donnée spécifique pour extraire des

fragments moléculaires dans des bases de données de molécules. Dans ce cas, les auteurs présentent les types de motifs extraits (fragments moléculaires), les contraintes élémentaires utilisables pour spécifier les fragments intéressants, la manière dont ces contraintes élémentaires peuvent être combinées pour obtenir des contraintes complexes (dans ce cas, la conjonction) ainsi que l'algorithme Molfea (autrement dit le solveur) pour extraire les fragments satisfaisant une contrainte complexe. Dans ce cas, les auteurs ont adapté des techniques d'extraction connues à un cadre applicatif nouveau. Nous pensons que cette démarche est prometteuse.

Annexe A

Intégration de l'élagage de certains libres dans Close

Dans le chapitre 3, nous avons vu comment L'algorithme Closet utilise une technique pour élaguer des itemsets appartenant à une classe dont le fermé a déjà été extrait. Nous allons ici présenter une méthode originale pour intégrer un élagage similaire dans l'algorithme Close. Pour extraire les itemsets fermés, l'algorithme Close extrait tous les itemsets libres et calcule ensuite leurs fermetures. Cependant, il peut y avoir plusieurs itemsets libres ayant la même fermeture, il n'est donc pas nécessaire de tous les extraire : extraire un seul itemset libre par classe d'équivalence est suffisant pour trouver tous les fermés. L'idée est donc d'essayer de n'extraire, si possible, qu'un itemset libre par classe d'équivalence. Cependant, en éliminant certains itemsets libres, on risque de perdre la complétude de l'algorithme : il faut être sûr d'extraire au moins un libre par classe d'équivalence. Autrement dit, il faut trouver une contrainte anti-monotone plus restrictive que $\mathcal{C}_{\mathsf{libre}}$ qui assure que l'on trouve au moins un itemset libre dans chaque classe d'équivalence. Une telle contrainte est introduite dans la proposition suivante.

Proposition 31 *Soit $\preceq$ un ordre total sur les itemsets ayant les propriétés suivantes :*
- *pour tous itemsets X, Y et tout item* `I` *n'apparaissant pas dans X et Y,*

$$X \preceq Y \Longleftrightarrow X \cup \{\texttt{I}\} \preceq Y \cup \{\texttt{I}\}\,;$$

- *pour tous itemsets X et Y, si $|X| < |Y|$ alors $X \prec Y$.*

Soit $\mathcal{C}_{\mathsf{libre}_\preceq}$ la contrainte définie par :

$$\mathcal{C}_{\mathsf{libre}_\preceq}(S) = (S = \min_\preceq \{T \mid \mathcal{C}_{\mathsf{libre}}(T) \wedge \mathsf{ferm}(T) = \mathsf{ferm}(S)\})\,.$$

Alors
- *la contrainte $\mathcal{C}_{\mathsf{libre}_\preceq}$ est anti-monotone ;*
- *pour tout itemset fermé F, il existe un itemset L, et un seul, de fermeture F qui satisfasse $\mathcal{C}_{\mathsf{libre}_\preceq}$.*

Preuve : L'existence et l'unicité de L est immédiate. Si F est un fermé, $\mathcal{C}_{\mathsf{libre}_\preceq}$ est satisfaite par le plus petit (pour l'ordre total $\preceq$) itemset libre qui a F comme fermeture. Montrons l'anti-monotonicité de la contrainte. Soit S un itemset satisfaisant $\mathcal{C}_{\mathsf{libre}_\preceq}$. On remarque d'abord que la taille de S est minimale dans sa classe d'équivalence (sinon il existerait un itemset libre W dans la classe de S de taille plus petite et donc plus petit

pour l'ordre $\preceq$). Soit I un item de S et T le sous ensemble de S tel que $S = T \cup \{\mathtt{I}\}$. Supposons que T ne satisfasse pas $\mathcal{C}_{\mathsf{libre}\preceq}$. Comme T est libre, cela signifie qu'il existe un itemset libre V dans la même classe que T et tel que $V \prec T$.

- Montrons que $V \cup \{\mathtt{I}\}$ est dans la même classe que S. D'une part $T \subseteq \mathsf{ferm}(T) = \mathsf{ferm}(V) \subseteq \mathsf{ferm}(V \cup \{\mathtt{I}\})$, d'autre part $\{\mathtt{I}\} \subseteq \mathsf{ferm}(V \cup \{\mathtt{I}\})$. Donc $T \cup \{\mathtt{I}\} \subseteq \mathsf{ferm}(V \cup \{\mathtt{I}\})$ et par conséquent $\mathsf{ferm}(T \cup \{\mathtt{I}\}) \subseteq \mathsf{ferm}(V \cup \{\mathtt{I}\})$. L'inclusion inverse se démontre de même. Finalement, $\mathsf{ferm}(V \cup \{\mathtt{I}\}) = \mathsf{ferm}(T \cup \{\mathtt{I}\}) = \mathsf{ferm}(S)$, donc $V \cup \{\mathtt{I}\}$ est dans la même classe d'équivalence que S.
- Montrons que $V \cup \{\mathtt{I}\}$ est libre. Comme $V \prec T$, $|V| \leq |T|$ donc $|V \cup \{\mathtt{I}\}| \leq |T \cup \{\mathtt{I}\}| = |S|$. Comme S est de taille minimum dans sa classe d'équivalence, $|V \cup \{\mathtt{I}\}| = |S|$ et il n'existe pas d'itemset de taille plus petite dans la classe de S, Donc $V \cup \{\mathtt{I}\}$ est nécessairement un itemset libre.
- Montrons que $V \cup \{\mathtt{I}\} \prec S$. Comme $V \prec T$ et I n'est pas dans T ou S, $V \cup \{\mathtt{I}\} \prec T \cup \{\mathtt{I}\} = S$.

On a donc montré que $T \cup \{\mathtt{I}\}$ est libre, dans la même classe que S et strictement plus petit que S pour $\preceq$. Donc S ne peut pas satisfaire $\mathcal{C}_{\mathsf{libre}\preceq}$ ce qui contredit notre hypothèse de départ. Donc supposer que T ne satisfait pas $\mathcal{C}_{\mathsf{libre}\preceq}$ était absurde et donc $\mathcal{C}_{\mathsf{libre}\preceq}$ est bien anti-monotone. □

La contrainte $\mathcal{C}_{\mathsf{libre}\preceq}$ permet de sélectionner exactement un itemset libre par classe d'équivalence, celui qui est le plus petit pour l'ordre $\preceq$. L'ensemble des itemsets libres satisfaisant cette contrainte a donc même cardinal que l'ensemble des itemsets fermés tout en permettant de les générer par leurs fermetures.

Comme cette nouvelle contrainte $\mathcal{C}_{\mathsf{libre}\preceq}$ est anti-monotone, il serait facile de modifier Close pour l'utiliser à la place de $\mathcal{C}_{\mathsf{libre}}$. Il suffirait de prendre pour $\preceq$ l'ordre d'énumération des candidats par la fonction de génération des candidats de Close (il est facile de vérifier que cet ordre d'énumération vérifie les hypothèses de la proposition sur l'ordre $\preceq$). Ensuite, après la passe sur les données, les libres dont la fermeture est un fermé déjà extrait précédemment seraient élagués. On diminuerait ainsi la taille de l'espace de recherche parcouru avec un surcoût, tester si la fermeture d'un itemset libre est déjà connue, très limité. Cette technique d'élagage n'a, à notre connaissance, jamais été utilisée dans un algorithme par niveau.

Bibliographie

[AAP00] Ramesh C. Agarwal, Charu C. Aggarwal et V.V.V. Prasad. – Depth first generation of long patterns. In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining (KDD'00)*. – Boston, USA, New York : ACM Press, août 2000. p. 108–118.

[AEMT01] Ahmed Ayad, Nagwa El-Makky et Yousry Taha. – Incremental mining of constrained association rules. In *First International SIAM Conference on Data Mining (SDM01)*. – Chicago, USA, avril 2001. Disp. en ligne (sept. 2002) `http://www.siam.org/meetings/sdm01/pdf/sdm01_01.pdf`, 11 pages.

[AFLM99] Yonatan Aumann, Ronen Feldman, O. Liphstat et Heikki Mannila. – An efficient algorithm for association generation in dynamic databases. *Journal of Intelligent Information Systems*, 1999, vol. 12, n1, p. 61–73.

[AIS93] Rakesh Agrawal, Tomasz Imielinski et Arun Swami. – Mining association rules between sets of items in large databases. In P. Buneman et S. Jajodia éds., *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'93)*. – Washington, D.C., USA, New York : ACM Press, mai 1993. p. 207–216.

[ALB02] Hunor Albert-Lorincz et Jean-François Boulicaut. – *Mining frequent sequential patterns under regular expressions : a highly adaptative strategy for pushing constraints*. – Bâtiment Blaise Pascal, F-69621 Villeurbanne cedex, LISI/LIRIS, Insa Lyon, septembre 2002. 14 pages., Rapport technique.

[AMS+96] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen et A. Inkeri Verkamo. – Fast discovery of association rules. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth et Ramasamy Uthurusamy éds., *Advances in Knowledge Discovery and Data Mining*, p. 307–328. – Menlo Park : AAAI Press, 1996.

[AS95] Rakesh Agrawal et Ramakrishnan Srikant. – Mining sequential patterns. In P. Yu et A. Chen éds., *Proceedings of the 11th International Conference on Data Engineering (ICDE'95)*. – Taipei, Taiwan, IEEE Computer Society, mars 1995. p. 3–14.

[BA99] Roberto J. Bayardo et Rakesh Agrawal. – Mining the most interesting rules. In Surajit Chaudhuri et David Madigan éds., *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining (KDD'99)*. – San Diego, USA, New York : ACM Press, août 1999. p. 145–154.

[BAG00] Roberto J. Bayardo, Rakesh Agrawal et Dimitrios Gunopulos. – Constraint-based rule mining in large, dense databases. *Data Mining and Knowledge Discovery*, 2000, vol. 4, n2/3, p. 217–240.

[Bay98] Roberto J. Bayardo. – Efficiently mining long patterns from databases. In Laura M. Haas et Ashutosh Tiwary éds., *Proceedings of ACM SIGMOD Confe-*

rence on Management of Data (SIGMOD'98). – Seattle, Washington, USA, New York : ACM Press, juin 1998. p. 85–93.

[BB00] Jean-François Boulicaut et Artur Bykowski. – Frequent closures as a concise representation for binary data mining. In Takao Terano, Huan Liu et Arbee L. P. Chen éds., *Proceedings of the 4th Pacif-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'00).* – Kyoto, Japan, Berlin : Springer-Verlag, avril 2000. p. 62–73.

[BBJ00] Jean-François Boulicaut, Artur Bykowski et Baptiste Jeudy. – Towards the tractable discovery of association rules with negations. In Henrik Legind Larsen et al. éds., *Proceedings of the Fourth International Conference on Flexible Query Answering Systems (FQAS'00).* – Warsaw, Poland, Heildelberg : Physica-Verlag, octobre 2000. p. 425–434.

[BBJ+02] Céline Becquet, Sylvain Blachon, Baptiste Jeudy, Jean-François Boulicaut et Olivier Gandrillon. – Strong-association-rule mining for large-scale gene-expression data analysis : a case study on human sage data. *Genome Biology*, 2002, vol. 3, n12. – À paraître, 9 pages.

[BBMM02] Marco Botta, Jean-François Boulicaut, Cyrille Masson et Rosa Meo. – A comparison between query languages for the extraction of association rules. In Y. Kambayashi, W. Winiwarter et M. Arikawa éds., *Proceedings of the 3rd International Conference on Data Warehousing and Knowledge Discovery (DaWaK'02).* – Aix-en-Provence, France, Berlin : Springer-Verlag, septembre 2002. p. 1–10.

[BBR00] Jean-François Boulicaut, Artur Bykowski et Christophe Rigotti. – Approximation of frequency queries by mean of free-sets. In D. Zighed, J. Komorowski et J. M. Zytkow éds., *Proceedings of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'00).* – Lyon, France, Berlin : Springer-Verlag, septembre 2000. p. 75–85.

[BJ00] Jean-François Boulicaut et Baptiste Jeudy. – Using constraint for itemset mining : should we prune or not ? In A. Doucet éd., *Actes des 16e Journées Bases de Données Avancées (BDA'00).* – Blois, France, Université de Tours, octobre 2000. p. 221–237.

[BJ01a] Jean-François Boulicaut et Baptiste Jeudy. – Constraint-based discovery of a condensed representation for frequent patterns. In *Proceedings of the Workshop Database Support for KDD co-located with the Fifth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'01).* – Freiburg, Allemagne, septembre 2001. Disp. en ligne (sept. 2002) `http://www.informatik.uni-freiburg.de/~ml/ecmlpkdd/workshop-proceedings%.html`, 11 pages.

[BJ01b] Jean-François Boulicaut et Baptiste Jeudy. – Mining free-sets under constraints. In Michel E. Adiba, Christine Collet et Bipin C. Desai éds., *Proceedings of the International Database Engineering & Applications Symposium (IDEAS'01).* – Grenoble, France, IEEE Computer Society, juillet 2001. p. 322–329.

[BKM98] Jean-François Boulicaut, Mika Klemettinen et Heikki Mannila. – Querying inductive databases : A case study on the MINE RULE operator. In Jan M. Zytkow et Mohamed Quafafou éds., *Proceedings of the 2nd European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'98).* – Nantes, France, Berlin : Springer-Verlag, septembre 1998. p. 194–202.

[BKM99] Jean-François Boulicaut, Mika Klemettinen et Heikki Mannila. – Modeling KDD processes within the inductive database framework. In M. Mohania et A. Min Tjoa éds., *Proceedings of the 1st International Conference on Data Warehousing and Knowledge Discovery (DaWaK'99)*. – Florence, Italy, Berlin : Springer-Verlag, septembre 1999. p. 293–302.

[BMS97] Sergey Brin, Rejeev Motwani et Craig Silverstein. – Beyond market baskets : Generalizing association rules to correlations. In J. M. Peckman éd., *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'97)*. – Tucson, USA, New York : ACM Press, mai 1997. p. 265–276.

[BMUT97] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman et Shalom Tsur. – Dynamic itemset counting and implication rules for market basket data. In J. M. Peckman éd., *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'97)*. – Tucson, USA, New York : ACM Press, mai 1997. p. 255–264.

[BP99] Elena Baralis et Giuseppe Psaila. – Incremental refinement of mining queries. In Mukesh Mohania et A. Min Tjoa éds., *Proceedings of the 1st International Conference on Data Warehousing and Knowledge Discovery (DaWaK'99)*. – Florence, Italy, Berlin : Springer-Verlag, septembre 1999. p. 173–182.

[BPT+00] Yves Bastide, Nicolas Pasquier, Rafik Taouil, Gerd Stumme et Lotfi Lakhal. – Mining minimal non-redundant association rules using frequent closed itemsets. In J. Lloyd et al. éds., *Proceedings of the First conference on Computational Logic (CL'00)*. – London, UK, Berlin : Springer-Verlag, juillet 2000. p. 972–986.

[BR01] Artur Bykowski et Christophe Rigotti. – A condensed representation to find frequent patterns. In *Proceedings of the 19th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database and Knowledgebase Systems (PODS'01)*. – Santa Barbara, USA, New York : ACM Press, mai 2001. p. 267–273.

[BTP+00] Yves Bastide, Rafik Taouil, Nicolas Pasquier, Gerd Stumme et Lotfi Lakhal. – Mining frequent patterns with counting inference. *SIGKDD Explorations*, décembre 2000, vol. 2, n2, p. 66–75.

[BTP+02] Yves Bastide, Rafik Taouil, Nicolas Pasquier, Gerd Stumme et Lotfi Lakhal. – Pascal : un algorithme d'extraction des motifs fréquents. *Techniques et Science Informatiques*, 2002, vol. 21, n1, p. 65–95.

[Byk99] Artur Bykowski. – *Frequent Set Discovery in Highly-correlated Data*. – F-69621 Villeurbanne cedex, Mémoire du DEA d'informatique de Lyon, Institut National des Sciences Appliquées de Lyon, juillet 1999. 30 pages.

[CBM02] Matthieu Capelle, Jean-François Boulicaut et Cyrille Masson. – Extraction de motifs séquentiels sous contrainte de similarité. In Daniele Herin et Djamel A. Zighed éds., *Actes des Journées francophones d'Extraction et de Gestion des Connaissances (EGC'02)*. – Montpellier, France, Paris : Hermes, janvier 2002. p. 65–76.

[cc02a] cInQ consortium. – *D10a : Strategies for the evaluation of complex inductive queries. Deliverable Project cInQ IST-2000-26469*. – Bâtiment Blaise Pascal, F-69621 Villeurbanne, France, Coordination INSA Lyon - LISI, mai 2002. 59 pages. Restricted access, Rapport technique.

[cc02b] cInQ consortium. – *D1a : Theory for Inductive Databases. Deliverable Project cInQ IST-2000-26469*. – Bâtiment Blaise Pascal, F-69621 Villeurbanne, France, Coordination INSA Lyon - LISI, mai 2002. 108 pages. Public report, Rapport technique.

[cc02c] cInQ consortium. – *D2a : Pattern Domain ITEM (itemsets, episodes and rules). Deliverable Project cInQ IST-2000-26469.* – Bâtiment Blaise Pascal, F-69621 Villeurbanne, France, Coordination INSA Lyon - LISI, février 2002. 38 pages. Restricted access, Rapport technique.

[cc02d] cInQ consortium. – *D6a : Solvers for the ITEM Pattern Domain. Deliverable Project cInQ IST-2000-26469.* – Bâtiment Blaise Pascal, F-69621 Villeurbanne, France, Coordination INSA Lyon - LISI, mai 2002. 86 pages. Restricted access, Rapport technique.

[CG02] Toon Calders et Bart Goethals. – Mining all non-derivable frequent itemsets. In Tapio Elomaa, Heikki Mannila et Hannu Toivonen éds., *Proceedings of the 6th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'02).* – Helsinki, Finland, Berlin : Springer-Verlag, août 2002. p. 74–85.

[CHNW96] D. Cheung, J. Han, V. Ng et C.Y. Wong. – Maintenance of discovered association rules in large databases : An incremental updating technique. In S. Su éd., *Proceedings of the 12th International Conference on Data Engineering (ICDE'96).* – New Orleans, Louisiana, USA, IEEE Computer Society, février 1996. p. 106–114.

[DDR97] L. Dehaspe et L. De Raedt. – Mining association rules in multiple relations. In N. Lavrač et S. Džeroski éds., *Proceedings of the 7th International Workshop on Inductive Logic Programming.* – Prague, Czech Republic, Berlin : Springer-Verlag, septembre 1997. p. 125–132.

[DGLS01] Cheikh Talibouya Diop, Arnaud Giacometti, Dominique Laurent et Nicolas Spyratos. – Extraction incrémentale de règles d'association par combinaison de tâches d'extraction. In N. Mouaddib éd., *Actes des 17e Journées Bases de Données Avancées (BDA'01).* – Agadir, Maroc, Toulouse : Cepadues, octobre 2001. p. 285–297.

[dR00] Luc de Raedt. – A logical database mining query language. In J. Cussens et A. Frisch éds., *Proceedings of the 10th International Workshop on Inductive Logic Programming.* – London, UK, Berlin : Springer-Verlag, juillet 2000. p. 78–92.

[dR02a] Luc de Raedt. – *A Logical View of Inductive Databases.* – Georges-Kohler-Allee, Gebaude 079, D-79110 Freiburg, Germany, Institut fur Informatik, Albert-Ludwigs-Universitat, mai 2002. 13 pages., Rapport technique.

[dR02b] Luc de Raedt. – Query evaluation and optimization for inductive database using version spaces (extended abstract). In *Proceedings of the EDBT 2002 International Workshop on Database Technologies for Data Mining DTDM'02.* – Praha, CZ, mars 2002. p. 19–28, Disp. en ligne (sept. 2002) `http://www.cinq-project.org/edbt/`, 10 pages.

[dRK01] Luc de Raedt et Stefan Kramer. – The levelwise version space algorithm and its application to molecular fragment finding. In Bernhard Nebel éd., *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01).* – Seattle, USA, San Francisco : Morgan Kaufmann, août 2001. p. 853–862.

[DT99] Luc Dehaspe et Hannu Toivonen. – Discovery of frequent Datalog patterns. *Data Mining and Knowledge Discovery*, 1999, vol. 3, n1, p. 7–36.

[DT01] Luc Dehaspe et Hannu Toivonen. – Discovery of relational association rules. In Nada Lavrač et Sašo Džeroski éds., *Relational Data Mining*, p. 189–212. – Berlin : Springer-Verlag, 2001.

[DTK98] Luc Dehaspe, Hannu Toivonen et Ross D. King. – Finding frequent substructures in chemical compounds. In Rakesh Agrawal, Paul E. Stolorz et Gregory Piatetsky-Shapiro éds., *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD'98)*. – New York, USA, Menlo Park : AAAI Press, août 1998. p. 30–36.

[ec] Projet européen cInQ (2001-2004). – consortium on discovering knowledge with Inductive Queries, financé par la commission européenne, branche Future and Emerging Technologies du programme Information Society Technologies, contrat IST-2000-26469. – Site web : `http://www.cinq-project.org/`.

[Fay00] Abdourahmane Faye. – *Découverte d'associations entre tables d'une base de données : une approche par la logique du premier ordre*. – 15, rue G. Clémenceau 91405 Orsay cedex, France, Thèse de doctorat, Université Paris-Sud XI, LRI, septembre 2000.

[Fuk96] Takeshi Fukuda et al. – Data mining using two-dimensional optimized association rules : Scheme, algorithms, visualization. In H. V. Jagadish et Inderpal Singh Mumick éds., *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'96)*. – Quebec, Canada, New York : ACM Press, juin 1996. p. 13–23.

[GdB99] Bart Goethals et Jan Van den Bussche. – A priori versus a posteriori filtering of association rules. In K. Shim et R. Srikant éds., *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'99)*. – Philadelphia, USA, mai 1999. Disp. en ligne (sept. 2002) `http://www.almaden.ibm.com/cs/dmkd/`, 5 pages.

[GdB00] Bart Goethals et Jan Van den Bussche. – On supporting interactive association rule mining. In Y. Kambayashi, M. Mohania et A.M. Tjoa éds., *Proceedings of the 2nd International Conference on Data Warehousing and Knowledge Discovery (DaWaK'00)*. – London, UK, Berlin : Springer-Verlag, septembre 2000. p. 307–316.

[GHRS00] Minos M. Garofalakis, Dongjoon Hyun, Rajeev Rastogi et Kyuseok Shim. – Efficient algorithms for constructing decision trees with constraints. In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining (KDD'00)*. – Boston, USA, New York : ACM Press, août 2000. p. 335–339.

[GLD02] A. Giacometti, D. Laurent et C. Diop. – Condensed representations for sets of mining queries. In Mika Klemettinen et Rosa Meo éds., *Proceedings of the Knowledge Discovery in Inductive Database (KDID) International Workshop of PKDD'02*. – Helsinki, Finland, août 2002. Disp. en ligne (sept. 2002) `http://www.cinq-project.org/ecmlpkdd2002/schedule.html`, 15 pages.

[GLDS00] Arnaud Giacometti, Dominique Laurent, Cheikh Talibouya Diop et Nicolas Spyratos. – La découverte de règles d'association entre vues : vers un processus incrémental. In A. Doucet éd., *Actes des 16e Journées Bases de Données Avancées (BDA'00)*. – Blois, France, Université de Tours, octobre 2000. p. 323–344.

[GR00] Minos M. Garofalakis et Rajeev Rastogi. – Scalable data mining with model constraints. *SIGKDD Explorations*, décembre 2000, vol. 2, n2, p. 39–48.

[GRS99] Minos M. Garofalakis, Rajeev Rastogi et Kyuseok Shim. – SPIRIT : Sequential pattern mining with regular expression constraints. In Malcolm P. Atkinson et al. éds., *Proceedings of the 25nd International Conference on Very Large Data Bases (VLDB'99)*. – Edinburgh, UK, San Francisco : Morgan Kaufmann, septembre 1999. p. 223–234.

[HK00] Jiawei Han et Micheline Kamber. – *Data Mining : Concepts and techniques.* – San Francisco : Morgan Kaufmann, 2000. 533 pages.

[HP00] Jiawei Han et Jian Pei. – Mining frequent patterns by Pattern-Growth : methodology and implications. *SIGKDD Explorations*, décembre 2000, vol. 2, n2, p. 14–20.

[HPY00] Jiawei Han, Jian Pei et Yiwen Yin. – Mining frequent patterns without candidate generation. In W. Chen, J. Naughton et P. Bernstein éds., *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'00).* – Dallas, Texas, USA, New York : ACM Press, mai 2000. p. 1–12.

[IM96] T. Imielinski et H. Mannila. – A database perspective on knowledge discovery. *Communications of the ACM*, novembre 1996, vol. 39, n11, p. 58–64.

[IV99] Tomasz Imielinski et Aashu Virmani. – MSQL : A query language for database mining. *Data Mining and Knowledge Discovery*, décembre 1999, vol. 3, n4, p. 373–408.

[IWM00] Akihiro Inokuchi, Takashi Washio et Hiroshi Motoda. – An apriori-based algorithm for mining frequent substructures from graph data. In D. Zighed, J. Komorowski et J. M. Zytkow éds., *Proceedings of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'00).* – Lyon, France, Berlin : Springer-Verlag, septembre 2000. p. 13–23.

[JB02a] Baptiste Jeudy et Jean-François Boulicaut. – Constraint-based discovery and inductive queries : application to association rule mining. In D.J. Hand, N.M. Adams et R.J. Bolton éds., *Proceedings of the ESF Exploratory Workshop on Pattern Detection and Discovery in Data Mining.* – London, UK, Berlin : Springer-Verlag, septembre 2002. p. 110–124.

[JB02b] Baptiste Jeudy et Jean-François Boulicaut. – Optimization of association rule mining queries. *Intelligent Data Analysis*, 2002, vol. 6, n4, p. 341–358.

[JB02c] Baptiste Jeudy et Jean-François Boulicaut. – Using condensed representations for interactive association rule mining. In Tapio Elomaa, Heikki Mannila et Hannu Toivonen éds., *Proceedings of the 6th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'02).* – Helsinki, Finland, Berlin : Springer-Verlag, août 2002. p. 225–236.

[KdR01] Stefan Kramer et Luc de Raedt. – Feature construction with version spaces for biochemical applications. In *Proceedings of the 18th International Conference on Machine Learning (ICML'01).* – William College, USA, San Francisco : Morgan Kaufmann, juillet 2001. p. 258–265.

[KdRH01] Stefan Kramer, Luc de Raedt et Christoph Helma. – Molecular feature mining in HIV data. In *Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining (KDD'01).* – San Francisco, USA, New York : ACM Press, août 2001. p. 136–143.

[LK02] Dao-I Lin et Zvi M. Kedem. – Pincer-search : An efficient algorithm for discovering the maximum frequent set. *Knowledge and Data Engineering*, 2002, vol. 14, n3, p. 553–566.

[LNHP99] Laks V.S. Lakshmanan, Raymond Ng, Jiawei Han et Alex Pang. – Optimization of constrained frequent set queries with 2-variable constraints. In Alex Delis, Christos Faloutsos et Shahram Ghandeharizadeh éds., *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'99).* – Philadelphia, USA, New York : ACM Press, juin 1999. p. 157–168.

[LPL00] Stephane Lopes, Jean-Marc Petit et Lotfi Lakhal. – Discovering agree sets for database relation analysis. In A. Doucet éd., *Actes des 16e Journées Bases de Données Avancées (BDA'00).* – Blois, France, Université de Tours, octobre 2000. p. 181–200.

[LRK02] Sau Dan Lee, Luc De Raedt et Stefan Kramer. – *Constraint-Based Mining of First Order Sequences in SEQLOG.* – George-Kohler-Allee Geb. 079, D-79110 Freiburg, Germany, University of Freiburg, mai 2002. Rapport technique.

[Mel92] Chris Mellish. – The description identification problem. *Artificial Intelligence*, 1992, vol. 52, n2, p. 151–168.

[Meo00] R. Meo. – Theory of dependence values. *ACM Transactions on Database Systems*, septembre 2000, vol. 5, n3, p. 380–406.

[Mit80] Tom Mitchell. – Generalization as search. *Artificial Intelligence*, 1980, vol. 18, n2, p. 203–226.

[MPC96] Rosa Meo, Giuseppe Psaila et Stefano Ceri. – A new SQL-like operator for mining association rules. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan et Nandlal L. Sarda éds., *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96).* – Mumbay, India, San Francisco : Morgan Kaufmann, septembre 1996. p. 122–133.

[MPC97] Rosa Meo, Giuseppe Psaila et Stefano Ceri. – An extension to SQL for mining association rules. *Data Mining and Knowledge Discovery*, octobre 1997, vol. 9, n4, p. 275–300.

[MPT00] F. Masseglia, Pascal Poncelet et Maguelonne Tesseire. – Incremental mining of sequencial patterns in large databases. In A. Doucet éd., *Actes des 16e Journées Bases de Données Avancées (BDA'00).* – Blois, France, Université de Tours, octobre 2000. p. 345–366.

[MT96a] Heikki Mannila et Hannu Toivonen. – Discovering generalized episodes using minimal occurrences. In Evangelos Simoudis, Jiawei Han et Usama Fayyad éds., *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96).* – Portland, USA, Menlo Park : AAAI Press, août 1996. p. 146–151.

[MT96b] Heikki Mannila et Hannu Toivonen. – Multiple uses of frequent sets and condensed representations. In Evangelos Simoudis, Jiawei Han et Usama Fayyad éds., *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96).* – Portland, USA, Menlo Park : AAAI Press, août 1996. p. 189–194.

[MT97] Heikki Mannila et Hannu Toivonen. – Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1997, vol. 1, n 3, p. 241–258.

[MTV95] Heikki Mannila, Hannu Toivonen et A. Inkeri Verkamo. – Discovering frequent episodes in sequences. In Usama M. Fayyad et Ramasamy Uthurusamy éds., *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95).* – Montreal, Canada, Menlo Park : AAAI Press, août 1995. p. 210–215.

[MTV97] Heikki Mannila, Hannu Toivonen et A. Inkeri Verkamo. – Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1997, vol. 1, n3, p. 259–289.

[MWZ00] Tadeusz Morzy, Marek Wojciechowski et Maciej Zakrzewicz. – Materialized data mining views. In D. Zighed, J. Komorowski et J. M. Zytkow éds., *Proceedings of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'00).* – Lyon, France, Berlin : Springer-Verlag, septembre 2000. p. 65–74.

[NC00] Noël Novelli et Rosine Cicchetti. – Mining functional and embedded dependancies using free sets. In A. Doucet éd., *Actes des 16e Journées Bases de Données Avancées (BDA'00).* – Blois, France, Université de Tours, octobre 2000. p. 201–220.

[NDD99] Biswadeep Nag, Prasad M. Deshpande et David J. DeWitt. – Using a knowledge cache for interactive discovery of association rules. In Surajit Chaudhuri et David Madigan éds., *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining (KDD'99).* – San Diego, USA, New York : ACM Press, août 1999. p. 244–253.

[NLHP98] Raymond Ng, Laks V.S. Lakshmanan, Jiawei Han et Alex Pang. – Exploratory mining and pruning optimizations of constrained associations rules. In Laura M. Haas et Ashutosh Tiwary éds., *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'98).* – Seattle, Washington, USA, New York : ACM Press, juin 1998. p. 13–24.

[Pas00] Nicolas Pasquier. – *Data mining : algorithmes d'extraction de de réduction des règles d'association dans les bases de données.* – Complexe scientifique des Céseaux, F-63177 Aubière cedex, France, Thèse de doctorat, Université Clermont-Ferrand II, LIMOS, janvier 2000. 223 pages.

[PBTL99a] Nicolas Pasquier, Yves Bastide, Rafik Taouil et Lotfi Lakhal. – Closed set based discovery of small covers for association rules. In C. Collet éd., *Actes des 15e Journées Bases de Données Avancées (BDA'99).* – Bordeaux, France, octobre 1999. p. 361–381.

[PBTL99b] Nicolas Pasquier, Yves Bastide, Rafik Taouil et Lotfi Lakhal. – Discovering frequent closed item for association rules. In Catriel Beeri et Peter Buneman éds., *Proceedings of the 7th International Conference on Database Theory (ICDT'99).* – Jerusalem, Israel, Berlin : Springer-Verlag, janvier 1999. p. 398–416.

[PBTL99c] Nicolas Pasquier, Yves Bastide, Rafik Taouil et Lotfi Lakhal. – Efficient mining of association rules using closed itemset lattices. *Information Systems*, janvier 1999, vol. 24, n1, p. 25–46.

[PH00] Jian Pei et Jiawei Han. – Can we push more constraints into frequent pattern mining ? In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining (KDD'00).* – Boston, USA, New York : ACM Press, août 2000. p. 350–354.

[PHL01] Jian Pei, Jiawei Han et Laks V. S. Lakshmanan. – Mining frequent itemsets with convertible constraints. In *Proceedings of the 17th International Conference on Data Engineering (ICDE'01).* – Heidelberg, Germany, IEEE Computer Society, avril 2001. p. 433–442.

[PHM00] Jian Pei, Jiawei Han et Runying Mao. – CLOSET an efficient algorithm for mining frequent closed itemsets. In Dimitrios Gunopulos et Rajeev Rastogi éds., *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'00).* – Dallas, Texas, USA, mai 2000. Disp. en ligne (sept. 2002) `http://www.cs.ucr.edu/~dg/DMKD.html`, 10 pages.

[PHMA+01] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal et M.-C. Hsu. – PrefixSpan mining sequential patterns efficiently by prefix projected pattern growth. In *Proceedings of the 17th International Conference on Data Engineering (ICDE'01).* – Heidelberg, Germany, IEEE Computer Society, avril 2001. p. 215–226.

[PZOD99] Srinivasan Parthasarathy, Mohammed Javeed Zaki, Mitsunori Ogihara et Sandhya Dwarkadas. – Incremental and interactive sequence mining. In *Proceedings of the 8th ACM International Conference on Information and Knowledge Management (CIKM'99).* – Kansas City, Missouri, USA, New York : ACM Press, novembre 1999. p. 251–258.

[RC03] François Rioult et Bruno Crémilleux. – *Optimisation d'extraction de motifs : une nouvelle méthode fondée sur la transposition de donnnées.* – GREYC, CNRS UMR 6072, Université de Caen, mars 2003. 15 pages. Soumis, Rapport technique.

[SA96] Ramakrishnan Srikant et Rakesh Agrawal. – Mining sequential patterns : Generalizations and performance improvements. In Peter M. G. Apers, Mokrane Bouzeghoub et Georges Gardarin éds., *Advances in Database Technology—5th International Conference on Extending Database Technology (EDBT'96).* – Avignon, France, Berlin : Springer-Verlag, mars 1996. p. 3–17.

[SG92] Padhraic Smyth et Rodney M. Goodman. – An information theoretic approach to rule induction from databases. *IEEE Transactions on Knowledge and Data Engineering*, août 1992, vol. 4, n4, p. 301–316.

[SON98] Ashoka Savasere, Edward Omiecinski et Shamkant B. Navathe. – Mining for strong negative associations in a large database of customer transactions. In *Proceedings of the 14th International Conference on Data Engineering (ICDE'98).* – Orlando, Florida, USA, IEEE Computer Society, février 1998. p. 494–502.

[SVA97] Ramakrishnan Srikant, Quoc Vu et Rakesh Agrawal. – Mining association rules with item constraints. In David Heckerman, Heikki Mannila, Daryl Pregibon et Ramasamy Uthurusamy éds., *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (KDD'97).* – Newport Beach, California, USA, Menlo Park : AAAI Press, août 1997. p. 67–73.

[TBAR97] Shiby Thomas, Sreenath Bodagala, Khaled Alsabti et Sanjay Ranka. – An efficient algorithm for the incremental updation of association rules in large databases. In David Heckerman, Heikki Mannila, Daryl Pregibon et Ramasamy Uthurusamy éds., *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (KDD'97).* – Newport Beach, California, USA, Menlo Park : AAAI Press, août 1997. p. 263–266.

[TKR+95] Hannu Toivonen, Mika Klemettinen, Pirjo Ronkainen, Kimmo Hätönen et Heikki Mannila. – Pruning and grouping of discovered association rules. In Gholemreza Nakhaeizadeh Yves Kodratoff et Charles Taylor éds., *Workshop Notes of the ECML-95 Workshop on Statistics, Machine Learning, and Knowledge Discovery in Databases.* – Heraklion, Crete, Greece, MLnet, avril 1995. p. 47–52.

[Toi96] Hannu Toivonen. – Sampling large databases for association rules. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan et Nandlal L. Sarda éds., *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96).* – Mumbay, India, San Francisco : Morgan Kaufmann, septembre 1996. p. 134–145.

[TR97] Fabien Torre et Céline Rouveirol. – Natural ideal operators in inductive logic programming. In Maarten van Someren et Gerhard Widmer éds., *Proceedings of the 9th European Conference on Machine Learning (ECML'97)*. – Prague, Czech Republic, Berlin : Springer-Verlag, avril 1997. p. 274–289.

[WHH00] Ke Wang, Yu He et Jiawei Han. – Mining frequent itemsets using support constraints. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter et Kyu-Young Whang éds., *Proceedings of 26th International Conference on Very Large Data Bases (VLDB'00)*. – Cairo, Egypt, San Francisco : Morgan Kaufmann, septembre 2000. p. 43–52.

[Woj01] Marek Wojciechowski. – Interactive constraint-based sequencial pattern mining. In A. Caplinskas et J. Eder éds., *Proceedings of the 5th East-European Conference on Advances in Databases and Information Systems (ADBIS'01)*. – Vilnius, Lithuania, Berlin : Springer-Verlag, septembre 2001. p. 169–181.

[Zak00a] Mohammed Javeed Zaki. – Generating non-redundant association rules. In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining (KDD'00)*. – Boston, USA, New York : ACM Press, août 2000. p. 34–43.

[Zak00b] Mohammed Javeed Zaki. – Sequence mining in categorical domains : incorporating constraints. In *Proceedings of the 9th ACM International Conference on Information and Knowledge Management (CIKM'00)*. – Washington DC, USA, New York : ACM Press, novembre 2000. p. 422–429.

[Zak01] Mohammed Javeed Zaki. – SPADE : An efficient algorithm for mining frequent sequences. *Machine Learning*, 2001, vol. 42, n1/2, p. 31–60.

[ZDK00] Xiuzhen Zhang, Guozu Dong et Ramamohanarao Kotagiri. – Exploring constraints to efficiently mine emerging patterns from large high-dimensional datasets. In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining (KDD'00)*. – Boston, USA, New York : ACM Press, août 2000. p. 310–314.

[ZH02] Mohammed Javeed Zaki et Ching-Jui Hsiao. – CHARM : An efficient algorithm for closed itemset mining. In Robert Grossman, Jiawei Han, Vipin Kumar, Heikki Mannila et Rajeev Motwani éds., *2nd International SIAM Conference on Data Mining (SDM02)*. – Arlington, USA, avril 2002. Disp. en ligne (sept. 2002) `http://www.siam.org/meetings/sdm02/`, 17 pages.

[ZPOL97] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsuniri Ogihara et Wei Li. – New algorithms for fast discovery of association rules. In David Heckerman, Heikki Mannila, Daryl Pregibon et Ramasamy Uthurusamy éds., *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (KDD'97)*. – Newport Beach, California, USA, Menlo Park : AAAI Press, août 1997. p. 283–286.

Remerciements

Je tiens tout d'abord à remercier Lionel Brunie et Jean-François Boulicaut pour leurs conseils et leur disponibilité.

Je tiens à remercier Dominique Laurent et Michèle Sebag pour avoir accepté d'être rapporteurs de ma thèse et pour leurs nombreux commentaires, François Jacquenet et Stefan Kramer pour leur participation au jury.

Je tiens également à remercier très chaleureusement les membres actuels et passés de l'équipe data mining du LISI pour l'ambiance tant pendant le travail qu'en dehors : Anne-Cécile Agam, Céline Becquet, Jeremy Besson, Artur Bykowski, Matthieu Capelle, Bruno Crémilleux, Thomas Daurel, Marion Leleu, Claire Leschi, Cyrille Masson, Nicolas Meger, Céline Robardet et tout particuliérement les membres fondateurs et les animateurs principaux de cette équipe : Jean-François Boulicaut et Christophe Rigotti.

Finalement, je tiens à remercier ceux que j'ai pu croiser au LISI ou ailleurs et qui m'ont aussi aidé pendant cette thèse : Antoine, Alice, Karen, Solomon, Rami, Abraham ...