N° d'ordre: 2005 ISAL 0042 Année 2005

MÉMOIRE DE THÈSE

présentée devant

L'Institut National des Sciences Appliquées de Lyon

pour obtenir

Le Grade de Docteur

École Doctorale: Informatique et Information pour la Société
Formation Doctorale: Extraction de Connaissances dans les Données

par

Cyrille Masson

Contribution au cadre des bases de données inductives: formalisation et évaluation de scénarios d'extraction de connaissances

soutenue publiquement le 7 juillet 2005 devant le jury:

Mme. Anne Doucet	Rapporteur	Professeur, Université Pierre et Marie Curie, Paris VI
M. Dominique Laurent	Rapporteur	Professeur, Université de Cergy-Pontoise
M. Jean-François Boulicaut	Directeur de thèse	Maître de Conférences, HdR INSA Lyon
M. Olivier Gandrillon	Examinateur	Chargé de Recherches, HdR Université Claude Bernard Lyon 1
M. Arno Siebes	Examinateur	Professeur, Université d'Utrecht, Pays-Bas
M. Laurent Trilling	Président	Professeur, Université Joseph Fourier, Grenoble 1

2005

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
	CHIMIE DE LYON Responsable : M. Denis SINOU	Université Claude Bernard Lyon 1 Lab Synthèse Asymétrique UMR UCB/CNRS 5622 Bât 308 2ème étage 43 bd du 11 novembre 1918 69622 VILLEURBANNE Cedex Tél : 04.72.44.81.83 Fax : 04 78 89 89 14 sinou@univ-lyon1.fr
E2MC	**ECONOMIE, ESPACE ET MODELISATION DES COMPORTEMENTS** Responsable : M. Alain BONNAFOUS	M. Alain BONNAFOUS Université Lyon 2 14 avenue Berthelot MRASH M. Alain BONNAFOUS Laboratoire d'Economie des Transports 69363 LYON Cedex 07 Tél : 04.78.69.72.76 Alain.bonnafous∂ish-lyon.cnrs.fr
E.E.A.	**ELECTRONIQUE, ELECTROTECHNIQUE, AUTOMATIQUE** M. Daniel BARBIER	M. Daniel BARBIER INSA DE LYON Laboratoire Physique de la Matière Bâtiment Blaise Pascal 69621 VILLEURBANNE Cedex Tél : 04.72.43.64.43 Fax 04 72 43 60 82 Daniel.Barbier@insa-lyon.fr
E2M2	**EVOLUTION, ECOSYSTEME, MICROBIOLOGIE, MODELISATION** **http://biomserv.univ-lyon1.fr/E2M2** **M. Jean-Pierre FLANDROIS**	M. Jean-Pierre FLANDROIS UMR 5558 Biométrie et Biologie Evolutive Equipe Dynamique des Populations Bactériennes Faculté de Médecine Lyon-Sud Laboratoire de Bactériologie BP 1269600 OULLINS Tél : 04.78.86.31.50 Fax 04 72 43 13 88 E2m2∂biomserv.univ-lyon1.fr
EDIIS	**INFORMATIQUE ET INFORMATION POUR LA SOCIETE** http://www.insa-lyon.fr/ediis M. Lionel BRUNIE	M. Lionel BRUNIE INSA DE LYON EDIIS Bâtiment Blaise Pascal 69621 VILLEURBANNE Cedex Tél : 04.72.43.60.55 Fax 04 72 43 60 71 ediis@insa-lyon.fr
EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTE** http://www.ibcp.fr/ediss M. Alain Jean COZZONE	M. Alain Jean COZZONE IBCP (UCBL1) 7 passage du Vercors 69367 LYON Cedex 07 Tél : 04.72.72.26.75 Fax : 04 72 72 26 01 cozzone@ibcp.fr
	MATERIAUX DE LYON http://www.ec-lyon.fr/sites/edml M. Jacques JOSEPH	M. Jacques JOSEPH Ecole Centrale de Lyon Bât F7 Lab. Sciences et Techniques des Matériaux et des Surfaces 36 Avenue Guy de Collongue BP 163 69131 ECULLY Cedex Tél : 04.72.18.62.51 Fax 04 72 18 60 90 Jacques.Joseph@ec-lyon.fr
Math IF	**MATHEMATIQUES ET INFORMATIQUE FONDAMENTALE** http://www.ens-lyon.fr/MathIS M. Franck WAGNER	M. Franck WAGNER Université Claude Bernard Lyon1 Institut Girard Desargues UMR 5028 MATHEMATIQUES Bâtiment Doyen Jean Braconnier Bureau 101 Bis, 1er étage 69622 VILLEURBANNE Cedex Tél : 04.72.43.27.86 Fax : 04 72 43 16 87 wagner@desargues.univ-lyon1.fr
MEGA	**MECANIQUE, ENERGETIQUE, GENIE CIVIL, ACOUSTIQUE** http://www.lmfa.ec-lyon.fr/autres/MEGA/index.html M. François SIDOROFF	M. François SIDOROFF Ecole Centrale de Lyon Lab. Tribologie et Dynamique des Systèmes Bât G8 36 avenue Guy de Collongue BP 163 69131 ECULLY Cedex Tél :04.72.18.62.14 Fax : 04 72 18 65 37 Francois.Sidoroff@ec-lyon.fr

INSTITUT NATIONAL DES SCIENCES APPLIQUEES DE LYON

***Directeur* : STORCK A.**

***Professeurs* :**

AMGHAR Y.	LIRIS
AUDISIO S.	PHYSICOCHIMIE INDUSTRIELLE
BABOT D.	CONT. NON DESTR. PAR RAYONNEMENTS IONISANTS
BABOUX J.C.	GEMPPM***
BALLAND B.	PHYSIQUE DE LA MATIERE
BAPTISTE P.	PRODUCTIQUE ET INFORMATIQUE DES SYSTEMES MANUFACTURIERS
BARBIER D.	PHYSIQUE DE LA MATIERE
BASKURT A.	LIRIS
BASTIDE J.P.	LAEPSI****
BAYADA G.	MECANIQUE DES CONTACTS
BENADDA B.	LAEPSI****
BETEMPS M.	AUTOMATIQUE INDUSTRIELLE
BIENNIER F.	PRODUCTIQUE ET INFORMATIQUE DES SYSTEMES MANUFACTURIERS
BLANCHARD J.M.	LAEPSI****
BOISSE P.	LAMCOS
BOISSON C.	VIBRATIONS-ACOUSTIQUE
BOIVIN M. (Prof. émérite)	MECANIQUE DES SOLIDES
BOTTA H.	UNITE DE RECHERCHE EN GENIE CIVIL - Développement Urbain
BOTTA-ZIMMERMANN M. (Mme)	UNITE DE RECHERCHE EN GENIE CIVIL - Développement Urbain
BOULAYE G. (Prof. émérite)	INFORMATIQUE
BOYER J.C.	MECANIQUE DES SOLIDES
BRAU J.	CENTRE DE THERMIQUE DE LYON - Thermique du bâtiment
BREMOND G.	PHYSIQUE DE LA MATIERE
BRISSAUD M.	GENIE ELECTRIQUE ET FERROELECTRICITE
BRUNET M.	MECANIQUE DES SOLIDES
BRUNIE L.	INGENIERIE DES SYSTEMES D'INFORMATION
BUFFIERE J-Y.	GEMPPM***
BUREAU J.C.	CEGELY*
CAMPAGNE J-P.	PRISMA
CAVAILLE J.Y.	GEMPPM***
CHAMPAGNE J-Y.	LMFA
CHANTE J.P.	CEGELY*- Composants de puissance et applications
CHOCAT B.	UNITE DE RECHERCHE EN GENIE CIVIL - Hydrologie urbaine
COMBESCURE A.	MECANIQUE DES CONTACTS
COURBON	GEMPPM
COUSIN M.	UNITE DE RECHERCHE EN GENIE CIVIL - Structures
DAUMAS F. (Mme)	CENTRE DE THERMIQUE DE LYON - Energétique et Thermique
DJERAN-MAIGRE I.	UNITE DE RECHERCHE EN GENIE CIVIL
DOUTHEAU A.	CHIMIE ORGANIQUE
DUBUY-MASSARD N.	ESCHIL
DUFOUR R.	MECANIQUE DES STRUCTURES
DUPUY J.C.	PHYSIQUE DE LA MATIERE
EMPTOZ H.	RECONNAISSANCE DE FORMES ET VISION
ESNOUF C.	GEMPPM***
EYRAUD L. (Prof. émérite)	GENIE ELECTRIQUE ET FERROELECTRICITE
FANTOZZI G.	GEMPPM***
FAVREL J.	PRODUCTIQUE ET INFORMATIQUE DES SYSTEMES MANUFACTURIERS
FAYARD J.M.	BIOLOGIE FONCTIONNELLE, INSECTES ET INTERACTIONS
FAYET M. (Prof. émérite)	MECANIQUE DES SOLIDES
FAZEKAS A.	GEMPPM
FERRARIS-BESSO G.	MECANIQUE DES STRUCTURES
FLAMAND L.	MECANIQUE DES CONTACTS
FLEURY E.	CITI
FLORY A.	INGENIERIE DES SYSTEMES D'INFORMATIONS
FOUGERES R.	GEMPPM***
FOUQUET F.	GEMPPM***
FRECON L. (Prof. émérite)	REGROUPEMENT DES ENSEIGNANTS CHERCHEURS ISOLES
GERARD J.F.	INGENIERIE DES MATERIAUX POLYMERES
GERMAIN P.	LAEPSI****
GIMENEZ G.	CREATIS**
GOBIN P.F. (Prof. émérite)	GEMPPM***
GONNARD P.	GENIE ELECTRIQUE ET FERROELECTRICITE
GONTRAND M.	PHYSIQUE DE LA MATIERE
GOUTTE R. (Prof. émérite)	CREATIS**
GOUJON L.	GEMPPM***
GOURDON R.	LAEPSI****.

GRANGE G. (Prof. émérite)	GENIE ELECTRIQUE ET FERROELECTRICITE
GUENIN G.	GEMPPM***
GUICHARDANT M.	BIOCHIMIE ET PHARMACOLOGIE
GUILLOT G.	PHYSIQUE DE LA MATIERE
GUINET A.	PRODUCTIQUE ET INFORMATIQUE DES SYSTEMES MANUFACTURIERS
GUYADER J.L.	VIBRATIONS-ACOUSTIQUE
GUYOMAR D.	GENIE ELECTRIQUE ET FERROELECTRICITE
HEIBIG A.	MATHEMATIQUE APPLIQUEES DE LYON
JACQUET-RICHARDET G.	MECANIQUE DES STRUCTURES
JAYET Y.	GEMPPM***
JOLION J.M.	RECONNAISSANCE DE FORMES ET VISION

Novembre 2003

JULLIEN J.F.	UNITE DE RECHERCHE EN GENIE CIVIL - Structures
JUTARD A. (Prof. émérite)	AUTOMATIQUE INDUSTRIELLE
KASTNER R.	UNITE DE RECHERCHE EN GENIE CIVIL - Géotechnique
KOULOUMDJIAN J. (Prof. émérite)	INGENIERIE DES SYSTEMES D'INFORMATION
LAGARDE M.	BIOCHIMIE ET PHARMACOLOGIE
LALANNE M. (Prof. émérite)	MECANIQUE DES STRUCTURES
LALLEMAND A.	CENTRE DE THERMIQUE DE LYON - Energétique et thermique
LALLEMAND M. (Mme)	CENTRE DE THERMIQUE DE LYON - Energétique et thermique
LAREAL P (Prof. émérite)	UNITE DE RECHERCHE EN GENIE CIVIL - Géotechnique
LAUGIER A. (Prof. émérite)	PHYSIQUE DE LA MATIERE
LAUGIER C.	BIOCHIMIE ET PHARMACOLOGIE
LAURINI R.	INFORMATIQUE EN IMAGE ET SYSTEMES D'INFORMATION
LEJEUNE P.	UNITE MICROBIOLOGIE ET GENETIQUE
LUBRECHT A.	MECANIQUE DES CONTACTS
MASSARD N.	INTERACTION COLLABORATIVE TELEFORMATION TELEACTIVITE
MAZILLE H. (Prof. émérite)	PHYSICOCHIMIE INDUSTRIELLE
MERLE P.	GEMPPM***
MERLIN J.	GEMPPM***
MIGNOTTE A. (Mle)	INGENIERIE, INFORMATIQUE INDUSTRIELLE
MILLET J.P.	PHYSICOCHIMIE INDUSTRIELLE
MIRAMOND M.	UNITE DE RECHERCHE EN GENIE CIVIL - Hydrologie urbaine
MOREL R. (Prof. émérite)	MECANIQUE DES FLUIDES ET D'ACOUSTIQUES
MOSZKOWICZ P.	LAEPSI****
NARDON P. (Prof. émérite)	BIOLOGIE FONCTIONNELLE, INSECTES ET INTERACTIONS
NAVARRO Alain (Prof. émérite)	LAEPSI****
NELIAS D.	LAMCOS
NIEL E.	AUTOMATIQUE INDUSTRIELLE
NORMAND B.	GEMPPM
NORTIER P.	DREP
ODET C.	CREATIS**
OTTERBEIN M. (Prof. émérite)	LAEPSI****
PARIZET E.	VIBRATIONS-ACOUSTIQUE
PASCAULT J.P.	INGENIERIE DES MATERIAUX POLYMERES
PAVIC G.	VIBRATIONS-ACOUSTIQUE
PECORARO S.	GEMPPM
PELLETIER J.M.	GEMPPM***
PERA J.	UNITE DE RECHERCHE EN GENIE CIVIL - Matériaux
PERRIAT P.	GEMPPM***
PERRIN J.	INTERACTION COLLABORATIVE TELEFORMATION TELEACTIVITE
PINARD P. (Prof. émérite)	PHYSIQUE DE LA MATIERE
PINON J.M.	INGENIERIE DES SYSTEMES D'INFORMATION
PONCET A.	PHYSIQUE DE LA MATIERE
POUSIN J.	MODELISATION MATHEMATIQUE ET CALCUL SCIENTIFIQUE
PREVOT P.	INTERACTION COLLABORATIVE TELEFORMATION TELEACTIVITE
PROST R.	CREATIS**
RAYNAUD M.	CENTRE DE THERMIQUE DE LYON - Transferts Interfaces et Matériaux
REDARCE H.	AUTOMATIQUE INDUSTRIELLE
RETIF J-M.	CEGELY*
REYNOUARD J.M.	UNITE DE RECHERCHE EN GENIE CIVIL - Structures
RICHARD C.	LGEF
RIGAL J.F.	MECANIQUE DES SOLIDES
RIEUTORD E. (Prof. émérite)	MECANIQUE DES FLUIDES
ROBERT-BAUDOUY J. (Mme) (Prof. émérite)	GENETIQUE MOLECULAIRE DES MICROORGANISMES
ROUBY D.	GEMPPM***
ROUX J.J.	CENTRE DE THERMIQUE DE LYON – Thermique de l'Habitat
RUBEL P.	INGENIERIE DES SYSTEMES D'INFORMATION
SACADURA J.F.	CENTRE DE THERMIQUE DE LYON - Transferts Interfaces et Matériaux
SAUTEREAU H.	INGENIERIE DES MATERIAUX POLYMERES
SCAVARDA S. (Prof. émérite)	AUTOMATIQUE INDUSTRIELLE

SOUIFI A.	PHYSIQUE DE LA MATIERE
SOUROUILLE J.L.	INGENIERIE INFORMATIQUE INDUSTRIELLE
THOMASSET D.	AUTOMATIQUE INDUSTRIELLE
THUDEROZ C.	ESCHIL – Equipe Sciences Humaines de l'Insa de Lyon
UBEDA S.	CENTRE D'INNOV. EN TELECOM ET INTEGRATION DE SERVICES
VELEX P.	MECANIQUE DES CONTACTS
VERMANDE P. (Prof émérite)	LAEPSI
VIGIER G.	GEMPPM***
VINCENT A.	GEMPPM***
VRAY D.	CREATIS**
VUILLERMOZ P.L. (Prof. émérite)	PHYSIQUE DE LA MATIERE

***Directeurs de recherche C.N.R.S.* :**

BERTHIER Y.	MECANIQUE DES CONTACTS
CONDEMINE G.	UNITE MICROBIOLOGIE ET GENETIQUE
COTTE-PATAT N. (Mme)	UNITE MICROBIOLOGIE ET GENETIQUE
ESCUDIE D. (Mme)	CENTRE DE THERMIQUE DE LYON
FRANCIOSI P.	GEMPPM***
MANDRAND M.A. (Mme)	UNITE MICROBIOLOGIE ET GENETIQUE
POUSIN G.	BIOLOGIE ET PHARMACOLOGIE
ROCHE A.	INGENIERIE DES MATERIAUX POLYMERES
SEGUELA A.	GEMPPM***
VERGNE P.	LaMcos

***Directeurs de recherche I.N.R.A.* :**

FEBVAY G.	BIOLOGIE FONCTIONNELLE, INSECTES ET INTERACTIONS
GRENIER S.	BIOLOGIE FONCTIONNELLE, INSECTES ET INTERACTIONS
RAHBE Y.	BIOLOGIE FONCTIONNELLE, INSECTES ET INTERACTIONS

***Directeurs de recherche I.N.S.E.R.M.* :**

KOBAYASHI T.	PLM
PRIGENT A.F. (Mme)	BIOLOGIE ET PHARMACOLOGIE
MAGNIN I. (Mme)	CREATIS**

* ***CEGELY***	*CENTRE DE GENIE ELECTRIQUE DE LYON*
** ***CREATIS***	*CENTRE DE RECHERCHE ET D'APPLICATIONS EN TRAITEMENT DE L'IMAGE ET DU SIGNAL*
*** ***GEMPPM***	*GROUPE D'ETUDE METALLURGIE PHYSIQUE ET PHYSIQUE DES MATERIAUX*
**** ***LAEPSI***	*LABORATOIRE D'ANALYSE ENVIRONNEMENTALE DES PROCEDES ET SYSTEMES INDUSTRIELS*

Table des matières

Remerciements

Je tiens à remercier ici, toutes les personnes qui, par leur participation directe ou indirecte, ont permis l'aboutissement de cette thèse. Tout d'abord, je tiens à remercier Jean-François Boulicaut pour avoir été mon directeur de thèse pendant ces années, ainsi que pour ses conseils tout au long de cette période.

Je souhaite également remercier Dominique Laurent et Anne Doucet pour avoir accepté la charge d'être rapporteurs de ce mémoire de thèse, ainsi que pour leurs remarques constructives pour l'amélioration de ce rapport. Je remercie aussi Olivier Gandrillon, Laurent Trilling et Arno Siebes d'avoir accepté d'être examinateurs.

Je tiens également à exprimer ma reconnaissance aux membres de l'équipe Data Mining et Bases de Données Inductives du LIRIS : Christophe Rigotti, Céline Robardet et Claire Leschi pour les permanents, ainsi que Nicolas, Artur, Baptiste, Marion, Thomas, Hunor, Jérémy, Iéva et Ruggero pour les doctorants, actuels ou anciens.

Enfin, je tiens à remercier ma famille, mes amis, et l'équipe du pôle info de l'IUT B pour m'avoir soutenu pendant toute la durée de ma thèse.

Résumé

Le succès des techniques de bases de données permet de collecter une quantité toujours plus grande d'informations dans différents domaines. L'ECD (Extraction de Connaissance dans les Données) se donne pour but d'aller plus loin dans le processus d'interrogation des données pour y découvrir, sous forme de motifs, de la connaissance cachée. La notion de base de données inductive (BDI) généralise le concept de base de données en intégrant données et motifs dans un cadre commun. Un processus d'ECD peut alors être vu comme un processus d'interrogation étendu sur une BDI.

Cette thèse s'intéresse à la formalisation et à l'évaluation des scénarios d'extraction dans le cadre des BDI. Nous montrons d'abord comment utiliser un langage abstrait pour les BDI pour décrire de manière formelle des processus d'extraction réalisables par l'utilisateur. Nous obtenons ainsi un scénario prototypique, i.e. un objet théorique composé d'une séquence de requêtes inductives, sur lequel il est possible de raisonner. Un tel scénario sert avant tout à formaliser des traitements pour le transfert d'expertise entre utilisateurs et spécialistes en ECD. Une autre application du concept de scénario est l'évaluation sur une base commune de différentes implémentations de BDI, dans la lignée des benchmarks existants pour les bases de données. Un scénario d'évaluation a le même aspect qu'un scénario prototypique, mais on s'intéresse ici aux problèmes algorithmiques et d'optimisation de séquences de requêtes inductives. Lors du calcul du plan d'exécution d'un tel scénario, le système devra analyser les propriétés des requêtes qui le composent, en découvrant des dépendances entre celles-ci ou des conjonctions de contraintes pour lesquelles nous souhaitons disposer d'outils d'extraction efficaces. Enfin, nous présentons un scénario d'évaluation en bioinformatique et nous montrons comment le résoudre en utilisant des techniques préexistantes dans l'équipe ou développées pour les besoins de ce scénario.

Abstract

The success of database technologies has lead to an always increasing mass of collected information in different application fields. Knowledge Discovery in Databases (KDD) aims at going further in the querying processes on such data so as to find in these data some hidden knowledge materialized under the form of patterns. The Inductive Database (IDB) concept is a generalization of the database concept which integrates patterns and data in a common framework. A KDD process can thus be seen as an extended querying process on an IDB.

This PhD. thesis is about the formalization and the evaluation of KDD scenarios in the IDB framework. We first show how to use an abstract language for IDBs to formally describe extraction processes that can be performed by the user. We thus obtain a prototypical scenario, i.e. a theoritical object made of a sequence of inductive queries and on which it is possible to reason. Such a kind of scenario is useful to formalize processes when transfering expertise between final users and KDD experts. Another application of the concept of scenario is the evaluation on a common basis of different implementations of IDBs, similarly to existing benchmarks for databases. An evaluation scenario has the same form than a prototypical scenario, but it focuses more on algorithmic issues and optimization techniques for sequences of inductive queries. When computing an execution plan for such a scenario, the IDB system should analyze the properties of queries composing it, by discovering dependencies between them or conjunctions of constraints for which it is useful to have efficient extraction tools. Finally, we present an evaluation scenario in the field of bioinformatics, and we show how to solve it by using techniques developed in our group or especially designed for the need of this scenario.

Chapitre 1

Introduction

1.1 Historique : des bases de données vers l'ECD

L'un des principaux moteurs du développement de l'informatique a été le besoin de stocker et de gérer des masses d'informations dont la taille n'a cessé de croître. Dès les années 60 apparait le notion de base de données, et la simultanéité du développement des réseaux informatiques va rapidement permettre un partage de ces informations entre les utilisateurs. L'architecture typique sera alors un serveur (mainframe) stockant l'information et auquel sont reliés différent terminaux "passifs" permettant de consulter l'information. Les années 70 verront le développement du modèle relationnel pour le stockage des collections de données, qui s'imposera rapidement comme un modèle de référence. S'en suivra une forte activité chez les éditeurs de logiciels pour proposer des Systèmes de Gestion de Bases de Données Relationnel (SGBDR) implémentant ces principes. L'existence de techniques de modélisation efficaces contribua largement à la large adoption du modèle relationnel par les utilisateurs dans les années qui suivirent. Puis dans les années 80, de nouveaux modèles furent proposés, et des SGBD propres à des besoins spécifiques firent leur apparition (Systèmes d'Information Géographiques notamment). Par exemple, les Bases de données Objet [BD97] reposent sur des concepts provenant des méthodes de programmation éponymes qui se sont développées à la même époque. On peut ainsi, entre autres, associer des méthodes aux objets stockés dans la base de données.

L'une des raisons de l'évolution des modèles et des techniques de stockages des données est l'explosion de la quantité d'information à stocker. Dans de nombreux domaines, les besoins ont rapidement évolués : de l' "informatisation" de l'existant, on est passé à la nécessité d'automatiser l'acquisition de données. Cette évolution, liée au progrès technologiques, a été particulièrement sensible dans certains domaines. En sciences expérimentales, par exemple, les mesures sont aujourd'hui réalisées par des capteurs capables d'enregistrer la valeur de plusieurs centaines de paramètres à des intervalles de temps très courts. Ceci permet, entre autres, de construire des modèles mathématiques de plus en plus précis pour les phénomènes étudiés. Dans le domaine de la grande distribution également, on dispose aujourd'hui de systèmes qui enregistrent en direct tous les produits achetés par les clients lors de leur passage en caisse. Ces données peuvent ensuite être traitées pour essayer de cibler des offres sur des catégories de clients (marketing ciblé). Enfin, dans le domaine médical, l'historique des différents examens et diagnostic des patients, est lui aussi numérisé, conduisant là encore à une explosion de la quantité d'information stockée.

Cette explosion du volume de données a rapidement conduit les utilisateurs vers de nouveaux

besoins. En sciences expérimentales, on s'est rapidement interrogé sur la possibilité d'obtenir des connaissances à partir de la masse de données collectées. De telles connaissances peuvent se matérialiser sous des formes très différentes : par exemple, cela peut consister en la découverte de régularités parmi certains enregistrements ou de dépendances entre certaines valeurs. Elles peuvent également permettre de faciliter la description des données. Mais l'information extraite peut aussi avoir un rôle explicatif des données. L'utilisation d'outils de classification supervisée comme les arbres de décision peut ainsi permettre de prédire la valeur de certains attributs grâce à des modèles compéhensibles par l'utilisateur. De même, les techniques de classification non supervisée, comme le "clustering", permettent de regrouper des enregistrements similaires, afin de mieux comprendre les données.

Deux grandes familles de techniques sont apparues autour des années 90 pour étendre les SGBD classiques avec des outils d'analyse de données :

- OLAP (On-Line Analytical Processing) : on désigne par ce nom une catégorie d'outils permettant de faire l'analyse "en ligne" d'information stockée dans des bases ou entrepôts de données. Cette technologie est surtout orientée vers l'analyse de données multidimensionnelles. Une de ses caractéristiques est qu'elle permet de manipuler des hiérarchies sur les dimensions. Il est ainsi possible de suivre les ventes de familles de produits par magasin et par période de l'année avec des niveaux de granularités différents (par exemple, ventes par types de produits par mois et par secteur géographique, ou bien ventes par types de produits par trimestre et par magasin).
- ECD (Extraction de Connaissances dans les Données) : on désigne par ce terme l'ensemble des techniques permettant d'extraire de l'information préalablement inconnue dans les données. L'idée est d'appliquer des algorithmes de découvertes de motifs (clusters, arbres de décision, ensembles d'items, motifs séquentiels, etc) sur les données contenues dans la base de données pour en extraire de la connaissance. L'ECD s'inscrit dans la continuité d'OLAP dans la mesure ou elle répond à de nouveaux besoins en terme d'interrogation et de visualisation, en particulier en matière de découverte automatique de motifs. Une définition plus précise de l'ECD est donnée dans la section suivante.

D'une manière générale, on constate que les besoins en matière d'interrogation de données évoluent de plus en plus vers l'explication et la caractérisation des données.

1.2 L'Extraction de Connaissances à partir de Données

1.2.1 Définition

L'ECD est devenu un domaine de recherche à part entière à partir de 1989, quand Gregory Piatetsky-Shapiro a organisé la première réunion de chercheurs sur l'extraction automatique de connaissances dans les grandes bases de données. L'Extraction de Connaissances à partir de Données (ECD ou KDD pour "Knowledge Discovery in Databases") a été définie par Piatetsky-Shapiro et Frawley comme le "processus non trivial d'extraction d'informations potentiellement utiles, implicites, et inconnues auparavant à partir d'un ensemble de données" [PSF91]. Le projet Quest d'IBM[1], lancé en 1993, a également été l'un des moteurs du développement de cette discipline, en développant un nombre important d'algorithmes pour la fouille de données. C'est un domaine multi-disciplinaire et au croisement de nombreuses thématiques relevant des

[1] `http ://www.almaden.ibm.com/cs/quest/`

mathématiques et de l'informatique, on peut citer entre autres :

- les bases de données. En effet, le but de l'ECD était initialement d'extraire de la connaissance dans des bases de données existantes. Cela a un double impact : d'une part, sur les algorithmes, qui doivent prendre en compte certaines caractéristiques propres aux bases de données (accès aux données, standardisation, volumes à traiter, etc). D'autre part, une adaptation du modèle des bases de données est souhaitable afin de mieux prendre en compte certaines propriétés des algorithmes d'ECD. Par exemple, dans le cadre du projet KESO (Knowledge Extraction for Statistical Offices), basé sur le système de gestion de base de données Monet [2], l'extraction de règles d'association s'est révélée plus efficace que dans des SGBD classiques, grâce notamment à une représentation des données plus adaptée aux algorithmes et à l'existence de primitives pour les manipulations ensemblistes (intersection, union) [HKMT95, SK97].
- l'apprentissage automatique ("Machine Learning") : en matière d'extraction de modèles de données, de nombreux travaux ont déjà été réalisés depuis le début des années 80 [Mit82, Mit97]. Même si les algorithmes proposés ne passent pas forcément à l'échelle des grandes bases de données et ne sont pas nécessairement tous orientés sur des critères d'optimisation du temps d'exécution, cette discipline a inspiré certaines techniques à l'ECD.
- l'interface homme-machine : une fois la connaissance extraite, il faut pouvoir la visualiser, ce qui n'est pas forcément simple avec des modèles dont la sémantique n'est pas immédiatement compréhensible par l'utilisateur final. Il faut néanmoins ajouter que ces problèmes de visualisation ne concerne pas que les motifs mais aussi les données originales. Ainsi, dans de nombreux cas pratiques, on découvre de la connaissance simplement en représentant et en regardant les données initiales sous un certain point de vue.
- les statistiques : le domaine des statistiques exploratoires s'intéresse depuis longtemps à la construction de modèles prédictifs et a également fourni de nombreuses méthodes à l'ECD, comme l'analyse en composantes principales ou la classification hiérarchique. Enfin, certains algorithmes utilisés en ECD font appel à de nombreuses mesures d'intérêt issues du monde des statistiques. Le post-traitement des motifs extraits et leur visualisation utilisent aussi de telles mesures.

Avant d'aller plus loin et de détailler plus précisément le processus de l'ECD, nous allons d'abord présenter de manière informelle quelques notions qui seront détaillées plus loin, mais qui sont nécessaires afin de comprendre la terminologie utilisée.

Il faut tout d'abord souligner que l'ECD repose avant tout sur l'existence d'algorithmes de fouilles de données. De tels algorithmes travaillent en général sur des données qui doivent avoir un format bien particulier généralement adapté au type de connaissance que l'on cherche à extraire. On appellera contexte d'extraction une telle représentation des données. Après extraction, l'algorithme renvoit un ou plusiers motifs (ou modèles) construits à partir du jeu de données. Un motif est une représentation de tout ou partie du contexte d'extraction initial. On distinguera deux types de motifs : les motifs globaux et les motifs locaux [Han02, DJHS01]. Les motifs globaux ont pour but de modéliser le jeu de données dans son ensemble. Parmi les motifs globaux, on trouve par exemple les arbres de décision [Qui86] ou les réseaux de neurones [Hay98]. Les motifs locaux ont pour but de décrire des propriétés locales des données, par exemple des corrélations entre quelques enregistrements. En recherchant des motifs locaux, on chercher à décrire le jeu de données par un ensemble de caractéristiques. Parmi les motifs locaux, on va

[2] `http ://db.cwi.nl/projecten/project.php4 ?prjnr=77`

trouver les ensembles d'items, les règles d'association et les motifs séquentiels.

L'approche choisie par le projet cInQ, et qui sera détaillée dans la Section 2, suggère de regrouper les motifs par domaines, pour lesquels on peut définir un langage de motifs sur un type de données particulier. Parmi ces différents domaines de motifs, on peut notamment citer le domaine ITEM, qui rassemble les motifs constitués d'items. Plus précisément, on considère un alphabet de symboles (items) Σ, on peut alors envisager différents motifs construits sur cet alphabet, dont, entre autres :

- les ensembles d'items : il s'agit de sous-ensembles de Σ. On dira qu'un ensemble d'items s apparaît dans une transaction t si et seulement si $s \subseteq t$. Théoriquement, il y a donc $2^{|\Sigma|}$ ensembles d'items possibles. Les applications dans lesquelles on a besoin d'obtenir ce type de motifs sont nombreuses. En biologie, par exemple, on peut imaginer qu'une table contient des niveaux d'expression de gènes (sur-expression ou non) et que chaque transaction de cette table correspond à une situation biologique donnée. On peut alors désirer rechercher des ensembles de gènes qui sont surexprimés simultanément dans un nombre de transactions supérieur à un seuil minimal.
- les motifs séquentiels : il s'agit d'une suite ordonnée d'items. Ainsi, si $\Sigma = \{a, b, c\}$, un motif séquentiel possible est $b \rightarrow a \rightarrow a$ (le symbole flèche $\rightarrow$ dénote la succession). Extraire ce genre de motif n'a de sens que si l'on travaille sur des données ordonnées, comme les bases de séquences, où chaque séquence correspondant à un ensemble ordonnée de transactions. Lorsqu'on analyse le parcours des utilisateurs sur un site web, chaque séquence en entrée peut correspondre à une succession de pages Web visités par un utilisateur. On peut alors souhaiter rechercher des parcours fréquents (i.e. apparaissant un nombre suffisant de fois) dans la base de séquences.

Les algorithmes d'extraction travaillant sur ces types de motifs doivent donc explorer un espace de recherche de taille importante voire infinie. Afin de faciliter le parcours de cet espace, il est pertinent de ne s'intéresser qu'aux motifs vérifiant certaines contraintes spécifiées par l'utilisateur, comme la fréquence minimale, où l'on ne cherche que des motifs apparaissant un nombre suffisant de fois dans le jeu de données. Une approche commune pour l'exploration de l'espace de recherche est l'approche par niveaux proposé par Agrawal et Srikant dans [AS94]. D'une manière générale, elle consiste à générer des ensembles de motifs candidats dont on teste ensuite s'ils vérifient les contraintes. Plus précisément, il s'agit d'un processus itératif en trois principales étapes :

1. La génération des motifs candidats de taille k à partir des motifs de taille inférieure $k-1$ vérifiant les contraintes. Lors de la première itération, les motifs de taille 1 sont directement générés à partir de l'alphabet Σ.

2. L'élagage a priori de l'ensemble des candidats de taille k en prenant en compte les propriétés (par exemple l'anti-monotonie) des contraintes.

3. La vérification de la validité des contraintes pour les candidats restants afin d'éliminer les candidats non intéressants.

Ce processus est répété jusqu'à ce qu'il ne soit plus possible de générer de nouveaux motifs candidats. Les contraintes peuvent également présenter des propriétés qui permettent de simplifier l'espace de recherche à parcourir grâce à des procédures de génération et d'élagage adaptées.

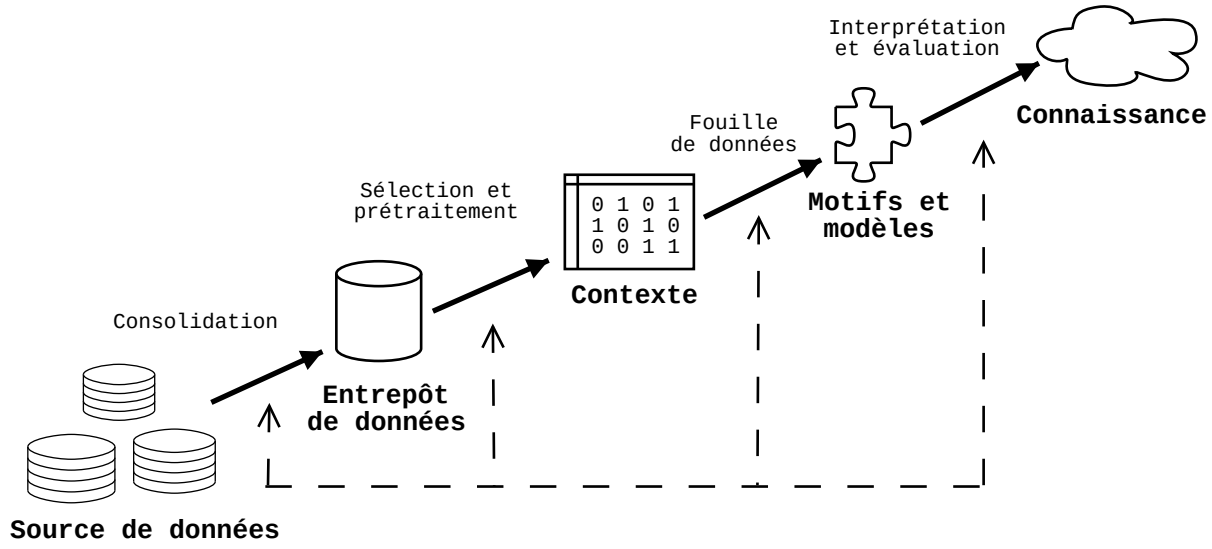

FIG. 1.1 – Le processus d'Extraction de Connaissances à partir de Données

1.2.2 Le processus d'ECD

Un processus d'Extraction de Connaissances dans les Données est un ensemble d'opérations qui englobe l'extraction de motifs ou de modèles mais concerne aussi les étapes de pré-traitement et de post-traitement. Il possède certaines caractéristiques importantes. Tout d'abord, il traite de grands volumes de données. Dans certains domaines d'application, il est d'usage d'avoir à traiter des relations possédant plusieurs millions d'enregistrements. Ensuite, un processus d'ECD est intéractif et itératif : l'information extraite provient des données disponibles et l'utilisateur doit pouvoir affiner ses recherches au fur et à mesure qu'il avance dans le processus. Il est également important d'extraire de la connaissance inconnue auparavant. Les connaissances du type "Si une personne est enceinte, alors c'est une femme" être écartées des résultats finaux, le mieux étant qu'elles ne soient pas calculées du tout. L'idée de la fouille de données est de tirer partie de la grande quantité de données pour extraire de l'information implicite pertinente et de réduire au maximum la quantité de trivialités extraites. Enfin, la connaissance extraite doit être exprimée sous une forme compréhensible par l'utilisateur.

D'une manière plus concrète, un processus d'extraction de connaissances dans les bases de données peut se décomposer en quatre grandes étapes (cf. Figure 1.1) :

- La consolidation : cette première étape permet de rassembler et d'unifier dans un cadre conceptuel commun les différentes données sur le domaine que l'on souhaite étudier. Cette étape inclue également certains traitements nécessaires au nettoyage des données.A la fin de cette étape, on obtient généralement un entrepôt de données, qui servira de référentiel de source de données pour la suite du processus.
- Sélection et prétraitement : cette étape va nous permettre de sélectionner parmi toutes les données du domaine celles qui concernent plus particulièrement le problème que l'on souhaite étudier. Dans le cadre de l'analyse de données d'expression de gènes, cela correspondra à la récupération des données concernant une population de cellules déterminée. Une fois ce travail réalisé, il peut être nécessaire de leur appliquer un prétraitement afin

que, par exemple, leur format corresponde à celui nécessaire pour la phase suivante de fouille de données. Selon la méthode, il pourra s'agir de discrétiser les valeurs d'attributs continus dans le cas d'apprentissage d'arbres de décision par ID3 [Qui86], ou alors de coder des propriétés booléennes afin d'y extraire des règles d'association. C'est aussi à cette étape qu'il faudra traiter le cas des valeurs manquantes ou aberrantes.

– La fouille de données (ou Data Mining) : c'est l'étape qui constitue véritablement le cœur du processus d'Extraction de Connaissances dans les données. C'est ici que l'on va chercher à extraire automatiquement des motifs ou des modèles à partir des données. Cette étape est la plus critique du point de vue algorithmique. En effet, l'espace de recherche est en général très vaste, et c'est ici qu'il faudra utiliser, lorsque cela est possible, les contraintes spécifiées par l'utilisateur lors de l'extraction. Cela permet de réduire l'espace de recherche à parcourir et donc de faire en sorte que l'algorithme s'exécute dans des temps raisonnables. De plus, un autre aspect important à prendre en compte est la manière dont l'algorithme va lire les données en entrée. Si le jeu de données est volumineux, faire plusieurs passes sur les données peut être coûteux en temps.
– L'interprétation et l'évaluation : à cette étape, il faut regarder si les modèles et motifs retournés par la phase de fouille de données correspondent à une connaissance nouvelle pertinente pour le domaine considéré. Dans le cas de l'extraction d'ensembles d'items ou de motifs séquentiels, il est courant de récupérer plusieurs milliers de motifs dont seuls quelques uns se révèleront véritablement intéressants, ce qui implique une phase d'analyse assez longue. Dans le cas des clusters ou des arbres de décisions, les modèles renvoyés étant de taille plus réduite, leur analyse peut en sembler simplifiée, mais leur interprétation et leur validation restent très difficiles.

1.2.3 L'ECD et les bases de données

Dans la pratique, un grand nombre d'algorithmes d'extraction de connaissance ont la forme de programmes exécutables travaillant sur des fichiers texte au lieu de véritables bases de données. Un couplage plus fort avec les technologies des bases de données est donc nécessaire afin de prendre en compte les contraintes propres à celles-ci mais également de tirer partie des progrès obtenus dans ce domaine. Ainsi, au cours des dernières années, les communautés scientifiques en bases de données et en fouille de données ont témoigné d'un intérêt réciproque grandissant. Ces intérêts de recherche communs entre les deux communautés ont conduit à un certain nombre de propositions pour l'intégration des techniques de "Data Mining" avec les bases de données. C'est ainsi que fut proposé en 1996 le cadre conceptuel des Bases de Données Inductives par T. Imielinski et H. Mannila [IM96]. Les bases de données inductives proposent d'intégrer dans un cadre commun des données brutes et la connaissance extraite à partir de ces données (cf. Figure 1.2). Elles sont différentes des bases de données classiques et des bases de données déductives. Les bases de données déductives utilisent la déduction pour augmenter une base de faits afin de contenir un ensemble potentiellement infini de faits dérivés ou déduits. Les bases de données inductives, quant à elles, intègrent des faits avec des motifs (i.e. des généralisations ou des régulatités qui caractérisent les données) dans un cadre commun afin de supporter les processus d'ECD dans le cadre des bases de données. Une définition plus formelle du cadre des bases de données inductives sera donnée dans la suite de ce mémoire.

D'une manière assez informelle, une base de données inductive est constituée d'une partie données, qui peut contenir des données brutes (en extension) ou des vues sur ces données

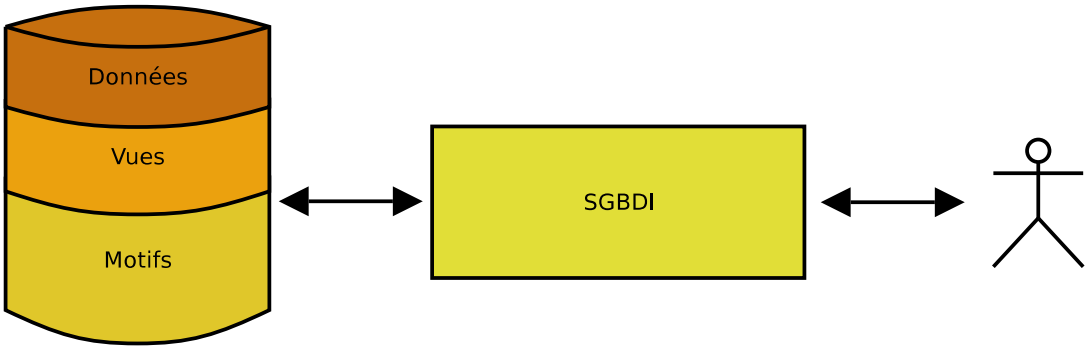

FIG. 1.2 – Le concept de Base de Données inductive

(représentation en intention), et d'une partie motifs. Ce sera ensuite le rôle du Système de Gestion de Bases de Données Inductives (SGBDI) que d'être l'interface entre cette représentation physique des données et motifs et l'utilisateur. Ce sera lui qui mettra en place les plans d'exécution des requêtes inductives complexes entrées par l'utilisateur et qui les exécutera. Outre les caractéristiques classiques héritées des SGBD traditionnels (comme la gestion des opérations concurrentes ou les droits d'accès), il devra également gérer les problèmes d'optimisation des requêtes inductives qui lui sont soumises, comme la mise en cache de certains motifs pour accélérer certains traitements. Cette notion d'intégration des motifs et des données sous-jacente au concept de BDI soulève plusieurs problèmes, comme la définition d'un langage de requête permettant de travailler aussi bien sur les données que sur les motifs, la spécification des phases d'extraction et de post-traitement, ou encore les modèles choisis pour interfacer les algorithmes d'extraction au SGBD.

1.2.4 Vers l'intégration de l'ECD avec les bases de données

Si l'on souhaite rapprocher les techniques de l'Extraction de Connaissance dans les Données avec celles des bases de données traditionnelles, un travail de fond doit être fait sur les stratégies d'intégration à envisager. En effet, actuellement, le fait de réaliser l'extraction de connaissance en dehors du cadre des bases de données est encore répandu. Les données sont ainsi enregistrées dans un espace de travail séparé du SGBD et où l'algorithme est exécuté pour extraire les motifs. Ce problème d'intégration a donné lieu à plusieurs travaux [AS96, STA98, Cha98] et nous présenterons ici les différentes solutions envisageables pour rapprocher les outils de fouille de données et les SGBD existants. Ainsi, on peut notamment citer :

– Cache and mine : c'est l'approche la plus communément utilisée pour l'instant. Le principe consiste à extraire d'abord les données de la base via une interface appropriée comme ODBC afin de les stocker dans un buffer sous forme de fichier plat ou en mémoire vive, puis à appliquer l'algorithme d'extraction de motifs sur l'ensemble extrait, indépendamment de l'espace mémoire du SGBD. Cette technique présente l'avantage d'optimiser les coûts d'Entrée/Sortie (un seul accès a la base de données réelle) et de permettre de mettre les données dans un format efficace et pertinent pour l'algorithme utilisé. Le buffer est vidé une fois l'exécution de l'algorithme terminée. Malgré l'espace supplémentaire nécessaire requis et le temps passé lors de la recopie de données du SGBD vers un autre espace

mémoire du système pour les besoins de l'algorithme, cette méthode est très efficace en pratique.
- Procédures stockées : dans cette approche, l'algorithme d'extraction est implémenté comme un procédure du SGBD et s'éxécute donc dans le même espace mémoire que celui-ci. Les résultats sont stockés en retour dans le SGBD. L'inconvénient principal est que l'accès aux données se faisant à travers les interfaces classiques du SGBD, les phases de comptage du support des algorithmes d'extraction peuvent être assez longues comparées à la méthode "Cache and Mine", où l'on dispose souvent de structures et de procédures particulièrement optimisées pour cette tâche.
- Approche SQL : elle consiste à rééecrire l'ensemble des algorithmes d'extraction avec SQL ou l'une de ses extensions [Ran04]. On peut ainsi avoir recours à des fonctions définies par l'utilisateur (User-Defined Functions, UDF).Le temps de développement peut aussi être plus court, car on n'a pas à s'occuper des problèmes de gestion de la mémoire et de l'espace disque, ou bien des techniques d'indexation. Cependant, certains algorithmes sont très difficiles à écrire en SQL, comme l'algorithme APriori réécrit en SQL par Rantzau dans [Ran03].

[STA98] a proposé une étude comparative de ces différentes approches sur le cas de l'algorithme APriori [AS94]. En terme d'efficacité calculatoire, il a ainsi été établi que l'approche Cache and Mine était la meilleure, suivie par l'approche SQL, et enfin les procédures stockées. La différence entre l'approche Cache and Mine et l'usage de procédures stockées est directement reliée au nombre de passes sur la base de données, le temps d'exécution de cette dernière étant approximativement celle de Cache Mine multipliée par le nombre de passes sur les données en entrée.

1.3 Contributions de la thèse

Ce travail de thèse s'intéresse au cadre des bases de données inductives sous l'angle de l'extraction de motifs locaux comme les ensembles d'items (itemsets), les règles d'association, les épisodes, les motifs séquentiels, etc. Il s'est déroulé dans le cadre du projet européen cInQ (IST-FET 2000-26469) qui avait pour objectif principal le développement des aspects théoriques et pratiques des bases de données inductives et qui s'est déroulé du 01/05/2001 au 30/04/2004. Les participants à ce projet étaient l'INSA de Lyon (Coordinateur, France), l'Università degli Studi di Torino (Italie), la Politecnico di Milano (Italie), l'Albert-Ludwigs Universität Freiburg (Allemagne), le Nokia Research Center d'Helsinki (Finlande) et le Jozef Stefan Institute (Slovénie). Plus précisément, le travail contenu dans ce mémoire apporte deux types de contributions au domaine des BDI : des contributions algorithmiques et des contributions méthodologiques.

1.3.1 Contributions algorithmiques

Comme nous l'avons déjà évoqué, le concept des bases de données inductives repose en grande partie sur l'existence d'un nombre important d'algorithmes d'extraction de différent types de motifs, aussi bien locaux (description de propriétés locales des données, comme la recherche d'ensemble d'items fréquents) que globaux (description globale des données par un modèle, comme les arbres de décision, ou une partition du jeu de données). Pour l'extraction de certains motifs, l'utilisateur peut souhaiter exprimer la connaissance a priori sur l'intérêt

des motifs qu'il cherche à extraire à l'aide de contraintes (par exemple, des contraintes syntaxiques, ou de taille sur les motifs à extraire). Il pourra alors être utile de disposer de plusieurs types d'algorithmes pour explorer l'espace de recherche et exploiter au mieux les propriétés de ces différentes contraintes. L'étude des propriétés de ces contraintes par rapport à la méthode de parcours de l'espace de recherche utilisée par l'algorithme peut permettre d'apporter des améliorations sensibles aux performances de ce dernier (réduction du temps d'exécution, ou du nombre d'opérations d'entrée/sortie réalisées). Dans le cas des motifs relevant du domaine ITEM (ensembles d'items, motifs séquentiels, épisodes, etc), et en présence de contraintes anti-monotones, la procédure de découverte peut par exemple suivre une approche par niveaux. L'utilisation de contraintes lors de l'extraction peut avoir un impact sur deux points clés de cette procédure itérative : la génération et l'élagage de l'ensemble des candidats. On parle alors d'exploitation active des contraintes. Typiquement, une propriété utile des contraintes sera l'anti-monotonie, qui caractérise les contraintes dont la validité sur un motif donné entraine la validité sur tous les sous-motifs du motif considéré. Cependant, certaines contraintes, notamment d'ordre syntaxique, ne présentent pas toujours des propriétés aussi utiles. Il faut alors développer des stratégies plus fines pour parvenir à en faire une exploitation active dans un algorithme par niveaux, c'est à ce sujet que nous nous intéresserons ici.

Nous proposerons en effet deux algorithmes réalisant des extractions sous contraintes. Le premier d'entre eux, nommé GALIBOT [CBM02, CMB02], s'intéresse à la fouille de motifs séquentiels, c'est-à-dire des suites ordonnées d'items dans des bases de séquences. On souhaite prendre en compte deux contraintes, l'une portant sur la fréquence minimale des motifs extraits dans les données sources (une contrainte anti-monotone classique dans l'extraction de motifs), et l'autre portant sur l'aspect syntaxique des motifs séquentiels à extraire. En effet, on ne souhaite trouver que des motifs séquentiels fréquents similaires à un motif de référence donné par l'utilisateur. Pour cela, on définira bien sûr une mesure de distance entre deux motifs séquentiels, inspirée des mesures classiques comme la distance d'édition entre deux chaînes. On pourra alors dériver facilement une contrainte de similarité minimale. Malheureusement, comme celle-ci n'est pas anti-monotone, il faudra mettre au point une stratégie particulière, en relaxant cette contrainte, pour pouvoir en faire une exploitation active dans un algorithme d'extraction par niveaux. Cela implique aussi un léger post-traitement de l'ensemble obtenu à la fin du processus d'itération afin de filtrer les motifs vérifiant la contrainte relaxée mais pas la contrainte originale. Enfin, comme on travaille ici sur une conjonction de contraintes, il sera intéressant d'étudier la sélectivité de chacune d'entre elle en fonction des paramètres initiaux.

L'autre algorithme que nous proposons s'intéresse à l'extraction de motifs séquentiels composés d'une suite d'atomes issus de la logique du premier ordre [MJ02a, MJ02b]. Il a pour but d'extraire des motifs qui sont non seulement fréquents mais dont la séquence des prédicats sur lesquels les atomes sont construits constitue une chaîne reconnue par une certaine expression rationnelle $\mathcal{R}$ spécifiée par l'utilisateur. Dans le cadre de l'extraction de motifs séquentiels d'items, il existe déjà une famille d'algorithmes, nommée SPIRIT [GRS02], permettant d'extraire des motifs fréquents et vérifiant une contrainte syntaxique $\mathcal{C}_{ER}$ basée sur une expression rationnelle. En général $\mathcal{C}_{ER}$ n'est ni monotone, ni anti-monotone, il faut utiliser une stratégie de relaxation afin de dériver des contraintes, certes moins fortes, mais présentant des propriétés qui pourront être exploitées dans les fonctions de génération et d'élagage des candidats. Voilà pourquoi Minos N. Garofalakis et al. ont proposé quatre relaxations possibles de la contrainte $\mathcal{C}_{ER}$ afin d'en faire une exploitation active durant l'extraction. Ces différentes relaxations conduisent lors des validations expérimentales à différents compromis entre les élaguages dus à la contrainte

de fréquence et ceux dus à la contrainte d'expression rationnelle $\mathcal{C}_{ER}$. Dans ce mémoire, nous proposons d'étendre SPIRIT à l'extraction de motifs séquentiels dont les éléments sont exprimés sous forme relationnelle. Si l'on suppose que l'on dispose d'un jeu de données correspondant aux différentes suite de commandes Unix entrées par différents utilisateurs lors de leurs session, on pourra alors extraire des motifs du type : $\langle mv(F, D)\ cd(D)\ emacs(F)\ gcc(F, E)$, où les noms des commandes apparaissent sous forme de symboles de prédicat et où les paramètres peuvent être généralisés à l'aide de variables. L'utilité de concepts issus de la Programmation Logique Inductive [MR94] est maintenant démontrée dans le domaine de la Fouille de Données Multi-Relationnelles : par exemple, WARMR [DT99] étend la découverte de règles d'association à la découverte de requêtes fréquentes et TILDE [BR98] étend la construction d'arbres de décision à des arbres logiques de décision, dont les nœuds sont des atomes de la logique du premier ordre (comme $dessus(A, B)$). Il nous semble donc pertinent d'essayer de s'attaquer au problème de l'extraction de séquences composées d'atomes, faisant intervenir des variables permettant la généralisation de certaines constantes. Cette idée nous semble d'autant plus pertinente que, dans certains cas pratiques, il est plus naturel de décrire une situation en utilisant des littéraux, par exemple lorsqu'on s'intéresse à des événements caractérisés par certains paramètres (comme ceux passés à un exécutable sur une ligne de commande UNIX). Dans ce mémoire, nous montrerons donc comment étendre les algorithmes de Garofalakis en une nouvelle famille d'algorithmes, nommée SPIRIT-LºG dédiée à l'extraction de séquences logiques.

Enfin, une dernière contribution algorithmique de ce mémoire concerne le post-traitement des motifs extraits et plus particulièrement les requêtes ensemblistes sur les ensembles d'items. Si on considère la caractéristique des BDI qui est l'intégration dans un cadre commun des données et des motifs, il est souvent assez raisonnable de stocker les ensembles d'items obtenus lors d'une extraction dans une table relationnelle. Cependant, de nombreuses requêtes de post-traitement font appel à des tests d'inclusion entre ensembles d'items (par exemple, la calcul d'une couverture d'un ensemble d'items). Ce besoin s'est fait clairement ressentir lorsque nous avons réalisé des extractions d'ensembles d'items sur des données d'expression de gènes et constitue d'ailleurs l'un des problèmes abordés dans le projet Bingo (Bases de Données Inductives pour la Génomique) dans le cadre de l'ACI "masse de données". De tels tests ensemblistes sont rarement proposés par les différentes implémentations de SQL. Dans la Section 5, nous proposons une nouvelle technique pour stocker des ensembles d'items dans des bases de données relationnelles. Celle-ci est basée sur l'utilisation de clés bitmap hachées et d'arbres d'ensembles d'items, ce qui permet d'encoder partiellement dans la structure de données les relations d'inclusion qui peuvent exister entre différents ensembles d'items. Cela permet d'accélérer sensiblement un grand nombre de requêtes ensemblistes sur les ensembles d'items.

1.3.2 Contributions méthodologiques

Le développement des bases de données inductives passe également par la mise au point d'un langage de requête adapté. En effet, celui-ci doit permettre de manipuler aussi bien les données brutes que les motifs, et être suffisamment expressif pour pouvoir écrire aussi bien des requêtes de pré-traitement que d'extraction ou de post-traitement. Mais avant de procéder à la recherche de nouvelles primitives, il faut examiner les différents propositions de langages de requêtes pour la fouille de données. En effet, il existe déjà quelques implémentations de systèmes qui intègrent des algorithmes d'extraction avec les bases de données et qui peuvent être vus comme les prémisses de futurs systèmes de gestion de bases de données inductives. Ces systèmes fournissent souvent

un langage (généralement une extension de SQL) pour manipuler les données et les motifs sur lesquels ils travaillent. Même s'ils sont souvent limités à un type de motif, il est intéressant de les étudier afin de mettre en évidence quelques propriétés fondamentales que doit vérifier un langage idéal pour les bases de données inductives. C'est l'un des principaux buts de l'étude réalisée dans [BBMM04], reprise et détaillée dans la Section 2 de ce mémoire.

De plus, si l'on souhaite à terme développer des sytèmes de bases de données inductives, il deviendra rapidement indispensable de disposer d'un moyen de décrire formellement les ensembles de manipulations qu'un utilisateur peut réaliser. Une telle formalisation pourra ainsi non seulement être utile pour la conception d'un langage de requêtes pour les BDI, mais aussi pour l'élaboration de véritables scénarios d'évaluation ou benchmarks pour ces systèmes. De tels benchmarks pourront alors prendre la forme d'un jeu de données, auquel aura été associé un problème d'ECD décrit de manière formelle sous forme d'une séquence de requêtes éventuellement inductives. De tels benchmarks permettront alors de faire une évaluation objective des bases de données inductives sur deux critères. En effet, outre la classique évaluation quantitative déjà massivement utilisée (temps CPU, Entrées/Sorties, etc), il sera possible de faire également une évaluation qualitative de différentes implémentations de BDI. Il pourra ainsi être intéressant de comparer les algorithmes mis en jeu pour résoudre un problème donné. Par exemple, on pourra étudier les stratégies de prise en compte des contraintes lors de l'appel d'algorithmes d'extraction sous contraintes (utilisation active pour la génération et l'élagage des candidats ou post-traitement).

C'est pourquoi, dans cette thèse, nous avons choisi une approche originale de cet aspect des bases de données inductives, en étudiant la conception, la formalisation et les usages de scénarios d'ECD. Par scénario, nous entendons la formalisation sous la forme d'une série de requêtes de l'ensemble des opérations constituant un processus d'ECD. Cette formalisation se fera dans le cadre théorique des bases de données inductives proposé par le consortium cInQ. On peut déjà distinguer deux types de scénarios :

- Les scénarios prototypiques : le but est de retranscrire de manière formelle un ensemble de manipulations réalisées ou potentiellement réalisables par l'utilisateur. Une fois cette formalisation effectuée, il sera plus simple de procéder à du transfert d'expertise entre les utilisateurs finaux et les spécialistes en ECD. Par exemple, on peut décrire un scénario prototypique d'extraction de modules de transcription candidats à partir de données d'expression biopuces.
- Les scénarios d'évaluation : ils ont pour but de permettre l'évaluation objective de systèmes de bases de données inductives, aussi bien sur le plan quantitatif (résultats en temps, en mémoire, etc) que qualitatif (pertinence des stratégies mises en œuvre, qualité des résultats obtenus, etc). En particulier, il est possible, grâce à de de tels scénarios, d'étudier les performances et le comportement de différents systèmes de bases de données inductives face à certaines difficultés algorithmiques qu'ils contiennent.

Cette approche scénario constitue le filigramme de ce travail de thèse. En effet, la formalisation des processus d'ECD dans le cadre des bases de données inductives développé dans le projet cInQ permet l'écriture des requêtes à l'aide d'un langage à base de contraintes. Il devient alors très simple de mettre en évidence des conjonctions ou disjonctions de contraintes pertinentes de par leurs propriétés et le type de motifs sur lequel elles portent. Cela permet d'identifier rapidement les besoins en terme de développement de nouveaux algorithmes dédiés à ce genre d'extraction. La conception de prototypes comme Galibot et SPIRIT-LºG s'inscrit pleinement dans cette logique. En outre, l'étude de la pertinence de notre approche scénario sur un exemple

tiré du domaine de la bioinformatique nous permet également de souligner d'autres besoins en termes de post-traitement des résultats, et notamment en matière de technique d'indexation pour l'optimisation des requêtes ensemblistes sur les motifs ensembles d'items, pour lesquelles nous proposerons une nouvelle solution originale [MRB04].

Après avoir formalisé le cadre des bases de données inductives, présenté et étoffé un langage générique pour la description des étapes d'un processus d'ECD, nous présenterons différents scénarios prototypiques inspirés de la réalité dans plusieurs domaines applicatifs. Ces exemples feront appel à plusieurs phases d'extraction de motifs de diverses natures, sous des contraintes variées. Après cela, nous développerons la notion de scénario d'évaluation et nous proposerons un premier scénario de ce type dans le cadre de la bioinformatique. Nous montrerons deux exécutions possibles de ce scénario en utilisant un ensemble de prototypes développés au LIRIS (équipe "Data Mining" et Bases de Données Inductives).

1.4 Organisation du mémoire

Le Chapitre 2 est consacré à un état de l'art du travail déjà publié dans le cadre des bases de données inductives [BBMM04]. Nous y verrons comment ce concept est apparu et nous étudierons les premières propositions faites en terme de langage de requêtes pour la fouille de données, et plus particulièrement l'extraction d'ensembles d'items. Cela nous permettra de mettre en évidence les caractéristiques indispensables d'un futur langage pour les Bases de Données Inductives (BDI). Après cet état de l'art, nous présenterons les motivations qui ont conduit à la création du projet cInQ et la méthode proposée par ce dernier pour développer la théorie des bases de données inductives. Nous obtiendrons ainsi un véritable agenda de recherche qui constitue le fil directeur de la suite du mémoire. Dans le Chapitre 3, nous présenterons ensuite plus formellement le cadre théorique des bases de données inductives proposé par le consortium cInQ. Nous proposerons alors un langage de requêtes à base de contraintes pour la spécification des différentes étapes d'un processus d'ECD. Cela nous permettra d'introduire plus précisément le concept de scénario d'ECD, et de donner des premiers exemples de scénarios prototypiques dans plusieurs domaines. Ces derniers permettent, entre autre, de mettre en évidence des conjonctions de contraintes particulières devant être utilisées lors de l'extraction. Dans le Chapitre 4, nous présenterons deux algorithmes dédiés aux motifs contenant de l'information de nature séquentielle (Galibot et SPIRIT-LºG). Enfin, dans le chapitre 5, nous montrerons l'apport de notre approche scénario d'évaluation dans le domaine de la bioinformatique. Pour cela, nous proposerons un premier scénario d'évaluation sur un jeu de données portant sur l'analyse de données d'expression de gènes. Nous décrirons un ensemble de traitements qui soulève différentes difficultés algorithmiques. En particulier, l'une d'elle concernera la réalisation d'un grand nombre de requêtes ensemblistes (recherche de sur-ensembles) sur les des motifs de types ensembles d'items. En effet, si on situe dans le cadre des bases de données inductives, où les motifs sont stockés au même titre que les données, cela pose des problèmes de représentation et d'indexation des motifs. Nous proposons donc une technique pour optimiser ce stockage dans le but de réaliser des requêtes ensemblistes. Enfin, le chapitre 6 nous permettra de conclure ce mémoire et d'identifier de futurs thèmes de recherche pour continuer à développer le cadre des bases de données inductives et les scénarios d'évaluation.

Chapitre 2

État de l'art

Le cadre des bases de données inductives (BDI) veut répondre au besoin d'une meilleure intégration entre les outils d'extraction de connaissances dans les données (ECD) et les systèmes de gestion de bases de données (SGBD). Dans cette partie, nous présenterons l'introduction du concept de base de données inductive par Tomasz Imielinski et Heikki Mannila [IM96]. Comme l'un des problèmes cruciaux associé à ce concept est la définition d'un langage qui y soit associé, nous présenterons aussi quelques propositions de langages faites afin d'améliorer l'intégration d'extracteurs de motifs dans des SGBD. La confrontation de ceux-ci sur un exemple inspiré de la pratique réelle nous permettra de mettre en évidence les apports et les lacunes des solutions existantes. Cela débouchera sur l'exposé des motivations du projet européen cInQ (IST-FET 2000-26469). Nous présenterons alors notre méthode de travail dans ce consortium pour développer le cadre des bases de données inductives et nous proposerons un agenda de recherche dans ce but, ce qui nous permettra de situer les différentes contributions de ce mémoire par rapport à ce plan de développement.

2.1 La notion de base de données inductive

2.1.1 Motivations et intuitions

La notion de base de donnée inductive a été proposée dans [IM96]. Les auteurs partent du postulat que le développement des outils d'extraction automatique de la connaissance pose de nouveaux défis à la technologie des bases de données, à cause notamment du caractère *ad hoc* de la connaissance extraite. Ils constatent également que le succès de Stuctured Query Language (SQL) est essentiellement dû au fait qu'il repose sur un faible nombre de primitives permettant de réaliser un très grand nombre de requêtes. Il est important de noter, que la plupart des systèmes d'ECD s'intéressent à un nombre restreint de type de motifs et ne sont pas intégrés dans des applications plus conséquentes. D'un certain point de vue, cette situation est très similaire à celle des systèmes de gestion de données au début des années 60, où chaque application devait être écrite à partir de rien sans avoir recours aux primitives de SQL et d'autres API, largement utilisées aujourd'hui. En fait, beaucoup d'applications d'ECD actuelles réalisent plutôt de la fouille de fichiers que de la véritable fouille de bases de données, étant donné que le couplage avec la base de données est faible. D'une certaine manière, l'interface entre les extracteurs et la base de données se résume à deux commandes simples : "lire depuis" (la base) et "écrire dans" (la base).

En outre, il est important de signaler que l'ECD ne peut se résumer à de l'apprentissage automatique qui serait simplement capable de travailler sur de grandes quantités de données. Il est également inexact de penser que les questions soulevées par l'aspect "base de données" des algorithmes d'ECD consistent seulement en des gains de performance comme l'optimisation des coûts d'Entrée/Sortie. Là encore, si on regarde comment est née l'algèbre relationnelle, on s'aperçoit que ce n'est pas simplement l'amélioration des performances de stockage qui a permis l'essor des SGBD, mais c'est bien l'apparition de langages de requêtes, de modèles et techniques d'optimisation et de gestion des transactions qui ont été la source de l'intense développement des bases de données au cours des trois dernières décennies. Plus précisément, c'est la nature *ad hoc* des requêtes qui a suscité le besoin d'avoir des optimiseurs de requêtes génériques. Du point de vue d'Imielinski et Mannila, le même chemin de développement doit être suivi en ECD afin d'intégrer les extracteurs et les SGBD dans un cadre commun.

On peut alors envisager deux pistes de recherche :

- l'une à court terme visant à améliorer l'interfaçage entre les algorithmes d'extraction de connaissance et les actuels systèmes de gestion de bases de données relationnelles. Cela peut être fait, par exemple, en développant de nouvelles techniques d'indexation ou de nouveaux aggrégats adaptés aux algorithmes d'ECD.
- et une autre, à plus long terme visant au développement de véritables systèmes de gestion de données et de découverte de connaissance, qui intègreraient dans un cadre de travail commun les données et les motifs extraits par les algorithmes d'ECD : les bases de données inductives (BDI). Celles-ci reposeront - entre autres - sur de véritables compilateurs de requêtes inductives et sur un langage de requêtes dédié.

2.1.2 Définitions

De manière informelle, une base de donnée inductive est un extension d'une base de données classique, qui permet de stocker et de manipuler des motifs obtenus par l'application de divers algorithmes d'ECD sur les données brutes qu'elle contient. Elle est donc constituée d'une partie "données" et d'une partie "motifs". Un utilisateur peut interroger une BDI de la même manière qu'une base de données classique. Cependant, il est aussi possible d'interroger la partie "motifs" de la base, en réalisant par exemple des opérations d'extraction ou des manipulations sur les motifs déjà stockés, ou encore en testant certains modèles préalablement extraits sur un ensemble de données. Ainsi, dans l'idéal, il sera possible de réaliser l'intégralité des étapes d'un processus d'extraction de connaissances à l'intérieur du cadre BDI : sélection des données à traiter et construction des contextes d'extraction, utilisation d'algorithmes de fouille, stockage et post-traitement des motifs obtenus.

Le schéma d'une base de données inductive est une paire $\mathcal{R} = (\mathbf{R}, (\mathcal{Q}_\mathbf{R}, e, \mathcal{V}))$, où $\mathbf{R}$ est un schéma de base de données classique, $\mathcal{Q}_\mathbf{R}$ est une collection de motifs, $\mathcal{V}$ est un ensemble de valeurs résultat, et e est un ensemble de fonctions d'évaluation, qui définissent la sémantique des motifs. Une fonction d'évaluation fait correspondre un élément de $\mathcal{V}$ à chaque paire $(\mathbf{r}, \theta_i)$, où $\mathbf{r}$ est une base de données sur $\mathbf{R}$ et θ_i est un motif de $\mathcal{Q}_\mathbf{R}$. Remarquons que la notation $\mathcal{Q}_\mathbf{R}$ met en valeur le fait que l'ensemble des motifs est dépendant du schéma de la partie "données". Une base de données inductive $\hat{\mathbf{r}} = (\mathbf{r}, s)$ est une instance du schéma $\mathcal{R}$ et consiste en une base de données $\mathbf{r}$ de schéma $\mathbf{R}$ et d'un sous-ensemble $s \subseteq \mathcal{Q}_\mathbf{R}$. Dans ce cadre, les requêtes qui retournent des données, celles qui retournent des motifs (requêtes d'extraction et de post-traitement) et celles qui croisent les données et les motifs sont toutes des requêtes inductives.

2.1.3 Vers un langage de requêtes inductif

Il est important de souligner que, dans un contexte de base de données inductive, les requêtes doivent être beaucoup plus générales qu'en SQL. De la même manière, les objets sur lesquels elles portent peuvent être beaucoup plus complexes que des tuples d'une base relationnelle. Pour cela, il est nécessaire de définir précisement ce que sont les objets en ECD et les requêtes inductives, et de voir comment le principe de fermeture d'un langage peut être transposé dans une BDI.

Par le terme d'objet d'ECD, nous désignons un classifieur, une règle, ou un cluster. Les classifieurs sont obtenus en utilisant par exemple des arbres de décision ou des réseaux de neurones. Les clusters sont des ensembles d'objets relativement similaires selon une métrique donnée. Enfin, les règles sont des formules probabilistes ou des corrélations multidimensionnelles.

Une requête inductive est un prédicat qui retourne un ensemble d'objets qui peuvent être des objets de bases de données classiques (comme des tuples) ou des objets d'ECD. Ces derniers n'existant pas a priori, il peut être nécessaire de les générer à l'exécution. Une fois obtenus, ceux-ci peuvent alors être stockés sous forme de metadonnées. Ainsi, il peut être possible de créer une base de règles [IVA96, MPC98]. Dans de tels cas, la requête équivaut alors à une sélection dans cette base de règles.. De manière générale, une base de données inductive doit être capable de stocker de manière persistante les objets d'ECD extraits et doit fournir un moyen de les interroger et de les manipuler. Ainsi, les requêtes peuvent avoir deux buts : la génération d'objets ou l'interrogation d'objets préalablement générés.

De plus, dans le cadre des bases de données relationnelles, le résultat d'une requête est une relation sur laquelle peuvent porter de futures requêtes. Ce principe est connu sous le nom de principe de **fermeture** et c'est l'un des concepts les plus importants dans le monde des bases de données. Il est naturellement souhaitable qu'un langage de requêtes pour l'extraction de connaissance dans les données possède également cette propriété de fermeture. Ainsi, une requête inductive prendra en entrée une instance d'une BDI et donnera comme résultat une nouvelle instance de BDI pouvant elle même être un argument d'une autre requête inductive.

Il apparaît enfin souhaitable que les requêtes relationnelles classiques constituent un sous-ensemble des requêtes inductives. Ainsi, une requête inductive pourra englober une requête relationnelle classique. A titre d'exemple, avec un langage approprié pour les bases de données inductives, on pourra écrire des requêtes correspondant aux opérations suivantes :

- Construire un classifieur sur une table relationnelle pour prédire la valeur d'un attribut particulier.
- Appliquer ce classifieur à la table initiale et créer une nouvelle table ne contenant que les tuples pour lesquels le classifieur s'est trompé.
- Appliquer un algorithme d'extraction d'itemsets fréquents sur cet ensemble de contre-exemples pour les caractériser.

Il est clair que les requêtes inductives peuvent traverser de nombreuses fois la "frontière" entre les objets de base de données (les tuples) et les objets d'ECD (les motifs), et ceci parfois à des niveaux avancés d'imbrication. De plus, ces requêtes inductives peuvent être enveloppées dans un environnement de programmation afin de fournir des API (Application Programming Interfaces) pour des applications de découverte de connaissance à l'instar des requêtes SQL qui peuvent être englobées dans des langages hôtes.

Selon Imielinski et Mannila, le développement des bases de données inductives doit se faire en deux étapes : la première consiste à définir formellement un langage pour l'extraction de

connaissance dans les bases de données et la deuxième en la conception d'outils d'optimisation de requêtes inductives permettant de traiter celles-ci grâce à des plans d'exécution efficaces. Ces plans d'exécution incluront les techniques d'apprentissage inductifs et d'analyse statistique existantes et pourront également en intégrer de nouvelles. On remarque que ce plan de développement est très similaire à celui qui fut utilisé pour les bases de données relationnelles.

Mais on peut déjà relever certaines difficultés majeures. En effet, concevoir des techniques d'optimisation de requêtes sera plus difficile dans le cadre des BDI que dans le cadre relationnel classique, à cause du plus fort pouvoir d'expression des requêtes inductives. Certaines approches génériques comme le fait de pousser les opérations de sélection dans l'arbre des requêtes peuvent aussi être utilisées ici. Dans ce cas, certaines sélections pourront être réalisées avant la génération de règles d'association par exemple. Une autre difficulté est que la frontière entre l'interrogation et la découverte dans le cadre des BDI reste floue. En effet, Imielinski et Mannila soulignent que, dans le cadre des bases de données inductives, le processus de découverte de connaissance dans les données doit être considéré comme un processus d'interrogation. Ils écrivent d'ailleurs : "Du point de vue de l'utilisateur, il n'y a pas de véritable découverte, tout est une question de puissance du langage de requête utilisé". Cela signifie que l'extraction de connaissances dans les entrepôts de données doit être réalisée en interrogeant aussi bien les données que les motifs extraits de ces données.

D'une manière plus formelle, dans le cadre des BDI, il devient possible de spécifier une tâche d'extraction comme un calcul de théories [Man97, BKM99]. Etant donnés un langage de motifs $\mathcal{L}$, une base de données $\mathbf{r}$ et un prédicat de sélection q, le but d'une requête inductive est de trouver la théorie $Th(\mathcal{L}, \mathbf{r}, q) = \{\phi \in \mathcal{L} \mid q(\phi, \mathbf{r}) \text{ est vraie}\}$. Ainsi, dans le cas des ensembles d'items, $\mathcal{L}$ est l'ensemble de tous les ensembles d'items possibles et le prédicat q peut par exemple correspondre à la contrainte de fréquence minimale sur l'ensemble de tuples $\mathbf{r}$. Si l'on considère des motifs globaux comme les arbres de décision, $\mathcal{L}$ est alors l'ensemble des arbres possibles et q est vrai uniquement pour l'arbre qui optimise une fonction objectif sur un sous-ensemble des données.

En conclusion, il apparaît que les bases de données inductives doivent fournir un langage de requête suffisamment puissant pour être utilisé tout au long du processus d'extraction de connaissances. En effet, il est clair que les processus issus du monde réel font appel à un vaste éventail de motifs différents (règles d'association, motifs séquentiels, clusters, etc) et à des interactions complexes entre données et motifs. De plus, un langage de requête pour les systèmes de gestion de bases de données inductives doit permettre :

- d'extraire une grande quantité de motifs différents afin de répondre à des tâches distinctes (descriptives, prédictives, comparatives, etc).
- de réaliser non seulement la partie "Fouille de données" d'un processus d'ECD mais aussi toutes les autres étapes, notamment le prétraitement et le post-traitement.

Dans la suite, nous allons analyser un certain nombre de langages pour l'extraction de connaissances dans les bases de données qui ont été proposés dans la littérature : `MSQL` [IV99, Vir98], `DMQL` [HFW+96, HK00], `MINE RULE` [MPC96, MPC98], et l'API `OLE DB for DM` [ole00, NCBF00, NCFB01, Cha98] proposée par Microsoft. Nous présenterons aussi rapidement le format PMML (Predictive Model Markup Language). Ces différents langages constituent des propositions intéressantes pour comprendre les besoins en matière de langage pour les BDI. Ces différentes propositions ont initialement été faites afin de mieux intégrer les bases de données avec les algorithmes d'extraction existants. Ils sont donc limités dans leurs buts, car souvent

limités à l'extraction d'un seul type de motifs sous des contraintes particulières. Néanmoins, ils possèdent plusieurs caractéristiques intéressantes qui permettent de mettre en évidence les propriétés souhaitées d'un langage pour les requêtes inductives. Afin de les comparer, nous les mettrons en œuvre sur un exemple jouet concernant l'extraction de règles d'association.

2.2 Langages de requêtes pour l'extraction de règles d'association

2.2.1 Propriétés souhaitées d'un langage de requêtes

Étant donné que les bases de données inductives ont pour but de gérer à la fois les données et les motifs ou modèles extraits à partir de ces données, un langage de requêtes pour les BDI se présentera généralement sous la forme d'une extension d'un langage de requêtes classique pour les bases de données. Plus précisement, ce langage aura été enrichi avec des primitives liées à l'exécution des différentes étapes d'un processus d'ECD, et permettant aussi de manipuler les motifs extraits ou stockés, i.e. :

- La sélection des données source pour l'extraction. Le langage doit offrir la possibilité de sélectionner (par exemple, via des requêtes standards mais aussi par d'autres procédés comme le sampling), de manipuler et d'interroger les données et les vues sur la base de données. Il doit aussi offrir la possibilité de manipuler des données multi-dimensionnelles.
- La spécification du type de motif à extraire. En effet, les processus d'ECD issus de la vie réelle recourrent à différents types de motifs comme les règles descriptives, les clusters ou les modèles prédictifs.
- La définition de la connaissance a priori (par exemple, la définition d'une hiérarchie de concepts dans le cas de l'extraction d'ensembles d'items et de règles d'association).
- La définition des contraintes que les motifs extraits doivent vérifier. Cela implique que le langage permette à l'utilisateur de spécifier les motifs intéressants (par exemple, en utilisant des mesures comme la fréquence, la généralité, la couverture, la similarité, etc).
- La satisfaction de la propriété de fermeture, c'est-à-dire que l'on doit pouvoir interroger la base de motifs extraits. Cela implique de pouvoir stocker les résultats de l'extraction dans la base de données.
- Le post-traitement des résultats. La langage doit permettre de naviguer parmi les motifs extraits, d'appliquer des modèles de sélection, et de croiser les bases de motifs et de données, en fournissant un opérateur pour, par exemple, sélectionner les données contenant certains motifs ou aggréger certains résultats.

2.2.2 Quelques langages de requêtes

`MSQL`

`MSQL` [IV99, Vir98] a été conçu à l'Université de Rutgers, dans le New Jersey, aux États-unis. Dans `MSQL`, les règles sont basées sur des descripteurs, chaque descripteur étant une expression de la forme $(A_i = a_{ij})$, où A_i est un attribut et a_{ij} est une valeur ou un ensemble de valeurs dans le domaine A_i. Un *conjunctset* est la conjonction d'un nombre quelconque de descripteurs tels qu'il n'y ait pas deux descripteurs construits sur le même attribut. Dans la pratique, `MSQL` extrait des règles propositionnelles de la forme $\mathcal{A} \Rightarrow \mathcal{B}$, où $\mathcal{A}$ est un conjunctset et $\mathcal{B}$ est un

descripteur (il s'ensuit qu'une seule proposition peut apparaître dans le conséquent de la règle). On dit qu'un tuple t d'une relation R satisfait un descripteur $(A_i = a_{ij})$ si la valeur de A_i dans t est égale à a_{ij}. En outre, t satisfait un conjunctset C si t satisfait tous les descripteurs de C. Enfin, t satisfait une règle $\mathcal{A} \Rightarrow \mathcal{B}$ s'il satisfait tous les descripteurs de $\mathcal{A}$ et $\mathcal{B}$, et t viole une règle $\mathcal{A} \Rightarrow \mathcal{B}$ s'il ne satisfait pas $\mathcal{A}$ ou $\mathcal{B}$. On peut également remarquer que le support d'une règle est défini comme le nombre de tuples satisfaisant $\mathcal{A}$ dans la relation sur laquelle l'extraction a été réalisée. De même, la confiance d'une règle est définie comme le ratio entre le nombre de tuples satisfaisant simultanément $\mathcal{A}$ et $\mathcal{B}$ et le support de la règle. Un exemple de règle extraite à partir d'une relation Emp contenant des informations sur des employés et ayant pour schéma (emp_id, job, sex, car) est $(job = doctor) \wedge (sex = male) \Rightarrow (car = BMW)$.

Les principales caractéristiques de `MSQL`, telles que présentées par ses auteurs, sont :

- *Sa capacité à englober les expressions SQL* comme les tris et le groupage et à permettre les requêtes SQL imbriquées par l'intermédiaire de la clause `WHERE`.
- *La vérification de le propriété de fermeture* et la disponibilité d'opérateurs pour permettre la manipulation des résultats issus de requêtes `MSQL` précédentes.
- *La possibilité de croiser les données et les règles* avec des opérations permettant d'identifier les sous-ensembles de données satisfaisant ou violant un ensemble donné de règles.
- *La distinction entre la génération des règles et leur interrogation.* Cela est rendu possible en séparant la génération des règles, généralement coûteuse en terme de ressources, du post-traitement des règles, qui doit être aussi interactif que possible.

`MSQL` comprend quatre opérateurs de base (cf. Section 2.2.3 pour des exemples plus détaillés) :

- `Create Encoding` qui permet de discrétiser les attributs à valeurs continues. Il est important de noter, que pendant l'extraction, la discrétisation des attributs est réalisée "à la volée", ce qui dispense de la nécessité de dupliquer la table.
- `GetRules` permet d'extraire des règles d'association dans les données et les matérialise dans une base de règles. Sa syntaxe précise est la suivante :

  ```
  %[PROJECT Body, Consequent, Confidence, Support]
  GETRULES(C) [AS R1] [INTO <rulebase_name>]
  [WHERE (RC|PC|MC|SQ)]
  [SQL-group-by clause] [USING encoding-clause]
  ```

  Une requête de type `GetRules` peut spécifier plusieurs types de contraintes sur les règles dans sa clause `WHERE` :
  - Les conditions sur le format de la règle (Rule Condition, RC), qui permettent de restreindre l'ensemble des items pouvant apparaître dans les éléments de la règle. Ces conditions ont la forme suivante :

    ```
    Body { in | has | is } <descriptor-list>
    Consequent { in | is } <descriptor-list>
    ```

  - Les conditions d'élagage (Pruning Conditions, PC), qui définissent les seuils pour les valeurs de support et de confiance des règles, ainsi que des contraintes sur la longueur des règles. Ces conditions ont la syntaxe suivante :

    ```
    confidence <relop> <float-val in [0.0,1.0]>
    support <relop> <integer>
    support <relop> <float-val in [0.0,1.0]>
    length <relop> <integer>
    relop ::= { < | <= | = | >= | > }
    ```

  - Les conditions d'exclusion mutuelle (Mutex Conditions,MC), qui permettent d'éviter

que deux attributs n'apparaissent dans la même règle (ce qui peut être utile si on a des connaissances sur des dépendances fonctionnelles entre les attributs de la relation). Leur syntaxe est :

```
Where <other-conditions>
    { AND | OR }    mutex(method, method [, method])
    [{ AND | OR}    mutex(method, method [, method])]
```

  - Les conditions sur les sous-requêtes (Subquery conditions, SQ), connectées au reste de la requête par le mot clé usuel `WHERE` en utilisant les connecteurs `IN` et `(NOT) EXISTS`.
- `SelectRules` peut être utilisé pour le post-traitement des règles, c'est-à-dire l'interrogation des bases de règles préalablement extraites. Sa syntaxe précise est la suivante :
  `SelectRules(rulebase_name) [where <conditions>]`
  où `<conditions>` peut porter sur le corps, le conséquent, le support ou la confiance de la règle.
- `Satisfies` et `Violates` qui permettent de croiser entre eux les ensembles de données et les règles. Ces deux primitives peuvent être utilisées conjointement dans une commande de sélection de données, à l'intérieur de la clause `WHERE` d'une requête.

`MINE RULE`

`MINE RULE` a été conçu à l'Institut Polytechnique de Turin et l'Institut Polytechnique de Milan, Italie [MPC96, MPC98]. Cet opérateur extrait un ensemble de règles d'association à partir d'une base de données et les stocke en retour dans cette même base de données dans une relation séparée.

Une règle d'association extraite par `MINE RULE` à partir d'une relation source quelconque est définie comme suit : considérons une relation source de schéma $\mathcal{S}$. Soient $\mathcal{R}$ et $\mathcal{G}$ deux sous-ensembles disjoints de $\mathcal{S}$, appelés respectivement schéma des règles et schéma de groupage. Une règle d'association est extraite à partir d'au moins un groupe de la relation source, où chaque groupe est un partition de la relation source selon les valeurs des attributs de $\mathcal{G}$. Une règle d'association est de la forme $\mathcal{A} \Rightarrow \mathcal{B}$, où $\mathcal{A}$ et $\mathcal{B}$ sont des ensembles d'éléments de règle ($\mathcal{A}$ est le corps de la règle et $\mathcal{B}$ la tête de la règle). Les éléments d'une règle sont pris parmi les tuples d'un groupe. En particulier, chaque élément de règle est une projection sur un sous-ensemble de $\mathcal{R}$. On remarquera cependant, que, pour une requête `MINE RULE` donnée, les schémas du corps et de la tête de la règle sont uniques (même s'ils peuvent, bien sûr, être différents).

Un exemple de règle extraite par `MINE RULE` sur une relation $Emp(emp\_id, job, sex, car)$ avec un groupage sur l'attribut emp_id et avec un schéma de corps de règle égal à (job, sex) et un schéma de tête de règle égal à (car) pourrait être le suivant : $\{(doctor, male)\} \Rightarrow \{(BMW)\}$. Cette règle est extraite depuis chaque tuple de la relation, parce que chaque groupe coïncide avec un tuple de la relation. En revanche, si on travaille sur une relation $Sales(transaction\_id$, $item$, $customer$, $payment)$, la collecte des données par achats, groupée par client et avec un schéma de règle égal à $(item)$ (et où les schémas du corps et de la tête de la règle coïncident), peut conduire à des règles de la forme : $\{(pasta), (oil), (tomatoes)\} \Rightarrow \{(wine)\}$.

Le langage `MINE RULE` est une extension de SQL. Ses principales caractéristiques sont :

- *La sélection de l'ensemble de données source pertinent pour l'extraction* de connaissances. Celle-ci peut être faite à différents niveaux de granularité, aussi bien au niveau des tuples (sélection classique d'une sous-ensemble des tuples de la relation) qu'au niveau des groupes (selon les conditions de groupages). La condition de groupage permet de déterminer quelles

sont les données de la relation qui prendront part à la détermination des règles d'association. Cette condition est similaire à la condition de groupage usuelle disponible en SQL. La définition des groupes, c'est-à-dire des partitions de la base de données originale à partir desquelles les règles seront extraites, est faite au moment de l'exécution et n'est pas fixée a priori par le choix des clés de la relation source (comme en DMQL).

- La définition de la structure des règles. Plus précisément, il est possible de spécifier si l'on cherche des règles mono-dimensionnelles ou multi-dimensionnelles. Une règle est mono-dimensionnelle si ses élements sont tous construits sur le même attribut (par exemple, des produits achetés ensembles), elle est multi-dimensionnelle sinon (on peut alors avoir des règles qui lient les produits achetés avec l'âge du consommateur par exemple). La structure des règles peut aussi être contrainte en précisant les cardinalités du corps et de la tête de la règle.
- La définition des contraintes appliquées à différents niveaux de granularité. On peut distinguer deux catégories de contraintes :
  - Les contraintes appliquées au niveau de la règle (Mining Conditions). Elles sont évaluées sur chaque tuple construit sur les attributs apparaissant dans la règle.
  - Les contraintes appliquées au niveau du cluster (Cluster Conditions). Un cluster est un sous-groupe d'un groupe principal (défini par la clause GROUP BY) et qui vérifie certaines conditions particulières, définies par des contraintes sur les valeurs de certains attributs. Les clusters peuvent être vus comme un groupage de deuxième niveau. En présence de clusters, les corps et têtes de règles sont extraits à partir de paires de clusters du même groupe vérifiant les contraintes de cluster. Par exemple, les clusters et les contraintes de cluster peuvent être utilisés afin d'extraire des règles d'association dans lesquelles le corps et la te sont ordonnées, ce qui peut constituer, par exemple, des motifs séquentiels élémentaires.
- La définition de mesures d'évaluation des règles. D'un point de vue pratique, le langage permet de définir des seuils de support et de confiance pour l'extraction de règles d'association.

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

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

<BodyDescr>:= [ <CardSpec> ] <AttrList> AS BODY
<BodyDescr>:= [ <CardSpec> ] <AttrList> AS HEAD
<CardSpec>:=<Number> .. (<Number> | n)
<AttrList>:=<AttrName>[,<AttrList>]
```

DMQL

DMQL a été conçu à l'Université Simon Fraser au Canada [HFW+96, HK00]. En DMQL, une règle d'association est une relation entre les valeurs de deux ensembles de prédicats évalués sur les

relations d'une base de données. Ces prédicats sont de la forme $P(X, c)$ où P est un prédicat prenant le nom d'un attribut d'une relation sous-jacente, X est une variable et c est une constante appartenant au domaine de l'attribut. Le prédicat est vérifié dans une relation s'il existe un tuple identifié par la variable X dont l'attribut homonyme prend la valeur c. On remarque qu'il est possible pour un prédicat d'être évalué sur différentes relations de la base de données. Par exemple, `DMQL` peut extraire des règles comme $town(X,'London') \Rightarrow buys(X,'DVD')$ où `town` et `buys` peuvent être deux attributs issus de relations différentes et X est un attribut présent dans les deux relations. Une règle peut être de deux types : une règle mono-dimensionnelle contient des occurences multiples d'un seul prédicat (par exemple, *buys*), alors qu'une règle multi-dimensionnelle implique plus de prédicats, chacun apparaissant une seule fois dans la règle. Cependant, la présence d'une ou plusieurs instances d'un même prédicat dans une même règle paut être imposée en ajoutant le symbole + à la suite du prédicat dans la requête `DMQL`. Une autre caractéristique importante de `DMQL` est qu'il permet de guider le processus d'extraction de connaissances en utilisant des *metamotifs*. Les métamotifs sont une sorte de modèle qui permet de contraindre l'aspect syntaxique des règles devant être extraites. De plus, ils permettent de prendre en compte certaines hypothèses faites pas l'utilisateur et il est possible d'y intégrer des contraintes plus complexes. Un exemple de metamotif pourrait être : $town(X : customer, London) \wedge income(X, Y) \Rightarrow buys(X, Z)$, qui restreint la découverte de règles à celles portant sur la ville et le niveau de revenus en partie gauche et sur les produits achetés en partie droite. En outre, les métamotifs peuvent autoriser ou non la présence d'un prédicat non instancié que l'algorithme se chargera d'instancier avec un nom d'attribut valide sur la relation lors de l'extraction. Par exemple, si l'on souhaite extraire des règles d'association décrivant les caractéristiques des consommateurs qui achètent des ensembles de produits revenant fréquemment dans la relation source, on pourrait utiliser le métamotif suivant pour contraindre l'espace de recherche des règles : $P(X : customer, W) \wedge Q(X, Y) \Rightarrow buys(X, Z)$ où P et Q sont des variables correspondant à des prédicats qui peuvent être instanciés avec les attributs pertinents de la relation analysée. X est une clé de la relation *customer*, W, Y, et Z sont des variables qui peuvent prendre les valeurs dans les domaines de leurs attributs respectifs.

`DMQL` consiste essentiellement en la spécification de quatre primitives majeures en Fouille de Données portant sur les points suivants :

- La sélection de l'ensemble de données pertinent pour le processus de fouille de données. Cette primitive peut être spécifiée de la même manière qu'avec un langage de requêtes conventionnel.
- La spécification du type de motif à extraire. Plus précisément, on peut spécifier si l'on cherche :
  - des règles d'association,
  - des règles de classification (qui attribuent aux données des classes disjointes correspondantes aux différentes valeurs d'un attribut donné).
  - des caractéristiques (i.e. des descriptions qui constituent un résumé des propriétés communes des tuples d'un ensemble de données)
  - des comparaisons (i.e. des descriptions qui permettent de comparer le nombre total de tuples appartenant à une classe par rapport à certaines classes de référence).
  - des relations généralisées, obtenues en généralisant un ensemble de données correspondant à des concepts de bas niveau en utilisant une hiérarchie de concepts particulière.
- La définition de mesure d'intérêt.
  Elles se fait via un ensemble de contraintes. Ainsi, pour les règles d'association, au delà

des mesures classiques de support et de confiance, DMQL permet de définir des seuils sur les mesures de bruit et sur la nouveauté de la règle, afin de séléctionner les règles les plus spécifiques.

La grammaire de DMQL pour l'extraction de règles d'association est une extension du celle du SQL conventionnel. Ainsi, il est possible d'utiliser des opérateurs relationnels comme HAVING, WHERE, ORDER BY et GROUP BY, mais il est aussi possible de spécifier la base de données sur laquelle on travaille, sélectionner les attributs pertinents pour l'extraction, définir des hiérarchies de concepts pour la généralisation ainsi que les seuils pour les différentes mesures d'intérêt et ainsi de guider le processus de découverte en utilisant des metamotifs. La syntaxe générale d'une requête DMQL est la suivante :

use database ⟨*database_name*⟩
{**use hierarchy** ⟨*hierarchy_name*⟩ **for** ⟨*attribute_or_dimension*⟩ }
in relevance to ⟨*attribute_or_dimension_list*⟩
mine associations [as ⟨*pattern_name*⟩] [**matching** ⟨*metapattern*⟩]
from ⟨*relation*(*s*)/*cube*(*s*)⟩ [**where** ⟨*condition*⟩]
[**order by** ⟨*order_list*⟩]
[**group by** ⟨*grouping_list*⟩][**having** ⟨*condition*⟩]
with ⟨*interest_measure*⟩ **threshold** = value

OLE DB for DM

OLE DB DM a été conçu par Microsoft Corporation [ole00, NCBF00, NCFB01]. C'est une extension de l'API (Application Programming Interface) OLE DB qui permet à n'importe quelle application d'accéder facilement à une source de données relationnelles sous le famille de Systèmes d'Exploitation Windows. La principale motivation à l'origine du développement d'OLE DB for DM est la simplification du développement des projets Data Mining ayant besoin d'un couplage fort et intégré avec le SGBD. En effet, le travail de recherche en Data Mining se focalise généralement sur des analyses de passage à l'échelle et sur des algorithmes travaillant sur des fichiers plats extraits de la base de données. Cette situation est génératrice de problèmes dans le déploiement de solutions Data Mining car la gestion et la maintenance des modèles de données se fait à l'extérieur des SGBD et doit donc être réalisée de manière *ad hoc*. OLE DB DM a pour principal but de faciliter la tâche de communication entre les algorithmes de Data Mining et les sources de données.

L'idée clé d'OLE DB DM est la définition de Modèles de Fouille de Données (MFD ou Data Mining Models, DMM), qui sont un type particulier de tables dont les lignes contiennent une description synthétique des données (appelée ensemble de cas). L'utilisateur peut peupler ce modèle avec des données issues de la base. Une fois la tâche de fouille de données réalisée, il est possible d'utiliser le Modèle de Fouille de Données pour prédire la valeur de certains attributs sur de nouveaux exemples ou de le parcourir afin de réaliser des opérations de post-traitement comme la visualisation.

La représentation des données dans le modèle dépend du format de données produit par l'algorithme. Celui-ci peut sortir ses résultats en utilisant le format PMML par exemple (Predictive Model Markup Language [pmm]). PMML est un standard proposé par le DMG et est basé sur XML. C'est un langage de balises pour la description de modèles statistiques et de fouille de données. PMML permet de décrire les entrées des modèles de fouilles de données et les paramètres utilisés

pour la génération des motifs eux-mêmes.

`OLE DB DM` repose sur un langage semblable à SQL qui permet aux applications clientes de réaliser plusieurs opérations clés du cadre de travail de `OLE DB DM` : définition d'un DMM (avec le mot clé `CREATE MINING MODEL`), exécution d'un algorithme d'extraction externe sur des données sous forme relationnelle, peuplement du modèle d'extraction (mot clé `INSERT INTO`), prédiction de la valeur de certains attributs sur de nouvelles données (`PREDICTION JOIN`), parcours du modèle (mot clé `SELECT`).

Ainsi, l'élaboration d'un modèle d'extraction en `OLE DB DM` peut être réalisée en utilisant des requêtes SQL. Une fois que l'algorithme d'extraction a été exécuté, il devient possible de réaliser certaines opérations de croisement entre le DMM et des données ayant un schéma identique à celui du DMM en utilisant le mot clé `PREDICTION JOIN`. Il s'agit d'une variante un peu particulière de la jointure classique de SQL qui permet de prédire la valeur de certains attributs des données en entrée en appliquant un DMM donné, à la condition que ces attributs aient été indiqués comme attributs cibles lors de la construction du DMM.

La grammaire pour la création d'un data mining model est la suivante :

```
<dm_create>::=CREATE MINING MODEL <identifier> (<col_def_list>)
              USING <algorithm> [(<algo_param_list>)]


<col_def_list>::= <col_def> |<col_def_list> , <col_def>
<col_def>::= <col_def_reg> | <col_def_tbl>
<col_def_reg>::= <identifier> <col_type> [<col_distribution>]
          [<col_binary>] [<col_content>] [<col_content_qual>]
        [<col_qualif>] [<col_prediction>] [<relation_clause>]

<col_def_tbl> ::= <identifier> TABLE <col_prediction>
                    ( <col_def_list> )

// 2 algorithms currently implemented in SQL server 2000
<algorithm> ::= MICROSOFT_DECISION_TREES | MICROSOFT_CLUSTERING

<algo_param_list>::=<algo_param> | <algo_param>,<algo_param_list>
<algo_param>::= <identifier> = <value>

<col_type>::= LONG | BOOLEAN | TEXT | DOUBLE | DATE

<col_distribution>-> NORMAL | UNIFORM

<col_binary>::= MODEL_EXISTENCE_ONLY | NOT NULL

<col_content>::= DISCRETE | CONTINUOUS
          | DISCRETIZED ( [<disc_method> [, <numeric_const>]] )
          | SEQUENCE_TIME

<disc_method>::=AUTOMATIC | EQUAL_AREAS | THRESHOLDS | CLUSTERS
```

```
<col_content_qual>::= ORDERED | CYCLICAL

<col_qualif>::= KEY | PROBABILITY | VARIANCE | STDEV | STDDEV
              | PROBABILITY_VARIANCE | PROBABILITY_STDEV
              | PROBABILITY_STDDEV | SUPPORT

<col_prediction>::= PREDICT | PREDICT_ONLY

<relation_clause>::= <related_to_clause> | <of_clause>

<related_to_clause>::=RELATED TO <identifier> | RELATED TO KEY

<of_clause>::= OF <identifier> | OF KEY
```

On remarque que cette grammaire permet de spécifier différentes propriétés des attributs. Par exemple, elle permet de spécifier le rôle d'un attribut dans le modèle (clé), le type de l'attribut, si le domaine de l'attribut est ordonné ou cyclique, s'il est discret ou continu (et dans cas, il est possible de spécifier la discrétisation utilisée), s'il correspond à une mesure de temps, etc. Il est possible de spécifier une probabilité ou une autre valeur statistique associée à une valeur d'attribut. La probabilité spécifie alors la certitude qu'on a du fait que la valeur de l'attribut soit correcte.

Le mot clé `PREDICT` caractérise les attributs de prédiction. Cela signifie que cet attribut sera un attribut cible lors de la construction d'un DMM prédictif (par exemple, utilisant une technique d'induction d'arbres de décision). Puis lors de l'utilisation de ce DMM sur un nouveau jeu de données, c'est la valeur de cet attribut qui sera prédite.

`RELATED TO` permet d'associer l'attribut à un autre attribut, par exemple pour une relation de clé étrangère, ou parce que l'attribut est utilisé pour classifier les valeurs d'un autre attribut.

On remarque que la règle de production <`col_def_tbl`> permet d'imbriquer des tables à l'intérieur d'un Data Mining Model. Les tables imbriquées sont des tables stockées comme des valeurs d'attribut dans une table externe. Les données en entrée d'un algorithme d'extraction sont souvent obtenues en regroupant et en joignant l'information éclatée dans différentes tables d'une base de données. Par exemple, les informations sur les clients et les ventes sont généralement conservées dans deux tables différentes. Ainsi, lorsque l'on fait la jointure des tables clients et ventes, il est possible de stocker dans une table imbriquée du modèle tous les produits achetés par un client donné. Ainsi, les tables imbriquées permettent de réduire l'information redondante d'un modèle.

Remarquons que `OLE DB DM` semble particulièrement conçu pour les tâches prédictives, i.e. pour prédire les valeurs d'un attribut dans une table relationnelle. En effet, l'implémentation courante d'`OLE DB DM` dans Microsoft SQL Server 2000, offre seulement deux algorithmes (Microsoft Decision Trees and Microsoft Clustering) conçus tous deux pour la prédiction d'attribut. Les algorithmes utilisant les data mining models pour la découverte de règles d'association, et plus généralement pour des tâches sans but prédictif, ne semblent pas pour l'instant être supportées par `OLE DB DM`. Cependant, selon les specifications [ole00], `OLE DB DM` devrait bientôt être étendu pour supporter l'extraction de règles d'association.

Remarquons qu'il est également possible de créer directement un mining model conforme au standard PMML en utilisant la commande suivante :

```
<pmml_create>::=CREATE MINING MODEL <id> FROM PMML <string>
```

Nous rappelons ici le schéma utilisé par PMML pour la définition de modèles basés sur des règles d'association.

```
 <!ENTITY  \% FIELD-USAGE-TYPE "(active |
                                predicted |
                                supplementary)" >

 <!ENTITY  \% OUTLIER-TREAT-METHOD "( asIs |
                                     asMissingValues |
                                     asExtremeValues ) " >

 <!ENTITY  \% MISS-VALUE-TREAT-METHOD "(asIs | asMean |
                                       asMode | asMedian |
                                       asValue) " >

<!ELEMENT MiningField (Extension*)>
<!ATTLIST MiningField
  name                    \%FIELD-NAME;              #REQUIRED
  usageType               \%FIELD-USAGE-TYPE;        "active"
  outliers                \%OUTLIER-TREAT-METHOD;    "asIs"
  lowValue                \%NUMBER;                  #IMPLIED
  highValue               \%NUMBER;                  #IMPLIED
  missingValueReplacement CDATA                      #IMPLIED
  missingValueTreatment   \%MISS-VALUE-TREAT-METHOD; #IMPLIED

<!ELEMENT MiningSchema   (MiningField+) >
```

Remarquons que selon cette spécification, il est possible de spécifier le schéma d'un modèle en donnant le nom, le type et le domaine de chaque attribut. De plus, il est possible de spécifier la méthode de traitement à utiliser si la valeur est manquante ou s'il s'agit d'un outlier par rapport à la valeur prédite pour cet attribut.

2.2.3 Exemple comparatif

Nous allons décrire ici un processus complet d'extraction de connaissances dans les données tiré de l'exemple classique de l'analyse de panier. Ce processus nous servira d'exemple comparatif pour montrer les forces et les faiblesses des différents systèmes. Nous considérons l'information disponible dans les relations *Sales*, *Transactions* et *Customers* qui sont représentées dans la Figure 2.1. La relation *Sales* contient des informations sur les items vendus par transaction ; la relation *Transactions* permet d'identifier les clients qui ont effectués les transactions ainsi que les modes de paiement employés ; enfin la relation *Customers* collecte les informations sur les clients.

transaction_id	item
1	ski_pants
1	hiking_boots
2	col_shirts
2	brown_boots
3	col_shirts
3	brown_boots
4	jackets
5	col_shirts
5	jackets
6	hiking_boots
6	brown_boots
7	ski_pants
7	hiking_boots
7	brown_boots
8	ski_pants
8	hiking_boots
8	brown_boots
8	jackets
9	hiking_boots
10	ski_pants
11	ski_pants
11	brown_boots
11	jackets

transaction_id	customer	payment
1	c1	credit_card
2	c2	credit_card
3	c3	cash
4	c4	credit_card
5	c5	credit_card
6	c6	cash
7	c7	credit_card
8	c8	credit_card
9	c9	credit_card
10	c3	credit_card
11	c2	cash

customer_id	customer_age	job
c1	26	employee
c2	35	manager
c3	48	manager
c4	39	engineer
c5	46	teacher
c6	25	student
c7	29	employee
c8	24	student
c9	28	employee

FIG. 2.1 – Table `Sales` (sur la gauche) ; Table `Transactions` (en haut à droite) ; Table `Customers` (en bas à droite).

A partir de l'information contenue dans ces tables, on souhaite extraire des règles d'association entre les produits achetés et l'âge des consommateurs sur l'ensemble des transactions réalisées avec une carte bancaire. Les règles d'association découvertes seront utilisées pour prédire l'âge des clients en fonction de leurs habitudes d'achat. Cette étape de la fouille de données nécessite d'abord de réaliser certaines opérations de prétraitement (sélection des transactions réalisées avec une carte de crédit et discrétisation de l'attribut `age`) afin d'avoir un jeu de données compatible avec les extractions qui seront réalisées ensuite. L'étape d'extraction consistera en l'application d'un algorithme d'extraction de règles d'association.

Supposons qu'après analyse des résultats des extractions précédentes, nous sommes uniquement intéressés par les transactions qui violent certains motifs extraits. En particulier, nous sommes intéressés par les règles d'association entre les ensembles de produits achetés apparaissant dans les transactions violant les règles possédant `'ski_pants'` dans leur partie gauche.

Dans ce but, nous pouvons réaliser une opération de croisement entre les règles et les données originales, en sélectionant les tuples de la table source qui violent les règles sus-citées. Ensuite nous réaliserons une deuxième extraction basée sur les résultats de l'étape précédente, où nous cherchons les règles d'association entre deux ensembles d'items avec un seuil de confiance très

t_id	ski_pants	hiking_boots	col_shirts	brown_boots	jackets	customer_age	payment
t1	1	1	0	0	0	26	credit_card
t2	0	0	1	1	0	35	credit_card
t3	0	0	1	1	0	48	cash
t4	0	0	0	0	1	39	credit_card
t5	0	0	1	0	1	46	credit_card
t6	0	1	0	1	0	25	cash
t7	1	1	0	1	0	29	credit_card
t8	1	1	0	1	1	24	credit_card
t9	0	1	0	0	0	28	credit_card
t10	1	0	0	0	0	41	credit_card
t11	1	0	0	1	1	36	cash

TAB. 2.1 – `Boolean_Sales` : Table utilisée avec `MSQL`.

élevé. Enfin, afin de compléter le processus, nous réaliserons deux opérations de post-traitement sur le nouvel ensemble de règles obtenu :

- Sélection des règles possédant 2 items dans le corps
- Sélection des règles ayant le plus grand corps possible pour une conclusion donnée.

La première chose à faire est de représenter les données sources dans un format compatible avec l'exécution de requêtes `MSQL`. En effet, `MSQL` prend en entrée une relation unique obtenue en faisant la jointure des relations *Sales*, *Transactions* et *Customers* sur les attibuts *transaction_id* et *customer_id*. De plus, la relation doit être encodée sous une forme binaire de telle sorte que chaque tuple représente une transaction avec autant de variables booléennes qu'il y a d'items possibles à acheter pour un client. On obtient ainsi la relation présentée dans la Table 2.1.

Cette étape de transformation des données met en évidence l'une des principales faiblesses de `MSQL`. `MSQL` est conçu pour découvrir des règles propositionnelles satisfaites par les valeurs des attributs d'un tuple d'une table. Si le nombre d'items à partir desquels une règle propositionnelle peut être générée est très important (comme par exemple le nombre réel de produits différents dans un supermarché), la table d'entrée obtenue sera d'une taille démesurée. En effet, pour chaque transaction, elle contiendra tous les items possibles (même ceux qui n'ont pas été achetés). Il est important de prendre en compte ce problème des tables booléennes, car leur existence est indispensable à l'utilisation du langage `MSQL`. Ce langage n'est donc pas très souple pour ce qui concerne son format d'entrée. De plus, la création de tables booléennes nécessite une transformation des données qui est coûteuse (notamment lorque le volume des tables est important). et doit être réalisée à chaque fois qu'un nouveau problème est soumis. Il est cependant vrai qu'il ne s'agit pas d'un problème insurmontable dans la mesure où la table binarisée n'a pas vocation à être lue par l'utilisateur mais utilisée par l'algorithme. Mais, cette transformation explicite peut demeurer complexe et inhabituelle pour un utilisateur souhaitant travailler avec des tables relationnelles classiques.

Prétraitement - Etape 1 : sélection du sous-ensemble de données. Nous sommes uiquement intéressés par les clients payants avec une carte de crédit. L'utilisation de `MSQL` requiert une sélection des données devant être fouillées avant l'exécution effective de la tâche d'extraction. La relation sur laquelle nous travaillerons est supposée avoir été correctement

t_id	ski_pants	hiking_boots	col_shirts	brown_boots	jackets	e_age	payment
t1	1	1	0	0	0	2	credit_card
t2	0	0	1	1	0	3	credit_card
t4	0	0	0	0	1	3	credit_card
t5	0	0	1	0	1	4	credit_card
t7	1	1	0	1	0	2	credit_card
t8	1	1	0	1	1	2	credit_card
t9	0	1	0	0	0	2	credit_card
t10	1	0	0	0	0	4	credit_card

TAB. 2.2 – `View_on_Sales` : table obtenue après les phases de prétraitement

extraite par une sélection sur l'ensemble de données préexistantes dans la Table 2.1. On peut pour cela utiliser une vue nommée *View_on_Sales*. Dans la suite, nous supposerons que nous travaillerons sur cette relation restreinte.

Prétraitement - Etape 2 : codage de l'âge. `MSQL` fournit une méthode pour faciliter l'encodage de certains attributs continus. Il est important de remarquer que `MSQL` peut réaliser cette discrétisation "à la volée", de telle sorte que l'encodage intermédiaire n'apparaisse pas dans les résultats finaux. La requête suivante permet de discrétiser l'attribut `age` :

```
CREATE ENCODING e_age ON View_on_Sales.customer_age AS
BEGIN
   (MIN, 9, 0), (10, 19, 1), (20, 29, 2), (30, 39, 3),    (1)
   (40, 49, 4), (50, 59, 5), (60, 69, 6), (70, MAX,7), 0
END;
```

La relation obtenue après ces deux étapes de prétraitemnet est décrite dans la Table 2.2.

Extraction de règles entre un ensemble d'items et l'âge du client. On souhaite extraire les règles associant un ensemble d'items à l'âge du consommateur et qui ont un support supérieur à 2 et une confiance supérieure ou égale à 50 %.

```
GETRULES(View_on_Sales) INTO SalesRB
WHERE BODY has {(ski_pants=1) OR (hiking_boots=1) OR      (2)
  (col_shirts=1) OR (brown_boots=1) OR (jackets=1)} AND
  Consequent is {(Age = *)} AND support>2 AND confidence>=0.5
USING e_age FOR customer_age
```

Cet exemple met en évidence une limite de `MSQL` : si le nombre d'items est élevé, le nombre de prédicats dans la clause `WHERE` va augmenter de manière correspondante. Les règles résultat sont décrites dans la Table 2.3.

Recherche des exceptions dans les données. On va sélectionner les tuples de la vue *View_on_Sales* qui violent toutes les règles extraites possédant l'attribut `ski_pants` dans leurs parties gauches (la première et la dernière règle dans la Table 2.3).

```
INSERT INTO Sales2 AS
SELECT * FROM View_on_Sales
WHERE VIOLATES ALL (                                      (3)
      SELECTRULES(SalesRB) WHERE BODY HAS {(ski_pants=1)})
```

Body	Consequent	Support	Confidence
(ski_pants=1)	(customer_age=[20,29])	3	75%
(hiking_boots=1)	(customer_age=[20,29])	4	100%
(brown_boots=1)	(customer_age=[20,29])	3	66%
(ski_pants=1) $\wedge$ (hiking_boots=1)	(customer_age=[20,29])	3	100%

TAB. 2.3 – Table `SalesRB` produite par `MSQL` après la première phase d'extraction

t_id	ski_pants	hiking_boots	col_shirts	brown_boots	jackets	e_age
t2	0	0	1	1	0	3
t4	0	0	0	0	1	3
t5	0	0	1	0	1	4
t9	0	1	0	0	0	2
t10	1	0	0	0	0	4

TAB. 2.4 – Tuples violant toutes les règles (au sens de `MSQL`) dans SalesRB possédant ski_pants dans leur partie gauche

On obtient les tuples constituant la Table 2.4. Nous rappelons qu'en `MSQL`, une transaction viole une règle dès qu'elle ne supporte pas le corps ou le conséquent de celle-ci. C'est ainsi que les transactions $t2$ et $t10$ violent la règle $(ski\_pants = 1) \Rightarrow (customer\_age = [20; 29])$ ($t2$ car elle ne suppore pas $(ski\_pants = 1)$ et $t10$ car elle ne supporte pas $(customer\_age = [20; 29])$).

Extraction de règles possédant deux items dans le corps. `MSQL` ne permet pas de spécifier une conjonction d'un nombre quelconque de descripteurs dans la conclusion de la règle. C'est pourquoi, pour cette étape, on peut seulement extraire les règles d'association entre un ensemble d'items en partie gauche et un unique item en partie droite. L'ensemble de règles obtenu contient seulement : $(brown\_boots = 1) \Rightarrow (color\_shirts = 1)$ with support=1 and confidence=100%.

```
GETRULES(Sales2) INTO SalesRB2
WHERE (Body has {(hiking_boots=1) OR (col_shirts=1)
                      OR (brown_boots=1)}
               AND Consequent is {(jackets=1)}
 OR Body has {(col_shirts=1) OR (brown_boots=1) OR (jackets=1)}
               AND Consequent is {(hiking_boots=1)}
 OR Body has {(brown_boots=1) OR (jackets=1)                (4)
                  OR (hiking_boots=1)}
               AND Consequent is {(col_shirts=1)}
 OR Body has {(jackets=1) OR (hiking_boots=1)
                   OR (col_shirts=1)}
               AND Consequent is {(brown_boots=1)})
 AND support>=0.0 AND confidence>=0.9
USING e_age FOR customer_age
```

On remarque que dans cette requête, la clause `WHERE` permet de spécifier différentes conditions sur les élements `Body` et `Consequent` de la règle. On souhaite en effet voir apparaître dans le

corps une conjonction de tous les attributs sauf celui qu'on force à apparaitre en partie droite. Il est ici possible d'écrire cette requête car le nombre total d'items demeure très faible, mais cela serait pratiquement impossible sur un exemple réel, où le nombre de conditions apparaissant en clause `WHERE` exploserait.

Post-traitement - Etape 1 : manipulation des règles. Sélection des règles avec 2 items dans le corps.

Comme `MSQL` extrait des règles avec un seul item en partie droite et qu'il permet de spécifier la longueur des règles à extraire grâce au mot-clé `length`, il suffit de ne sélectionner que les règles contenant 3 items.

```
SelectRules(SalesRB) where length=3                    (5)
```

La seule règle qui satisfait cette condition est :
$(ski\_pants = 1) \wedge (hiking\_boots = 1) \Rightarrow (customer\_age = [20; 29])$

Post-traitement - Etape 2 : Extraction de règles avec un corps maximal. Cela revient à rechercher un ensembles de règles où il n'y ait pas deux règles avec la même partie droite et où le corps d'une des règles est inclus dans le corps de l'autre règle.

```
SELECTRULES(SalesRB) AS R1
WHERE NOT EXISTS (SELECTRULES(SalesRB) AS R2
                  WHERE R2.body has R1.body            (6)
                    AND NOT (R2.body is R1.body)
                    AND R2.consequent is R1.consequent )
```

Il y a deux règles qui satisfont cette condition :
$(ski\_pants = 1) \wedge (hiking\_boots = 1) \Rightarrow (customer\_age = [20; 29])$
$(brown\_boots = 1) \Rightarrow (customer\_age = [30, 39])$

Avantages et inconvénients de `MSQL`.

Clairement, le principal avantage de `MSQL` est qu'il est possible d'interroger aussi bien les règles extraites que les données, en utilisant les mots clés **SelectRules** sur les bases de règles et **GetRules** sur les données. Un autre point fort de `MSQL` est qu'il a été conçu comme une extension du SQL classique, rendant le langage assez facile à comprendre. Par exemple, il est assez simple de tester les règles sur un jeu de données et de réaliser des opérations de croisement entre les données originales et les résultats d'une extraction, en utilisant les mots clés `SATISFIES` et `VIOLATES`. Pour pouvoir être considéré comme un bon candidat pour les bases de données inductives, il est clair que `MSQL`, qui est essentiellement conçu autour de la phase d'extraction, doit être étendu, particulièrement avec une meilleure prise en charge des étapes de pré- et post-traitement. Ainsi, même s'il existe déjà certains opérateurs de prétraitement comme `ENCODE` pour la discrétisation des attributs continus, il ne fournit aucune primitive pour des manipulations plus complexes comme le sampling. De plus, les tuples sur lesquels la tâche d'extraction doit être réalisée sont supposés avoir été sélectionnés à l'avance. Concernant la phase d'extraction, l'utilisateur peut spécifier certaines contraintes sur les règles à extraire (par exemple, l'inclusion d'un item dans le corps ou la tête de la règle, l'exclusion mutuelle entre certains items, etc), ainsi que les seuils de support et de confiance à utiliser. Toutefois, il serait utile d'avoir la possibilité de spécifier des contraintes et des mesures d'intérêt plus complexes, voire définies par l'utilisateur.

transaction_id	item	customer_age	payment
1	ski_pants	26	credit_card
1	hiking_boots	26	credit_card
2	col_shirts	35	credit_card
2	brown_boots	35	credit_card
3	col_shirts	48	cash
3	brown_boots	48	cash
4	jackets	39	credit_card
5	col_shirts	46	credit_card
5	jackets	46	credit_card
6	hiking_boots	25	cash
6	brown_boots	25	cash
7	ski_pants	29	credit_card
7	hiking_boots	29	credit_card
7	brown_boots	29	credit_card
8	ski_pants	24	credit_card
8	hiking_boots	24	credit_card
8	brown_boots	24	credit_card
8	jackets	24	credit_card
9	hiking_boots	28	credit_card
10	ski_pants	48	credit_card
11	ski_pants	35	cash
11	brown_boots	35	cash
11	jackets	35	cash

TAB. 2.5 – Vue `SalesView` obtenue en joignant les relations d'entrée.

2.2.4 `MINE RULE`

`MINE RULE` ne requiert pas une transformation spécifique du format des données en entrée. Ainsi, nous pouvons travailler sur des données représentées sous forme de relations normalisées (*Sales*, *Transactions* et *Customers* représentées sur la Figure 2.1) ou sur une vue résultant de la jointure de celles-ci. Cette vue, nommée *SalesView*, est représentée dans la Table 2.5 et nous supposons que c'est elle qui sera utilisée pour notre tâche d'extraction. Il n'est pas absolument nécessaire d'avoir recours à une vue mais cela permet de simplifier les requêtes SQL en rassemblant toutes les informations initialement éclatées dans une seule table. Ainsi, l'utilisateur peut focaliser son attention sur l'écriture de requêtes prenant en compte les contraintes pertinentes pour l'extraction.

Prétraitement - Etape 1 : Sélection du sous-ensemble de données à traiter. Contrairement à `MSQL`, `MINE RULE` ne nécessite pas obligatoirement l'application d'une vue prédéterminée sur les données. En effet, comme il s'agit d'une extension de SQL, il est possible de sélectionner le sous-ensemble de données intéressant en utilisant les clauses `FROM.. WHERE..` de la requête.

Prétraitement - Etape 2 : Codage de l'âge. Etant donné que `MINE RULE` ne possède pas d'opérateur d'encodage prédéfini, cette opération doit être réalisée à la main par l'utilisateur.

Extraction de règles liant des produits avec l'âge du consommateur. En `MINE RULE`, on spécifie que l'on souhaite chercher des règles associant un ou plusieurs produits (corps de la règle) avec l'âge du client (tête de la règle) :

```
MINE RULE SalesRB AS
SELECT DISTINCT 1..n item AS BODY, 1..1 customer_age AS HEAD,
       SUPPORT, CONFIDENCE
FROM SalesView WHERE payment='credit_card'                    (7)
GROUP BY t_id
EXTRACTING RULES WITH SUPPORT: 0.25, CONFIDENCE: 0.5
```

Si nous souhaitons stocker les résultats dans une base de données supportant le modèle relationel, les règles extraites doivent être stockées dans une table $SalesRB(r\_id, b\_id, h\_id, sup, conf)$ où r_id, b_id, h_id sont respectivement les identifiants de règle, du corps de la règle et de la tête de la règle. Les ensembles d'items correspondant au corps et à la tête de la règle sont respectivement stockés dans des tables $SalesRB\_B(b\_id, item)$ et $SalesRB\_H(h\_id, customer\_age)$. Les Tables $SalesRB$, $SalesRB\_B$ et $SalesRB\_H$ sont représentées sur la Figure 2.2.

Rule_id	Body_id	Head_id	Support	Confidence
1	1	5	37.5%	75%
2	2	5	50%	100%
3	3	5	37.5%	66%
4	4	5	37.5%	100%

Body_id	item
1	ski_pants
2	hiking_boots
3	brown_boots
4	ski_pants
4	hinking_boots

Head_id	customer_age
5	[20,29]

FIG. 2.2 – Tables normalisées contenant les règles produites par `MINE RULE` lors de la première phase d'extraction.

Recherche d'exceptions dans les données source. Nous voulons trouver les transactions de la relation source dont les tuples violent toutes les règles contenant `ski_pants` dans leur corps. Comme les composants de règles (corps et têtes) sont stockées dans des tables relationnelles, nous utilisons une requête SQL classique pour manipuler ces ensembles d'items. La requête correspondante est la suivante :

```
SELECT * FROM SalesView AS S1 WHERE NOT EXISTS
  (SELECT * FROM SalesRB AS R1,
                SalesRB_B AS R1_B, SalesRB_H AS R1_H
   WHERE R1.b_id=R1_B.b_id AND R1.h_id=R1_H.h_id AND
   S1.customer_age=R1_H.customer_age AND S1.item=R1_B.item     (8)
   AND EXISTS(SELECT * FROM SalesRB_B AS R2_B
       WHERE R2_B.b_id=R1_B.b_id AND R2_B.item='ski_pants')
   AND NOT EXISTS
         (SELECT * FROM SalesRB_B AS R3_B
          WHERE R1_B.b_id=R3_B.b_id AND NOT EXISTS
            (SELECT * FROM SalesView AS S2
             WHERE S2.t_id=S1.t_id AND S2.item=R3_B.item)))
```

Cette requête est difficile à écrire et à comprendre. Elle a pour but de sélectionner les tuples de la relation $SalesView$ originale (renommée en `S1`), pour lesquels aucune règle contenant *ski_pants* en partie gauche ne soit valide. Ces propriétés sont vérifiées par les deux premières clauses `SELECT` imbriquées. De plus, nous voulons que ces règles soient satisfaites par des tuples correspondant à la même transaction de la table originale `S1`. En d'autres termes, qu'il n'y ait pas d'éléments du corps de la règle qui ne soit pas satisfait par un tuple de la même transaction. C'est pourquoi nous vérifions que chaque élément du corps de la règle est satisfait par un tuple de la relation $SalesView$ (renommée en `S2`) dans la même transaction que le tuple de `S1` en cours de considération. Nous en profitons pour rappeler que la notion de violation d'une règle est différente en `MINE RULE` de ce qu'elle est en `MSQL`. En effet, pour qu'une transaction viole une règle en `MINE RULE`, il faut qu'elle supporte sa partie gauche mais pas sa partie droite.

Extraction de règles entre deux ensembles d'items. C'est l'exemple classique de l'extraction de règles d'association formées de deux ensembles d'items. En utilisant `MINE RULE`, cette étape peut s'écrire comme suit :

```
MINE RULE SalesRB2 AS
SELECT DISTINCT 1..n item AS BODY, 1..n item AS HEAD,
       SUPPORT, CONFIDENCE                                    (9)
FROM Sales2
GROUP BY t_id
EXTRACTING RULES WITH SUPPORT: 0.0, CONFIDENCE: 0.9
```

Dans le cas de cet exemple jouet, les résultats coïncident avec ceux trouvés par `MSQL`.

Post-traitement - Etape 1 : Manipulation des règles. Encore une fois, comme tous les ensembles d'items correspondant à des composants de règles sont stockés dans des tables ($SalesRB\_B$, $SalesRB\_H$), on peut sélectionner les règles possédant deux items en partie gauche avec une simple requête SQL.

```
SELECT * FROM SalesRB AS R1 WHERE 2=                          (10)
  (SELECT COUNT(*) FROM SalesRB_B R2 WHERE R1.b_id=R2.b_id)
```

Post-Traitement - Etape 2 : Sélection des règles ayant un corps maximal. Nous souhaitons sélectionner les règles ayant le plus grand corps possible pour une partie droite donnée. Là encore, le fait que les composants de règles soient stockés sous forme relationnelle nous oblige à utiliser SQL, étant donné que `MINE RULE` ne possède pas d'opérateurs ensemblistes.

```
SELECT * FROM SalesRB AS R1    # We select the rules in R1
WHERE NOT EXISTS               # such that there are no
 (SELECT * FROM SalesRB AS R2  # other rules (in R2) with
  WHERE R2.h_id=R1.h_id        # the same head, a different
    AND NOT R2.b_id=R1.b_id    # body such that it has no
    AND NOT EXISTS (SELECT *   # items that do not occur in
      FROM SalesRB_B AS B1     # the body of the R1 rule
      WHERE R1.b_id=B1.b_id AND NOT EXISTS (SELECT *
           FROM SalesRB_B AS B2                               (11)
           WHERE B2.b_id=R2.b_id AND B2.item=B1.item)))
```

Cette requête assez complexe a pour but de sélectionner les règles telles qu'il n'y ait pas d'autre règle avec le même conséquent et un corps qui soit strictement inclus dans le corps de la première. Les deux sous-requêtes internes sont utilisées pour vérifier que le corps de la règle

dans R1 est un sur-ensemble du corps de la règle dans R2. Il est clair que ces requêtes de post-traitement seraient plus simples si le standard SQL-3 était adopté pour la représentation des sorties des règles.

Avantages et inconvénients de MINE RULE.

Le premier avantage de MINE RULE est qu'il a été conçu comme une extension de SQL. De plus, comme il englobe SQL, il est possible d'utiliser les mots clés classiques pour prétraiter les données, et, par exemple, sélectionner le sous-ensemble de données à traiter. Comme MSQL, le prétraitement des données est limité aux opérations pouvant être exprimées en SQL : il n'est pas possible de faire un sampling des données avant extraction, et la discrétisation doit être réalisée par l'utilisateur. Remarquons, cependant, que, en utilisant le mot clé CLUSTER BY, il est possible de spécifier sur quels sous-groupes d'un groupe les règles d'association doivent être extraites. A l'instar de MSQL, MINE RULE permet à l'utilisateur de spécifier certaines contraintes sur les règles (items appartenant au corps et à la tête, cardinalités des deux parties de la règle, taxonomie sur les items). Le lecteur intéressé est invité à lire [MPC96, MPC98] pour avoir une illustration de ces dernières capacités. Comme MSQL, MINE RULE est essentiellement conçu autour de la phase d'extraction, et il fournit un support assez faible aux autres étapes classiques d'un processus d'ECD (par exemple, le post-traitement doit être réalisé avec des requêtes SQL). Enfin, à notre connaissance, MINE RULE est l'un des rares langages à posséder une sémantique bien définie pour toutes les opérations qu'il propose [MPC98]. En effet, il est clair qu'un cadre théorique propre est un élément essentiel à la génération d'optimiseurs de requêtes performants.

2.2.5 DMQL

DMQL peut travailler sur des bases de données traditionnelles ; il peut donc prendre en entrée les relations source *Sales*, *Transactions* et *Customers* montrées sur la Figure 2.1 ou bien la vue résultant de la jointure de ces tables décrite dans la Table 2.5. Comme il a été fait pour MINE RULE, nous considérerons l'usage de la vue *SalesView* en entrée, de telle sorte que l'attention de l'utilisateur soit plus centrée sur les contraintes de la tâche d'extraction.

Pré-Traitement - Etape 1 : Sélection du sous-ensemble de données devant être fouillé. Comme MINE RULE, DMQL englobe SQL pour les manipulations relationnelles, la sélection du sous-ensemble de données intéressantes pour l'extraction (i.e. clients achetant des produits avec leur carte de crédit) peut donc être faite via l'usage de la clause WHERE dans la requête d'extraction.

Pré-Traitement - Etape 2 : Codage de l'âge. DMQL ne fournit aucune primitive pour l'encodage des données comme le fait MSQL. Cependant, il permet de définir une hiérarchie pour spécifier des plages de valeurs pour l'âge du client, comme suit :

```
define hierarchy age_hierarchy for customer_age on SalesView as
level1:{min...9}$<$level0:all
level1:{10...19}$<$level0:all                                (12)
      ...
level1:{60...69}$<$level0:all
level1:{70...max}$<$level0:all
```

Extraction de règles entre un ensemble d'items et l'âge du client. DMQL permet à l'utilisateur de spécifier des modèles (templates), appelés *metapatterns*, pour les règles à

item$^+$(X,{I})	customer_age(X,A)	Support	Confidence
item(X,ski_pants)	customer_age(X,20...29)	37.5%	75%
item(X,hiking_boots)	customer_age(X,20...29)	50%	100%
item(X,brown_boots)	customer_age(X,20...29)	37.5%	66%
item(X,ski_pants)∧item(X,hiking_boots)	customer_age(X,20...29)	37.5%	100%

TAB. 2.6 – Résultats obtenus avec `DMQL` pour la première phase d'extraction (`SalesRB`).

découvrir, en utilisant le mot clé **matching**. Ces metapatterns peuvent être utilisés pour imposer des contraintes syntaxiques relativement fortes sur les règles à découvrir. Nous pouvons donc facilement spécifier que nous cherchons des règles possédant un corps relatif à des produits achetés et une tête relative à l'âge du client. De plus, nous pouvons spécifier que nous souhaitons utiliser la hiérarchie prédéfinie pour l'attribut âge.

use database Sales_db
use hierarchy age_hierarchy **for** customer_age
mine association as SalesRB
matching with $item^+(X, \{I\}) \Rightarrow customer\_age(X, A)$ (13)
from SalesView
where payment='credit_card'
group by t_id
with support threshold=25%
with confidence threshold=50%

où le metapattern avec la notation $\{I\}$ décrit les règles possédant un ou plusieurs occurences du prédicat *item*, comme $item(X, I_1) \wedge item(X, I_2) \ldots item(X, I_j)$ où $\{I_1,\ I_2,\ \ldots\ I_j\}= I$ sont différents éléments de l'ensemble I obtenu en entrée par l'utilisation de la clause `WHERE`. Le résultat est décrit dans la Table 2.6.

Recherche d'exceptions dans les données sources. Comme `MINE RULE`, `DMQL` ne fournit aucun système pour le croisement des données et des motifs : il faut donc passer par une requête SQL comme pour `MINE RULE` (Requête (8)). Notons que la notion de violation d'une règle par une transaction est la même avec `DMQL` qu'avec `MINE RULE`.

Extraction de règles entre ensembles d'items. Cette phase est réalisée par la manipulation suivante :

use database Sales_db
mine association as SalesRB2
matching with $item^+(X, \{I\}) \Rightarrow item^+(X, \{J\})$ (14)
from Sales2
group by t_id
with confidence threshold=90%

Post-Traitement - Etape 1 : Sélection des règles possédant deux items en partie gauche. Comme `MINE RULE`, `DMQL` ne fournit aucun support aux opérations de post-traitement des règles. Comme nous n'avons pas d'accès direct aux règles et que nous ne connaissons pas la méthode exacte de stockage utilisée, nous faisons l'hypothèse que les règles sont stockées d'une manière similaire à `MINE RULE`, ce qui nous permet de comparer les langages sous les mêmes

conditions de format. Pour cette étape, une requête similaire à la numéro (10) déjà utilisée avec les exemples de MINE RULE est nécessaire.

Post-Traitement - Etape 2 : Sélection des règles avec un corps maximal. Comme MINE RULE, DMQL ne fournit aucun support aux opérations pour la manipulation des règles comme la sélection des règles les plus générales. Pour les mêmes raisons que la précédente requête, une requête SQL analogue à celle utilisée en (11) sera nécessaire.

Avantages et inconvénients de DMQL.

Comme pour MINE RULE, l'un des principaux avantages de DQML est qu'il englobe SQL, et qu'il est donc ainsi simple à appréhender pour un nouve utilisateur. De plus, DMQL est conçu pour pouvoir fonctionner sur des bases de données et datacubes classiques. En ce qui concerne la phase d'extraction, DMQL permet d'imposer de fortes contraintes syntaxiques sur les motifs à extraire, en utilisant les metapatterns qui permettent à l'utilisateur de spécifier l'aspect des règles à extraire. Un autre avantage de DMQL est qu'il est possible d'inclure certaines formes de *background knowledge* dans les processus d'extraction, par exemple en définissant des hiérarchies sur les items de la base de données, ce qui permet de rechercher des règles d'association entre les différents niveaux de la hiérarchie. Une fois les règles extraites, nous pouvons réaliser des opérations de "roll-up" et "drill-down" sur les règles extraites. Cependant, comme pour les autres langages étudiés jusqu'à présent, l'un des principaux inconvénients de DMQL est qu'il s'agit d'un langage essentiellement conçu pour la phase d'extraction. Il faut donc utiliser du SQL classique ou des outils additionnels pour réaliser les opérations de pré- et post-traitement. Finalement, il est important de souligner qu'au delà des règles d'association, DMQL peut effectuer d'autres tâches d'extraction, entre autres relatives à la classification.

2.2.6 OLE DB for DM

OLE DB for DM est conçu pour pouvoir travailler facilement avec des données relationnelles déjà accessibles via OLE DB. Ainsi, il peut directement travailler avec les données existantes et la création d'une vue n'est pas nécessaire, car la mise en forme des données dans le bon format est exactement l'un des buts de la définition et du remplissage d'un "Data Mining Model".

Pre-Traitement - Etape 1 : Sélection du sous-ensemble de données à traiter. Dans le cadre de travail proposé par OLE DB for DM, la sélection des données à traiter est faite lors de la définition du "Data Mining Model" et de son remplissage. D'un point de vue conceptuel, cela reste très similaire à la création d'une vue. Ici, le DMM est nommé [SalesRB] par analogie avec les exemples des autres langages, et est créé comme suit :

```
CREATE MINING MODEL [SalesRB](
[transaction_id] LONG KEY,
[customer_age]   LONG DISCRETIZED PREDICT,
[items]          TABLE (
   [item] TEXT KEY
   )
)
USING [My_assoc_Algo] (min_support=2, min_confidence=0.5)
```

On remarque que nous avons utilisé une table imbriquée [items] pour spécifier les produits achetés par un client dans une transaction et nous faisons référence à un algorithme d'extraction,

, `My_assoc_Algo`, pour les règles d'association. Pour insérer des données dans le "Data Mining Model", nous utilisons la commande suivante :

```
INSERT INTO [SalesRB]
([transaction_id],[customer_age],[items])
 SHAPE
 {SELECT [transaction_id],[customer_age]
    FROM Transactions,Customers
    WHERE Transactions.customer=Customers.customer_id
           AND Transactions.payment="credit_card"
  APPEND(
  {SELECT [item] FROM Sales
   ORDER BY [tr_id]}
   RELATE [transaction_id] TO [tr_id])
   AS [items]}
```

On remarque que la sélection des données source pertinentes (achats faits avec une carte de crédit) est réalisée à cette étape. Remarquons aussi que le mot clé `APPEND` construit la table imbriquée `[items]` contenant les items de la relation source `Sales` achetés au cours d'une même transaction.

Pré-Traitement - Étape 2 : Codage de l'âge. La définition du "Data Mining Model" permet la spécification d'attributs discrétisés ainsi que de la méthode de discrétisation à utiliser. Cependant, la réalisation de cette discrétisation doit être assurée par le fournisseur de l'algorithme d'ECD.

Extraction de règles entre un ensemble d'items et l'âge du client. Dans SQL Server 2000, aucun algorithme pour l'extraction de règles d'association n'est fourni, mais d'après les spécifications de `OLE DB for DM`, ce type d'algorithme peut être supporté. Nous supposerons ici que l'utilisateur possède une telle implémentation d'un algorithme nommée `My_assoc_Algo`, qui prend en entrée les seuils minimum de support et confiance et qui travaille sur le contenu de la table imbriquée `[items]` pour générer des règles d'association.

Les résultats de l'extraction de règles d'association peuvent être stockés par l'algorithme sous forme d'une table relationnelle décrite par la représentation `PMML` suivante :

```
<Item id="1" value="ski_pants" />
<Item id="2" value="hiking_boots" />
<Item id="3" value="brown_boots" />

<Itemset id="1" support="0.5" numberOfItems="1">
  <ItemRef itemRef="1">
</Itemset>
<Itemset id="2" support="0.5" numberOfItems="1">
  <ItemRef itemRef="2">
</Itemset>
<Itemset id="3" support="0.375" numberOfItems="1">
  <ItemRef itemRef="3">
</Itemset>
<Itemset id="4" support="0.375" numberOfItems="2">
```

```
     <ItemRef itemRef="1" />
     <ItemRef itemRef="2" />
</Itemset>

<Item id="4" value="[20,29]" />

<Itemset id="5" support="0.5" numberOfItems="1">
     <ItemRef itemRef="4" />
</Itemset>

<AssociationRule support="0.375" confidence="0.75"
             antecedent="1" consequent="5" />
<AssociationRule support="0.50" confidence="1.0"
             antecedent="2" consequent="5" />
<AssociationRule support="0.375" confidence="0.66"
             antecedent="3" consequent="5" />
<AssociationRule support="0.375" confidence="1.0"
             antecedent="4" consequent="5" />
```

On remarquera que la description PMML est très similaire aux structures de stockage proposé par MINE RULE.

Recherche d'exceptions dans les données source. Pour cette tâche, nous devons écrire une requête en SQL classique. Comme les règles d'association produites par l'algorithme peuvent être stockées dans le format PMML, proche du format utilisé par MINE RULE, on en déduit que la requête utilisée sera très similaire à celle utilisée avec MINE RULE.

En ce qui concerne le post-traitement, ou l'utilisation des règles après extraction, il faut noter que OLE DB for DM fournit seulement des outils pour la prédiction, notamment avec le mot clé PREDICTION JOIN. Cependant, ceci n'est pas utile ici, car nous ne considérons pas une tâche prédictive.

Extraction de règles entre deux ensembles d'items. Nous voulons réaliser une nouvelle tâche d'extraction ici, il est donc nécessaire de définir un nouveau "Data Mining Model". Celui-ci est analogue au modèle utilisé à l'étape d'extraction précédente à l'exception que l'âge du client n'est plus pertinent ici. En effet, la différence entre cette tâche d'extraction et la précédente réside dans le fait que nous cherchons des règles à l'intérieur des ensembles d'items achetés et non des règles concluant sur un attribut comme l'âge.

```
CREATE MINING MODEL [SalesRB2](
[transaction_id] LONG KEY,
[items]          TABLE (
   [item] TEXT KEY  PREDICT
   )
)
USING [My_association_Algorithm] (min_support=0, min_confidence=0.9)
```

Nous devons ensuite peupler ce modèle avec de nouvelles données

```
INSERT INTO [SalesRB2]
(
 [transaction_id],[items]
 SHAPE
 {
  SELECT [transaction_id] FROM Transactions
  WHERE Transactions.payment="credit_card"
  APPEND
  {
   SELECT [tr_id],[item] FROM Sales
    ORDER BY [tr_id]
    RELATE [transaction_id] TO [tr_id]
  }
 }
)
```

Post-Traitement - Etape 1 : Manipulation des règles. Ici encore, nous avons besoin d'accéder aux composants des règles. Etant donné que `OLE DB for DM` suggère de stocker les corps et têtes de règles en suivant le format `PMML`, nous obtiendrons une requête très similaire à celle utilisée pour `MINE RULE`.

Post-Traitement - Etape 2 : Sélection de règles avec un corps maximal. Encore une fois, étant donné que les règles sont stockées selon un format PMML, nous pouvons utiliser une requête similaire à celle utilisé pour `MINE RULE`.

Avantages et inconvénients de `OLE DB` for `DM`. Le premier avantage de `OLE DB for DM` est qu'il correspond à une première tentative de standard industriel et qu'il commence à être implémenté dans des solutions commerciales (comme SQL Server 2000). Il est conçu comme une extension de SQL et un DBA peut écrire des requêtes qui sont similaires aux requêtes SQL classiques pour définir et remplir les "Data Mining Models". Mais le principal problème est que `OLE DB for DM` n'est pas vraiment un langage pour l'ECD au même titre que les trois autres. C'est une API qui a avant tout pour but de faciliter la communication entre les bases de données relationnelles et les algorithmes d'extraction. Il peut donc travailler avec un grand nombre de ces derniers à la condition que ceux-ci soient compatibles avec les Mining Model d'`OLE DB for DM`. Cependant, il ne fournit aucun moyen simple pour gérer les contraintes typiques utilisées dans le cadre de l'extraction de règles d'association. comme celles sur les items, la fréquence ou la confiance. Plus généralement, tous ces types de contraintes doivent être donnés comme paramètres de l'algorithme d'extraction. De plus, l'accès aux résultats de la fouille de données et la navigation parmi les motifs extraits doit être gérée par le fournisseur de l'algorithme, ce qui rend difficile l'élaboration d'une méthode générale pour le post-traitement. Enfin, il n'existe pas de sémantique formelle aussi bien définie que celle de `MINE RULE`.

2.2.7 Résumé des caractéristiques

Nous avons étudié trois langages, `MSQL`, `MINE RULE` et `DMQL` et une `API` pour l'ECD, `OLE DB for DM`, utilisant un langage ressemblant à SQL.Nous avons comparé les différentes capacités de ces langages avec les propriétés souhaitées d'un langage de requêtes idéeal pour les bases de données inductives dédiées aux règles d'association. La Table 2.7 résume ces caractéristiques et

Caractéristique	MSQL	MINE RULE	DMQL	OLE DB for DM
Satisfaction de la propriété de fermeture	Oui	Oui	Oui	Oui[1]
Selection des données source	Oui	Oui	Oui	Oui
Spécification de différents types de motifs	Non[2]	Quelques uns [3]	Oui	Pas directement [1]
Spécification du "Background Knowledge"	Non	Non	En partie [4]	Non
Post-traitement des résultats générés	Oui[5]	Non	En partie [6]	En partie [7]
Spécification de contraintes	Oui	Oui	Oui	Non[8]

TAB. 2.7 – Résumé des principales caractéristiques des différents langages étudiés. ([1]Dépendant de l'algorithme. [2] seulement pour les règles d'association. [3]règles d'association et motifs séquentiels élémentaires. [4] hiérarchies de concept. [5]`SelectRules`, `Satisfy` et `Violates`. [6] Opérateurs pour la visualisation. [7]`PREDICTION JOIN` [8]paramètres de l'algorithme)

montre comment les propositions étudiées ici s'en rapprochent. Nous avons pour cela préparé un exemple benchmark et nous avons testé les différentes propositions sur celui-ci. Cet exemple jouet est en fait un hypothétique scénario d'ECD, inspiré de la pratique dans ce domaine, et dans lequel nous réalisons un certain nombre de requêtes. Nous avons testé la possibilité et la facilité pour l'utilisateur d'exprimer ces requêtes avec les langages précédemment mentionnés.

Tous les langages présentés ici vérifient la propriété de fermeture, qui est cruciale pour le développement des bases de données inductives. De plus, tous permettent la sélection des données source et nécessitent l'extraction à partir d'une base de données relationnelle des motifs d'ECD, et en particulier les règles d'association. Aucun langage ne fournit des primitives pour le sampling, mais tous permettent la manipulation de données multi-dimensionnelles (parce que c'est une caractéristique inhérente à SQL). `MSQL` ne peut néanmoins travailler que sur une relation qui est déjà dans un certain format (une table booléenne en fait). En outre, même si `MINE RULE` et `MSQL` peuvent traiter les hiérarchies si la relation `"est-un"` est encodée dans une relation séparée, `DMQL` permet de la définir explicitement et de l'utiliser pendant l'extraction de motifs. En ce qui concerne la spécification de contraintes pour l'extraction, tous les langages présentés proposent des mots clés pour spécifier au moins les valeurs de certains seuils. En revanche, pour `OLE DB for DM`, c'est au fournisseur de l'algorithme de faire en sorte qu'il soit possible de paramétrer les contraintes. De plus, si l'on compare les différents langages sur les types de motifs qu'ils permettent d'extraire, seul `DMQL` permet d'extraire d'autres types de motifs que les règles d'association. On peut toutefois remarquer, que d'une certaine manière, les différents niveaux de groupage de `MINE RULE` permettent d'extraire des motifs séquentiels élémentaires (composée de deux items). Enfin, en ce qui concerne les possibilités de post-traitement, `MSQL` est plus riche que les autres langages, il possède entre autres un opérateur dédié `SelectRules`, pour l'interrogation d'une base de règles. `DMQL` offre quelques options de visualisation. Mais, d'une manière générale,

les différents langages proposés offrent des possibilités de post-traitement assez pauvres.

D'une manière générale, le résultat est qu'aucun d'entre eux ne présente toutes les propriétés souhaitées. `MSQL` semble être celui qui offre le plus grand nombre de primitives conçues pour le post-traitement et la discrétisation efficace des données à la volée. `DMQL` permet d'extraire différents types de motifs, de définir et d'utiliser des hiérarchies sur les items, et de visualiser certains résultats. `MINE RULE` est le seul qui permette de partitionner dynamiquement la relation source grâce à l'utilisation de groupages de premier et de deuxième niveau (clusters) à partir desquels des contraintes plus sophistiquées sur les règles peuvent être définies. De plus, à l'heure actuelle, il s'agit du seul langage possédant une sémantique algébrique bien définie, ce qui important si l'on souhaite faire une analyse détaillée des problèmes d'optimisation. `OLE DB for DM` est une API, qui permet aux applications d'accéder à des sources de données par le biais de requêtes ressemblant à du SQL, et de les coupler avec des algorithmes d'ECD. La principale motivation de la conception d'`OLE DB for DM` est la simplification de la communication entre l'application, le système gérant les données et les algorithmes d'ECD disponibles. Cependant, à l'heure actuelle, il ne supporte pas les tâches d'ECD non prédictives. On remarque également que le fait qu'`OLE DB for DM` supporte ou non certaines caractéristiques est très fortement lié à l'algorithme d'extraction référencé lors de la création du "Data Mining Model".

Il faut cependant souligner, que lorsque l'on considère différents langages, il est important d'identifier précisément le type de règles descriptives qui sont extraites. Tous les langages présentés peuvent extraire des règles d'association intra-tuples, i.e. des règles qui associent les valeurs des attributs d'un tuple. Les règles d'association obtenues décrivent les propriétés communes d'un nombre suffisant de tuples de la relation. Plus précisément, seuls `DMQL` et `MINE RULE` peuvent extraire des règles d'association inter-tuples, i.e. des règles qui associent les valeurs des attributs de différents tuples et qui donc décrivent des propriétés d'un ensemble de tuples. L'utilisation de tables imbriquées dans les "Data Mining Models" de `OLE DB for DM` peuvent faciliter l'extraction de règles d'association inter-tuples par l'algorithme d'extraction. En effet, elles permettent d'inclure sur une unique ligne du modèle les caractéristiques de différents tuples des tables de données source. D'une manière générale, les règles d'association intra-tuples semblent constituer le cœur commun des capacités d'expresion des langages étudiés. La capacité du langage à pouvoir traiter les règles inter-tuples affecte la représentation des données en entrée pour le moteur d'extraction. Comme il a déjà été dit, `MSQL` considère seulement les règles d'association intra-tuples, mais cette limite peut être dépassée en changeant la représentation de la relation en entrée, c'est-à-dire en incluant les attributs pertinents des différents tuples dans un unique tuple d'une nouvelle relation. Cependant, ceci peut être compliqué et nécessiter un long travail de prétraitement. De plus, dans ces cas là, les requêtes `MSQL` qui ont la même sémantique que les requêtes équivalentes en `DMQL`et `MINE RULE`, peuvent être complexes et difficiles à comprendre.

Enfin, nous pouvons faire une dernière remarque sur les capacités des langages, sur le fait qu'alors que `DMQL` et `MINE RULE` utilisent effectivement des fonctions d'aggrégats (resp. sur les éléments des règles et sur les clusters) pour l'extraction de règles d'association, `MSQL` les fournit seulement comme outil de post-traitement sur les résultats.

Perspectives

À la suite de cette étude comparative de différents langages pour l'extraction de règles d'association, il apparaît que ceux-ci ressemblent encore trop à du sucre syntaxique autour de SQL pour l'intégration de certains algorithmes d'ECD. C'est en fait la nature des algorithmes sous-

jacents qui influe sur les caractéristiques de ces différentes propositions. Ainsi, dans le cas de `MINE RULE`, le moteur `MINE RULE` va analyser les mots clés présents dans une requêtes (en particulier ceux concernant les différents niveaux de groupage) pour choisir d'utiliser un extracteur particulier dans une famille d'extracteurs existants. De même, les différents types de motifs que peut extraire `DMQL` correspondent en fait chacun à un prototype particulier intégré dans le système `DMQL`, chargé de fouiller ce type de motifs. D'une manière générale, les propositions de langages présentées dans la section précédente ne servent souvent qu'à intégrer de manière *ad hoc* des algorithmes existants dans des SGBD. C'est ce qui explique le faible nombre de primitives pour la spécification des contraintes dans les différents langages, qui sont en fait réduites à un nombre prédéfini de "built-in". C'est également le caractère *ad hoc* de ces différents propositions de langage pour les BDI qui limite les possibilités de pré-traitement des données et de post-traitement des motifs. En effet, les langages ayant avant tout été conçus pour "englober" des algorithmes d'extraction de motifs, ils ne sont pas ciblés sur ce type d'opérations.

Du point de vue de l'utilisateur, il serait donc souhaitable que la grammaire des langages accepte un certain degré d'extensibilité et de flexibilité. En effet, il n'est actuellement pas possible pour l'utilisateur d'introduire des fonctions qu'il a définies dans les requêtes. Ceci permettrait de prendre en compte des contraintes sophistiquées sur les motifs, basées par exemple sur de nouvelles mesures d'évaluation des motifs. Ce serait alors au moteur de traitement des requêtes de la BDI de déterminer les éventuelles propriétés intéressantes de ces contraintes ainsi que la manière de les traiter. De plus, si l'on souhaite développer un langage de requêtes pour les bases de données inductives, il est clair que nous devons nous intéresser à d'autres types de motifs que les règles d'association. Une plus grande flexibilité du langage permettrait alors de spécifier une plus large variété de tâches d'extraction de manière déclarative. Cela nécessitera donc la définition de nouvelles primitives propres à de nouveaux domaines de motifs et qui pourront correspondre à de nouvelles opérations ou contraintes.

Un autre problème crucial relatif aux langages de requêtes pour la fouille de données est l'optimisation d'une séquence de requêtes (par exemple, détecter l'inclusion entre deux requêtes). Un premier travail de ce type a été réalisé dans le cadre de l'extraction de règles d'association avec MINE RULE [Meo03]. On peut ainsi mettre en évidence des propriétés d'inclusion, d'équivalence ou de dominance entre deux requêtes `MINE RULE`, qui permettent de réutiliser des résultats déjà obtenus lors d'une précédente extraction. De même, un formalisme pour l'extraction incrémentale de règles d'association à partir de contextes d'extraction spécifiant la forme des règles à extraire a été présenté dans [DGLS02]. Enfin, dans le cas de séquences de requêtes portant sur l'extraction de représentations condensées d'ensembles d'items fréquents [JB02b, GLD04], on sait qu'il est possible, sous certaines conditions, de passer d'une forme de représentations condensées à une autre (par exemples, des 0-libres [BBR03] ou des $\vee$-libres [BR03] aux clos) sans avoir à refaire l'extraction. Un cadre unificateur a été proposé dans [CG02] pour les représentations condensées d'ensembles d'items fréquents.

D'une manière générale, si l'on souhaite développer un langage pour les bases de données inductives, il est donc nécessaire d'avoir une autre approche que celle utilisée pour les langages présentés dans cet état de l'art. Il faut en effet faire une étude plus profonde des processus d'ECD afin de mettre en évidence les primitives fondamentales pour la caractérisation des tâches d'extraction sur différents types de motifs. De plus, il faut mettre au point un langage qui permette de combiner ces différentes primitives entre elles afin de rendre compte de la réalité des processus. C'est la démarche qui a été suivie dans le projet cInQ qui est décrit dans la Section suivante. Concernant les primitives de post-traitement, il est clair qu'un travail de fond du même

type doit être entrepris. En effet, l'implémentation des langages que nous avons présentés ici essaie de résoudre les problèmes de post-traitement que nous avons considérés en se reposant sur la fait qu'ils sont soit une extension d'un SGBD classique (comme `MINE RULE`), soit sur le fait qu'ils sont intégrés à un système d'ECD, qui offre une interface graphique au langage (comme `DMQL`).

2.3 L'approche du projet cInQ

Après l'analyse faite précédemment, il apparaît qu'aujourd'hui, peu de langages peuvent être considérés comme de bons candidats pour les bases de données inductives. Ils sont pour la plupart conçus comme des extensions de SQL, mais ils sont limités par le type de motifs sur lesquels ils peuvent travailler (typiquement des ensembles d'items), par le faible nombre de contraintes qu'ils peuvent gérer et enfin par le support limité des opérations de pré- et de post-traitement. D'une manière générale, les modèles proposés pour l'instant pour les bases de données inductives sont assez préliminaires et assez peu compris aujourd'hui. De plus, les relations entre ces différentes propositions restent assez floues. Là encore, cette situation est proche de celle que connaissait le domaine des bases de données au début des années 60, où l'on disposait d'un pléthore de propositions, chacune adaptée à un domaine bien précis. En fait, ce qui manque aujourd'hui au cadre des BDI est une compréhension globale des primitives fondamentales nécessaires pour incorporer l'extraction de connaissance dans les bases de données. On pourrait faire un parallèle avec la théorie de l'algèbre relationnelle de Codd, dont les propriétés simples et claires ont permis la construction d'optimiseurs génériques de requêtes relationnelles.

C'est cet état de fait qui a motivé le lancement du projet européen cInQ (IST-FET 2000-26469), qui se donnait comme objectif principal *la définition des primitives nécessaires pour interroger une base de données inductive*. Ces primitives correspondent en fait aux opérations de base et aux représentations nécessaires pour réaliser des fouilles de données efficaces. D'une manière schématique, on peut dire que le but du projet cInQ était de trouver un équivalent de la théorie de Codd pour l'algèbre relationnelle mais avec un approche fouille de données en tête au lieu d'une approche transactionnelle. Une fois ces primitives identifiées, elles pourront alors être utilisées dans un langage de requête pour les BDI.

Le projet cInQ se proposait de développer en profondeur les aspects théoriques et pratiques des Bases de Données Inductives et pour assurer la généricité de son approche, il a proposé d'étudier des extensions inductives pour différents types de motifs et d'évaluer l'applicabilité des BDI dans des domaines d'application variés (bioinformatique, télécommunications, données environnementales, etc). Pour parvenir à ses fins, un plan de travail constitué des axes suivants a été proposé :

- Le développement d'une théorie générale des bases de données inductives qui fasse abstraction des domaines de motifs considérés.
- La spécification de différents domaines de motifs (comprenant des types de motifs ainsi que des contraintes ou primitives pour les interroger) et la définition de leur sémantique, afin d'aboutir à la spécification d'un langage de requêtes inductif.
- La conception d'algorithmes et l'implémentation de prototypes pour répondre à des requêtes inductives données sur ces domaines de motifs.
- L'étude et l'évaluation des bases de données inductives sur un ensemble varié d'applications, comme la bioinformatique, l'analyse des traces d'utilisateurs sur un site web ou

encore l'exploitation de données issues du monde des télécommunications.
- La conception de méthodes d'utilisation, d'évaluation et de comparaison des bases de données inductives.

A son terme, les résultats du projet incluaient entre autres une théorie des bases de données inductive ainsi qu'un premier ensemble de prototypes pouvant être utilisés pour l'évaluation de requêtes inductives. L'ensemble des rapports et déliverables produits dans le cadre du projet cInQ sont accessibles sur le site `http ://www.cinq-project.org`. Ce projet a également permis de faire de la thématique des bases de données inductives un domaine de recherche à part entière en initiant différents workshops consacrés à ce problème (KDID'02, KDID'03, IDB'04, KDID'04).

Pour ce qui concerne les aspects concrets des BDI, le projet cInQ s'est déroulé selon trois phases majeures :

- La conception d'algorithmes. Plus précisément, nous avons spécifié précisément un ensemble de domaines de motifs, comme les ensembles d'items, les épisodes, les motifs séquentiels, les requêtes Datalog, les dépendances fonctionnelles et les dépendances d'inclusion. Ensuite nous avons défini un ensemble de fonctions d'évaluation sur ces motifs permettant de déterminer leur sémantique dans un jeu de données précis. A partir - entre autres - de ces fonctions, il a été possible de définir des primitives sous forme de contraintes qui pourront être utilisées comme des prédicats dans une requête inductive. Enfin, nous avons conçu un certain nombre d'algorithmes pour la résolution de requêtes inductives sur ces domaines de motifs. Il pouvait s'agir soit de requêtes simples (c'est-à-dire ne contenant qu'une seule contrainte "primitive" sur les motifs à extraire), soit de requêtes complexes (i.e. faisant intervenir une conjonction ou une disjonction de contraintes primitives). Dans ce dernier cas, des techniques algorithmiques évoluées doivent être mises en œuvre pour obtenir des algorithmes d'extraction qui tirent au maximum profit de ces contraintes afin de réduire l'espace de recherche parcouru et donc *in fine* d'atteindre la faisabilité des BDI.
- L'implémentation qui a aboutit au développement de nombreux prototypes basés sur les algorithmes élaborés pendant la première étape. Cette étape a également permis d'identifier plus précisément les liens entre les différents algorithmes, ce qui est utile si l'on souhaite développer des moteurs de traitement de requêtes évolués. Ainsi, dans le cas de l'extraction d'ensembles d'items fréquents, nous possédons des extracteurs capables d'extraire plusieurs types de réprésentations condensées dont certaines peuvent être dérivées à partir des autres. Dans le cadre du projet cInQ, chaque partenaire était responsable du développement de solveurs (implémentations d'algorithmes) sur un type de motifs particulier : le domaine ITEM (ensembles d'items, motifs séquentiels) pour l'INSA de Lyon, le domaine DLOG (requêtes Datalog) pour l'Université de Fribourg, le domaine EQN (équations) pour l'Institut Jozef Stefan et les extensions inductives de SQL pour l'Université de Turin. Cela n'a pas empêché quelques collaborations croisées : ainsi, l'algorithme SPIRIT-LºG développé à l'INSA relève du domaine DLOG.
- Le but de cette phase était d'évaluer l'impact de l'approche bases de données inductives sur plusieurs domaines applicatifs, et également de disséminer les résultats du projet cInQ dans la communauté scientifique. L'évaluation de l'approche BDI et des prototypes développés pendant la deuxième étape a été réalisée dans des domaines sur lesquels les partenaires avaient une certaine expérience ou pour lesquels il pouvaient facilement recourier à un expert : Web Usage Mining, télécommunications, données médicales et environnementales, bioinformatique et plus particulièrement analyse d'expression de gènes. Un autre objectif majeur de cette étape fut le développement de méthodes d'évaluation des bases de

données inductives et notamment la conception de benchmarks permettant de réaliser une appréciation objective de différentes approches BDI, aussi bien sur un plan qualitatif que sur un plan quantitatif.

Le développement du cadre théorique des bases de données inductives s'est lui déroulé tout au long du projet, parallèlement aux aspects plus pratiques des BDI. Il a essentiellement permis de déboucher sur une véritable formalisation du cadre des bases de données inductives qui sera présenté au Chapitre 3.

2.4 Un agenda de recherche pour les BDI

Le projet cInQ a permis de clairement identifier le domaine de recherche des bases de données inductives. Grâce au travail réalisé dans ce cadre, il est possible de dresser désormais un véritable agenda de recherche pour le développement concret et l'implémentation d'un système de base de données inductives. C'est ce que nous proposons ici en identifiant ces axes de développement. Pour chacun d'entre eux, nous présenterons la contribution apportée par ce travail de thèse.

2.4.1 Définition des domaines de motifs

Le but de l'extraction de connaissance dans les bases de données étant d'extraire de la connaissance sous forme de motifs, il semble indispensable d'arriver à caractériser les motifs et de les regrouper par familles de motifs. On pourra ainsi définir des domaines de motifs (Pattern Domains) possédant des définitions et des propriétés clairement identifiées. Plus précisément, la définition d'un domaine de motifs consistera en :

- La définition formelle des motifs appertenant à ce domaine. Cette définition fixera donc l'aspect syntaxique des motifs. Par exemple, dans le cas du domaine de motifs ITEM, on trouvera les ensembles d'items, les épisodes et les motifs séquentiels, entre autres. On pourra également définir des opérateurs sur les motifs, comme les relations de généralisation et de spécialisation.
- La définition des fonctions d'évaluation pouvant être rattachées à ces motifs. Celles-ci permettront de préciser la sémantique du motif. On pourra ainsi définir des fonctions qui ne portent que sur le motif (taille d'un ensemble d'items, largeur d'un motif séquentiel, etc), sur le motif et les données (fréquence d'un ensemble d'items, confiance d'une règle d'association, etc), ou encore sur plusieurs motifs simultanément (similarité de deux motifs, par exemple).
- La spécification des contraintes sur ces motifs. Elles sont généralement dérivées à partir des fonctions d'évaluation définies précédemment et permettent d'écrire des requêtes inductives complexes dans le cadre des bases de données inductives que nous présenterons au chapitre suivant. On désignera par contrainte primitive une contrainte pour laquelle il existe dans la littérature un solveur capable d'extraire tous les motifs la vérifiant. Un exemple de contrainte primitive peut être la fréquence minimale $C_{MinFreq}$, qui n'est vérifiée par un motif que si sa fréquence dans un jeu de données particulier est supérieur à un certain seuil défini par l'utilisateur. Cependant, il n'est pas indispensable que les contraintes soient directement liées à une fonction d'évaluation. On peut ainsi imaginer des contraintes d'optimisation qui ne sont vérifiées par un motif que si celui si possède la meilleure valeur selon une certaine fonction.

Dans le cadre du projet cInQ, plusieurs domaines de motifs ont été identifiés. En effet, nous avons étudié et détaillé les domaines de motifs ITEM [cc04b], DLog [cc04c], DD (Dépendances dans les données, comme les dépendances fonctionnelles et les dépendances d'inclusion) [cc02], XML [cc04e], EQN (Equations) [cc04d]. Il est important de préciser que cette étude n'est pas exhaustive et que le projet cInQ s'est essentiellement intéressée aux motifs locaux. Il est nécessaire de continuer à identifier d'autres domaines de motifs, en particulier ceux relatifs aux motifs globaux, comme les domaines "clusters" ou "Arbres de Décision".

2.4.2 Formalisation des processus ECD

Nous avons vu précédemment que pour l'instant aucun langage de requêtes pour les bases de données inductives n'existe vraiment. En effet, aucune de ces propositions n'est suffisamment générique pour permettre de modéliser complètement un processus réel d'ECD. Nous savons que l'élaboration d'un tel langage nécessite d'abord de parvenir à formaliser les processus d'ECD de manière abstraite. En effet, actuellement, quand on parle à un utilisateur de systèmes d'ECD d'un processus d'ECD, celui-ci se le représente généralement comme l'utilisation d'une série d'outils particuliers (et souvent conçus pour l'occasion) sur des données particulières. Il faut donc parvenir à s'abtraire de cette conception *ad hoc* de l'ECD comme cela a été fait dans le monde de l'algèbre relationnelle. Pour cela, il est nécessaire de formaliser les étapes de ces processus avec un langage basé sur des primitives de haut niveau. Une fois qu'une telle formalisation aura été mise en œuvre, il sera possible de mettre en évidence certaines parties de processus qui peuvent être transférées à d'autres personnes spécialisées dans certains traitements. De plus, une fois les requêtes vues comme des objets formels sur lesquels on peut raisonner, il sera possible de concevoir des techniques de compilation et d'optimisation sur les requêtes inductives, avec l'élaboration de véritables plans d'exécution des requêtes.

Ce que nos proposons dans ce mémoire, c'est d'utiliser le cadre théorique des bases de données inductives présenté par le projet cInQ [cc04a] pour décrire ces processus. Plus précisément, nous proposons un langage simple à base de contraintes pour la description des processus d'ECD. Nous avons étoffé ce langage afin qu'il puisse prendre en compte les requêtes portant sur plusieurs types de motifs et également un plus grand nombre de contraintes. D'une manière plus précise, nous proposons d'utiliser la notion de scénario d'ECD comme le concept fondateur de l'abstraction de ces processus. D'un point de vue pratique, un scénario est un ensemble de requêtes inductives formelles décrivant une série d'opérations sur les données. Dans le Chapitre 3, nous détaillerons plus précisément ce que sont les scénarios prototypiques et nous donnerons quelques exemples. L'idée globale d'un scénario prototypique étant d'abstraire les manipulations de l'utilisateur afin de les décrire de manière formelle et non ambigüe. Nous obtiendrons ainsi des objets (requêtes) sur lesquels il sera possible de raisonner.

2.4.3 Développement des techniques de pré-traitement

La proposition d'API `OLE DB for DM` de Microsoft met en avant un concept intéressant de ce que doit être une base de donnée inductive : le "Data Mining Model". L'idée est simple et part du principe que même dans le cadre d'une BDI, il est indispensable qu'avant d'appliquer un algorithme, le jeu de données ait un format adapté à l'exécution de celui-ci. Or, bien souvent, les données stockées dans une base de donnée ont un format et une structure complètement différents. Il est donc important d'associer à chaque algorithme, ou mieux, à chaque type de motif

à extraire, un schéma type de contexte d'extraction. De plus, nous savons que la construction de jeux de données avant l'utilisation d'un extracteur peut être longue. En effet, en plus des opérations de discrétisation, de sélection et de jointure, on peut éventuellement être amené à réaliser des manipulations statistiques complexes comme du sampling ou du nettoyage de données. Des expériences réelles montrent que le temps passé sur la phase de prétraitement peut constituer plus de la moitié du temps correspondant à l'intégralité du processus d'ECD.

Le développement d'un prototype de base de données inductive pose donc forcément la question de l'intégration des opérations de prétraitement. Cela peut se faire en offrant un certain nombre d'opérateurs préconçus ("built-in") mais aussi en prévoyant des fonctionnalités qui soient fortement paramétrables (discrétisation par exemple). Initialement, le projet cInQ contenait un petit volet consacré au prétraitement, qui fut finalement abandonné. En effet, il faut signaler que sur cette thématique, un travail très conséquent a été réalisé quasi-simultanément dans le cadre du projet européen Mining Mart ("Enabling End-User Data warehouse Mining", IST-FET 1999-11993), piloté par le Professeur Katharina Morik à l'Université de Dortmund en Allemagne[1]. Ce projet partait de la constatation que les utilisateurs expérimentés connaissaient bien le type de modification à appliquer à des données pour les rendre exploitables par des techniques d'ECD. Il se proposait donc de constituer une base de cas des meilleures "chaines d'opérations de prétraitement" afin de pouvoir facilement les transposer à d'autres exemples chez de nouveaux utilisateurs. Pour cela, il a fallu décrire les données à un niveau élevé d'abstraction. Ainsi, Mining-Mart propose un meta-langage pour la description des données et des opérateurs de prétraitement (jointure, segmentation, sélection, etc), qui permet de s'abstraire des domaines d'application pour concevoir des "patterns" génériques de prétraitement.

2.4.4 Développement de solveurs

Par définition, le concept de Base de Données Inductive repose sur l'existence d'un certain nombre de solveurs (algorithmes) pour extraire des motifs relevant de divers domaines. Ces solveurs doivent prendre en compte, non seulement des contraintes primitives, mais aussi des requêtes complexes faites de conjonctions ou de disjonctions de contraintes primitives. Les contraintes définies sur les motifs lors de l'extraction constituent en effet la meilleure manière de spécifier et d'exploiter la connaissance a priori dont l'utilisateur dispose sur ce qu'il souhaite trouver. De plus, si on se place dans la logique de la conception à terme de moteurs de traitement des requêtes complexes, il devient indispensable de disposer soit d'une assez large palette de solveurs, soit d'un solveur générique. En effet, seule l'existence de plusieurs d'entre eux permettra d'envisager plusieurs solutions de compilation pour une même requête.

Au delà du fait de disposer de solveurs prenant en compte certaines contraintes particulières, il est aussi nécessaire de disposer de solveurs génériques pour certains ensembles de contraintes partageant des propriétés particulières. Ainsi, pour la famille des contraintes anti-monotones (comme la fréquence minimale), les algorithmes par niveau comme A Priori [AS94] peuvent s'appliquer. En outre, on sait mettre au point des solveurs génériques pour les contraintes de la forme $C_m \wedge C_{am}$ (où C_m est monotone, et C_{am} est anti-monotone) [JB02a, BJ05, GLD04]. Il est pertinent de s'intéresser aux conjonctions ou disjonctions de certaines contraintes pour mettre en place des techniques génériques qui permettent d'optimiser les extractions en tirant au mieux partie des propriétés des contraintes concernées [BGMP03b, BGMP03a, BGKW03, FR04, LR05]. Une fois ces différents solveurs génériques conçus, il faudra identifier les relations

[1] `http ://www-ai.cs.uni-dortmund.de/MMWEB/index.html`

qui les caractérisent. Dans le cas des extracteurs de représentations condensées par exemple, on sait qu'il est possible de régénérer les ensembles clos à partir des ensembles libres [BBR03].

2.4.5 Développement de techniques de post-traitement

Dans le contexte d'un processus d'ECD réaliste, il est difficile de penser que les résultats obtenus après l'étape de fouille de données soient directement exploitables par l'utilisateur. Ceci est particulièrement vrai dans le cas de la recherche d'ensembles d'items fréquents où l'utilisateur se retrouve souvent face à une très grande quantité de résultats, pas forcément tous pertinents. Mais ce problème peut aussi se produire lors de l'extraction de motifs globaux. Dans le cadre de l'extraction d'arbres de décision, l'arbre peut nécessiter certains élagages, par exemple, et il peut même être nécessaire de relancer une nouvelle extraction avec de nouveaux paramètres pour l'algorithme. D'une manière générale, il faut donc que les motifs obtenus après l'extraction soient non seulement accessibles à l'utilisateur, mais aussi stockés dans un format qui facilite les opérations de post-traitement pour l'utilisateur. La nature de ces opérations sera bien sûr dépendante du type de motifs. On aura ainsi besoin d'opérateurs ensemblistes pour les ensembles d'items, d'opérateurs sur les arbres pour les arbres de décision, ou encore d'opérateurs logiques pour les requêtes DATALOG. Mettre au point des structures de données qui, dans les cadres des bases de données inductives, permettent de faciliter le stockage et la manipulation des motifs, reste un domaine ouvert.

De plus, comme il a été dit dans l'étude sur les langages de requêtes pour l'extraction de règles d'association faite précédemment, le support des opérations de post-traitement par ces langages reste faible. Hormis les mots clés `SATISFY` et `VIOLATES` proposés par `MSQL`, l'utilisateur doit souvent composer avec des langages standards pas forcément confortables pour réaliser des opérations ensemblistes par exemple. Dans le cadre des règles d'association, un langage uniquement dédié au post-traitement de règles d'association a été proposé dans [TL02]. Ce langage, qui se définit comme un extension de SQL, propose notamment des opérateurs pour accéder aux parties gauches et droites des règles d'association, ainsi que des opérateurs ensemblistes (comme $\subseteq$,$\supseteq$, $\bigcup$, etc) afin de faciliter l'écriture de certaines requêtes de post-traitement. Il devient alors très simple de spécifier le calcul d'une couverture ou bien la recherche de règles ayant un corps le plus grand possible pour une partie droite donnée (comme c'était le cas dans notre exemple comparatif). Néanmoins, l'écriture d'opérateurs de post-traitement pour d'autres motifs reste aussi un problème ouvert.

2.4.6 Méthodes d'évaluation : des scénarios benchmark

Il est clair qu'une fois que plusieurs implémentations de Bases de Données Inductives auront été réalisées, il sera nécessaire de disposer d'une base commune pour évaluer ces différentes solutions. Si on trace un parallèle avec le développement des bases de données relationnelles, on remarque que l'existence de benchmarks a permis de comparer sur une base commune différentes implémentations de SQL et de techniques d'optimisation de requêtes. Les benchmarks TPC[2] par exemple sont désormais reconnus comme un moyen de comparaison standard pour les bases de données relationnelles. Ils sont disponibles en différentes déclinaisons, chacune étant plutôt orientée vers un type d'usage particulier (transactions concurrentes, manipulations OLAP, etc). D'autre part, si l'on regarde les benchmarks disponibles dans la communauté d'Apprentissage

[2] `http ://www.tpc.org`

Automatique, on trouve souvent des fichiers plats auxquels sont assignés des tâches de classification (arbres de décision, réseaux de neurones, ...). Il est clair qu'aucun de ces types de benchmarks n'est suffisant à lui seul pour évaluer une implémentation de BDI. Nous ne pouvons pas en effet prendre un jeu de données et demander une réponse à une question donnée de classification, car, par définition, les processus d'ECD sont interactifs et itératifs, et peuvent concerner différents domaines de motifs et techniques d'extraction. Il est donc clair qu'il manque aujourd'hui à la communauté de l'ECD une base commune pour l'élaboration de benchmarks pour les BDI. Une première contribution à cette problématique est le travail réalisé dans le cadre du projet FIMI (Frequent Itemset Mining Implementations Repository[3]), où il est proposé de comparer différents extracteurs d'ensembles d'items fréquents sur un jeu de données commun. Il est clair que cette expérience, limitée à un type de motifs, doit être étendue à d'autres motifs et faire appel à des traitements plus complexes si l'on veut la transposer dans le cadre des BDI. Dans ce mémoire, nous proposons d'utiliser le concept de scénario d'ECD pour la spécification de benchmarks pour les BDI. Un scénario benchmark aura ainsi l'aspect d'une série de requêtes complexes, écrites dans un langage à base de contraintes, qui sera décrit au chapitre 3.

2.5 Synthèse

Dans ce chapitre, nous avons présenté la notion de base de données inductives telle qu'elle a été proposée par Imielinski et Mannila dans [IM96]. Nous avons précisé ce qu'était une requête inductive en mettant en évidence le lien avec le calcul des théories. Puis, nous avons soulevé un certain nombre de problématiques scientifiques liées à la notion de BDI. Parmi celles-ci, la définition d'un langage de requête pour les bases de données inductives est un problème crucial. Après une analyse comparative de différents langages candidats pour les BDI restreintes au seul cas des règles d'association, nous avons pu voir qu'aucune proposition ne convient vraiment pour être considérée comme un possible langage pour les BDI. Le support des opérations de pré- et post-traitement est souvent limité, et ces langages reposent sur un nombre trop limité de solveurs. De plus, un vrai Système de Gestion de Bases de Données Inductives doit aller au delà de la simple interface entre l'utilisateur, les données et les motifs, et incorporer des systèmes de traitement de requêtes complexes. C'est cela qui a motivé la proposition du projet cInQ, qui se donnait pour but le développement du cadre des Bases de Données Inductives et l'identification des primitives fondamentales d'un langage pour les BDI. Pour cela, un plan de travail a été proposé et un agenda de recherche a été défini afin d'aider au développement futur de prototypes de bases de données inductives. Dans ce mémoire, nous proposons un certain nombre de contributions qui se situent à différents niveaux de cet agenda de recherche.

Tout d'abord dans le Chapitre 3, nous présenterons une formalisation simple du cadre des bases de données inductives issu du projet cInQ [cc04a]. Nous présenterons également un premier langage à base de contraintes pour l'écriture de requêtes inductives complexes. Ce langage est une extension de celui proposé dans [cc04a] afin de permettre entre autres d'écrire des requêtes pouvant porter sur plusieurs motifs simultanément, et nous permettra de spécifier nos premiers scénarios prototypiques. Ceux-ci permettront notamment de mettre en évidence des conjonctions de contraintes intéressantes, dont certaines seront étudiées par la suite. Ce chapitre concernera donc plutôt les points 1 et 2 de l'agenda de recherche.

Ensuite, dans le Chapitre 4 relatif au point 4 de l'agenda, nous présenterons deux solveurs

[3] `http ://fimi.cs.helsinki.fi`

pour l'extraction de deux types de motifs séquentiels différents. L'un portera sur l'extraction de motifs séquentiels logiques fréquents avec une contrainte syntaxique, et l'autre sur l'extraction de motifs séquentiels composés d'items qui vérifient une contrainte de fréquence minimale et une contrainte de similarité par rapport à un motif de référence donné par l'utilisateur.

Puis, dans le Chapitre 5, nous utiliserons notre approche scénario pour l'appliquer à la mise au point de benchmarks pour les BDI. À partir d'un exemple tiré de la pratique en bioinformatique, nous proposerons un premier benchmark mettant en jeu des prototypes développés dans le cadre du projet cInQ. Nous proposerons deux exécutions possibles pour ce benchmark. Enfin, comme l'une des opérations de ce benchmark concernera des manipulations ensemblistes sur les ensembles d'items, et que ces opérations sont rarement bien supportées par les SGBD relationnels, nous proposerons une nouvelle structure de données pour optimiser les requêtes de recherche de surensembles et terminer le traitement de notre benchmark. Ce Chapitre 5 concernera donc les points 5 et 6 de notre agenda de recherche.

Chapitre 3

Formalisation des processus d'ECD : Vers des scénarios qualitatifs

Dans ce chapitre, nous présenterons le cadre théorique des bases de données inductives tel qu'il a été défini dans le projet cInQ [Rae03, cc04a]. Cela nous permettra de définir précisément ce que sont les domaines de motifs, les fonctions d'évaluation sur les motifs et les contraintes primitives qui en sont dérivées. Cela permettra notamment de définir un langage pour la spécification des tâches d'extraction en ECD. Nous étendrons ce langage afin de permettre l'écriture de requêtes portant sur des motifs de natures différentes. De plus, nous pourrons définir de manière formelle la notion de contexte d'extraction en ECD, ce qui permettra de clarifier le type de données utilisées lors de la description d'expérience. Une fois ce langage formel pour les BDI présenté, nous pourrons alors introduire l'un des concepts nouveaux importants de cette thèse : le concept de scénario, i.e. de séquences formelles de requêtes inductives permettant de décrire des processus d'ECD. Nous verrons alors deux déclinaisons possibles des scénarios : les scénarios prototypiques, principalement orientés vers la description de tâches et le transfert de compétence, et les scénarios d'évaluation, tournés vers le benchmarking qualitatif et quantitatif des solutions BDI.

3.1 Le cadre des Bases de Données Inductives

3.1.1 Définition formelle du cadre BDI

Une base de données inductive I est composée d'une partie données $\mathcal{D}$ et d'une partie motifs $\mathcal{P}$, et peut s'écrire sous la forme $I(\mathcal{D}, \mathcal{P})$ [Rae03]. Les deux parties sont en fait des ensembles d'ensembles et peuvent être manipulées de manière duale, étant donné que les processus d'ECD impliquent des manipulations aussi bien sur les données que sur les motifs. Dans ce cadre de travail, une tâche d'extraction peut être vue comme un processus d'interrogation particulier. Plus précisément, une tâche d'ECD peut être définie comme suit : étant donnés un langage $\mathcal{L}$ de motifs, une base de données $\mathcal{D}$ et un prédicat de sélection q, rechercher des motifs revient à trouver la théorie $Th(\mathcal{L}, q, \mathcal{D}) = \{\phi \in \mathcal{L} \mid q(\phi, \mathcal{D}) \text{ est vrai}\}$. Cependant, cette définition peut sembler un peu restreinte. En effet, lorsqu'un utilisateur réalise une extraction de motifs, il veut souvent aussi connaître la valeur d'une fonction d'évaluation (par exemple la fréquence dans les données) pour les différents motifs de manière à selectionner les plus intéressants. C'est pour

cela que dans [Jeu02, BJ05] a été proposée la notion de requête étendue.

Définition 1 *Une requête inductive étendue σ est un quintuplet $(\mathcal{L}, q, \mathcal{D}, e, F)$, où $\mathcal{L}$ est un langage de motifs, q un prédicat de sélection, $\mathcal{D}$ la base de données et e une fonction d'évaluation à valeurs dans F. Le résultat d'une requête inductive étendue est défini par la collection :*

$$Res(\mathcal{L}, q, \mathcal{D}, e, F) = Res(\sigma) = \{(\phi, e(\phi, \mathcal{D})) \in \mathcal{L} \times F \mid q(\phi, \mathcal{D} = vrai\}$$

et la théorie de σ est définie par

$$Th(\mathcal{L}, q, \mathcal{D}) = \{\phi \in \mathcal{L} \mid q(\phi, \mathcal{D}) = vrai\}$$

3.1.2 Un premier langage pour les BDI

Une première ébauche de langage pour les BDI a été présentée dans [Rae03]. Elle fournit trois types d'opérations de base symétriques pour les parties données et motifs d'une BDI.

- **create** data (ou pattern) **set** (ou **(view)**) **as** : permet de créer un ensemble de données ou de motifs. Il n'est pas nécessairement matérialisé, comme les vues en `SQL`.
- **delete** data (ou pattern) **set** : permet d'effacer un ensemble de données ou de motifs.
- **update** data (ou pattern) **set** : permet de mettre à jour un ensemble de données ou de motifs.

Lors de la manipulation des ensembles de motifs, ce langage pour les bases de données inductives propose d'utiliser des contraintes pour la description des tâches d'extraction. Il fournit également des relations de spécialisation et généralisation sur les motifs ($\preccurlyeq$ et $\succcurlyeq$) ainsi que des opérateurs ensemblistes ($\cup$, $\cap$, $\subseteq$). D'une manière générale, les primitives de ce langage sont de quatre types :

- les fonctions d'évaluation : qui retournent des informations sur la sémantique du motif dans un contexte d'extraction (e.g. la fréquence d'un ensemble d'items).
- les contraintes qui sont des fonctions booléennes sur leurs arguments. Celles-ci sont basées sur des fonctions d'évaluation et peuvent être de différents types : satisfaction d'un seuil minimal ou maximal pour les valeurs de ces fonctions, appartenance aux n motifs ayant les valeurs les plus élevées pour une certaine fonction d'évaluation (contrainte d'optimisation), inclusion d'un item dans un motif, etc. La définition d'une contrainte peut être dépendante du type de motifs utilisé. Il peut ainsi exister des contraintes d'inclusion sur tous les types de motifs alors qu'une contrainte de confiance ne pourra être définie que pour les règles. On désignera par contrainte primitive une contrainte pour laquelle nous disposons d'un algorithme d'extraction capable d'exploiter de manière efficace les propriétés de celles-ci lors du parcours de l'espace de recherche. De telles propriétés pourront être la monotonie, l'anti-monotonie, le caractère convertible, etc. On appellera contrainte complexe une conjonction ou une disjonction de contraintes primitives.
- les opérateurs logiques : comme $\vee$,$\wedge$ et $\neg$.
- les opérateurs de croisement entre les données et les motifs (opération également appelée "crossing-over" dans le domaine de l'ECD), qui permettent, par exemple, de récupérer les transactions d'une table relationnelle qui contiennent certains ensembles d'items..

Dans la suite, nous allons définir formellement le domaine ITEM et préciser quels types de motifs il recouvre, puis nous introduirons la notion de contexte d'extraction pour le domaine ITEM. Après cela, nous décrirons les fonctions d'évaluation et les contraintes typiques que l'on

peut définir sur le domaine de motifs ITEM. Remarquons que ces listes sont très loin d'être exhaustives et peuvent être aisément étendues à l'aide des nombreuses autres fonctions d'évaluation et contraintes présentées dans la littérature concernant les algorithmes d'extraction en ECD. A ce titre, remarquons que, dans [Bou04], une présentation du domaine ITEM en suivant l'approche cInQ (i.e. définition du domaine de motifs, des fonctions d'évaluation, des contraintes, des requêtes inductives et présentation des solveurs) est proposée. Nous montrerons ensuite comment étendre le langage avec notamment des primitives nécessaires pour les opérations de croisement entre données et motifs ("crossing-over") et nous donnerons quelques exemple d'écriture de requêtes à l'aide de ce langage. Enfin, nous présenterons notre concept de scénario d'ECD et ses deux déclinaisons : les scénarios prototypiques et les scénarios d'évaluation. Pour chaque type, nous préciserons cette notion avec plusieurs exemples inspirés de la pratique en Web Usage Mining. L'intérêt des scénarios d'évaluation sera démontré sur un exemple applicatif dans le domaine de la bioinformatique dans le Chapitre 5.

3.1.3 Domaine de motif ITEM

Dans cette partie, nous donnons les définitions formelles des types de motifs que nous utiliserons dans la suite. Nous supposons que `Items` est un ensemble fini de symboles, e.g., `Items`$= \{A, B, C, \ldots\}$. Une *transaction* t est un sous-ensemble d' `Items`. Une *base de données transactionnelle* $\mathcal{D}$ est un ensemble fini et non vide $\mathcal{D} = \{t_1, t_2, \ldots, t_n\}$ de transactions. Un événement est un couple de la forme (I, t) où $I \in$ `Items` et t est l'instant associé à l'occurence I. Une séquence est une liste ordonnée d'événements (triés sur la valeur t). Une base de séquences est un ensemble de séquences.

Exemple. Considérons un ensemble `Items` $= \{A, B, C, D\}$, la Figure 3.1 nous donne alors un exemple d'une base de transactions T, d'une séquence S et d'une base de séquence W.

$T =$

t_1	ABCD
t_2	BC
t_3	AC
t_4	AC
t_5	ABCD
t_6	ABC

$S =$ (B, 5) (A, 7) (D, 8) (C, 10) (D, 11) (A, 14)

$W =$

W_1	W_2	W_3	W_4
(B, 5)	(C, 2)	(C, 1)	(A, 4)
(A, 8)	(D, 6)	(B, 2)	(D, 9)
(D, 9)	(A, 8)	(B, 5)	(C, 11)
	(A, 9)	(D, 7)	(C, 15)
		(A, 9)	(B, 16)

FIG. 3.1 – Bases de données de transactions et de séquences sur `Items`

Ensembles d'items.

Un *ensemble d'items* (ou **itemset**) est un sous-ensemble d'`Items`. Nous dénoterons $\mathcal{I}$(`Items`) l'ensemble des ensembles d'items pouvant être générés à partir de `Items`. Une transaction t *supporte* un ensemble d'items S si et seulement si $S \subseteq t$.

Exemple. Avec les ensembles `Items` et T de la Figure 3.1, $I_1 = \{B, C\}$ et $I_2 = \{A, C, D\}$ sont des ensembles d'items. Les transactions t_1, t_2, t_5 et t_6 supportent I_1 et les transactions t_1 et t_5 supportent I_2.

Règles d'association.

Une règle d'association est dénotée $X \Rightarrow Y$ où $X \cap Y = \emptyset$ et $X \subseteq \texttt{Items}$ est le *corps* ou l'*antécédent* de la règle et $Y \subseteq \texttt{Items}$ est la *tête* ou le *conséquent* de la règle. Une transaction t *supporte* une règle $X \Rightarrow Y$ si elle supporte $X \cup Y$.

Exemple. Avec l'ensemble $\texttt{Items} = \{A, B, C, D\}$, $\{A, C\} \Rightarrow B$ est une règle d'association. Elle est supportée par les transactions t_1, t_5 et t_6.

Concepts.

Soient $\mathcal{D}$ une base de données transactionnelle et $\mathcal{M}$ la matrice booléenne associée (avec les items en colonne), on notera $\mathcal{T}$ l'ensemble des identifiants de transactions de $\mathcal{D}$ qui apparaissent en tant que lignes de $\mathcal{M}$. Un concept est un couple (G, T), où G est un ensemble de propriétés (ici des items) et T est un ensemble d'objets (ici des identifiants de transactions), et où G et T sont reliés entre eux par les opérateurs de Galois f et g [Wil82]. Nous rappelons que $f(G) = \{t \in \mathcal{T} \mid \forall g \in G, (t, g) \in \mathcal{M}\}$ et $h(T) = \{i \in \texttt{Items} \mid \forall t \in T, (t, i) \in \mathcal{M}\}$. Étant donnés un ensemble de symboles $\texttt{Items}$ et un ensemble $\mathcal{T}$ d'identifiants de transactions, nous noterons $\mathcal{K}(\texttt{Items}, \mathcal{T})$ l'ensemble des concepts que l'on peut construire sur $\texttt{Items}$ et $\mathcal{T}$.

Exemple. Prenons la base de données transactionnelle $\mathcal{D}$ de l'exemple 3.1, la matrice booléenne associée est donnée sur la Figure 3.2. On a alors que $(\{A, B, C, D\}, \{t_1, t_5\})$ est un concept.

	A	B	C	D
t_1	1	1	1	1
t_2	0	1	1	0
t_3	1	0	1	0
t_4	1	0	1	0
t_5	1	1	1	1
t_6	1	1	1	0

FIG. 3.2 – Matrice booléenne

Épisodes.

Formellement, un *épisode* est un couple $\alpha = (V, \leq)$, où V est une collection d'items, et $\leq$ est un ordre partiel sur V. Étant donnée une séquence S d'événements, un épisode $\alpha = (V, \leq)$ *apparaît* dans S s'il existe une manière de satisfaire les items de V en utilisant les événements de S de telle sorte que l'ordre partiel $\leq$ soit respecté. De manière intuitive, les épisodes consistent en des items qui ont certaines propriétés et qui apparaissent en suivant un certain ordre partiel. Si l'ordre est total, on parle d'épisodes séries, sinon, on parle d'épisodes parallèles. Nous dénoterons $\mathcal{E}(\texttt{Items})$ l'ensemble des épisodes qui peuvent être définis sur $\texttt{Items}$.

Exemple. Si on considère l'ensemble $\texttt{Items} = \{A, B, C, D\}$, alors $A \rightarrow D \rightarrow A$ est un épisode (le symbole $\rightarrow$ spécifie l'ordre d'apparition des items). Avec l'exemple de la Figure 3.1, nous pouvons dire que $A \rightarrow D \rightarrow A$ apparaît dans S.

Motifs séquentiels.

Nous utilisons la même définition d'événement que celle donnée dans le cas des épisodes. Un motif séquentiel est une liste ordonnée d'items. Il est de la forme $M = M_1 \rightarrow \ldots \rightarrow M_n$. Le symbole $\rightarrow$ dénote la relation de succession entre les items. Chaque élément $M_i, 1 \leq i \leq n$ est de le forme $M_i = \{I_1, \ldots, I_p\}$, où $I_i \in \texttt{Items}$ pour tout $i \in 1, \ldots, p$. La longueur d'un motif M est n, sa largeur est la taille du plus grand de ses élements. Si on considère une séquence d'événements en entrée $S = \{(A_1, t_1), (A_2, t_2), \ldots, (A_n, t_n)\}$, on dira qu'un motif séquentiel $M = M_1 \rightarrow \ldots \rightarrow M_p$ est inclus dans S si il existe des entiers $i_1 < \ldots < i_p$ tels que $M_1 \subseteq A_{i_1}, \ldots, M_p \subseteq A_{i_p}$ Nous dénoterons par $\mathcal{S}(\texttt{Items})$ l'ensemble des motifs séquentiels qui peuvent être définis sur l'ensemble `Items`. La différence majeure avec l'extraction d'épisodes, c'est qu'ici, nous ne considérons pas une seule séquence en entrée, mais un ensemble de séquences d'événements, chacune d'elle étant relative à un sujet particulier (par exemple, un client, une cellule, etc).

Exemple. Si on considère l'ensemble $\texttt{Items} = \{A, B, C, D\}$ et la base de séquences W donnée par la Figure 3.1, nous avons que le motif séquentiel $C \rightarrow D \rightarrow A$ est inclus dans les séquences W_2 et W_3.

3.1.4 Contextes d'extraction pour le domaine ITEM

Un contexte d'extraction pour les requêtes inductives sur les motifs du domaine ITEM est une représentation transactionnelle des données, définie par rapport à un type de motif recherché et dont le schéma est compatible avec l'exécution d'algorithmes d'extraction pour le type de motif considéré. Une requête inductive sur le domaine ITEM peut impliquer différents types de motifs du domaine ITEM. Par exemple, on peut chercher à extraire des règles d'association et des motifs séquentiels sur un contexte donné. D'une manière globale, un contexte d'extraction est similaire à un Mining Model comme défini dans [ole00] adapté au cas du domaine ITEM. Le type de contexte que nous considérerons pour les requêtes générales sur le domaine ITEM est de la forme $\mathcal{C}(tuple\_id, ind\_attr, ordering\_attr, event\_attr, A_1, \ldots, A_a)$ où :

- $\mathcal{C}.ind\_attr$ est un attribut discret.
- $\mathcal{C}.ordering\_attr$ est un attribut sur lequel existe un ordre total.
- $\mathcal{C}.event\_attr$ est un attribut discret.
- $\mathcal{C}.A_l, 1 \leq l \leq a$ sont des attributs booléens.

Remarquons qu'une telle table n'existe pas forcément dans des jeux de données réels où l'information peut être éclatée à travers différentes tables. Néanmoins, dans les problèmes d'extraction sur le domaine ITEM que nous présentons ici, il est possible de se ramener à ce type de contexte.

Prenons le cas de la bioinformatique (plus précisément de l'analyse de données d'expression de gènes) et supposons que les données soient disposées comme suit. Tout d'abord, nous avons une table *individus* décrivant les individus (ou objets) sur lesquels les expériences sont faites ; dans notre cas, il s'agira de cellules prélevées sur différentes mouches. Ensuite, nous avons une table *evenements* contenant les événements décrivant l'état biologique de l'individu au moment de la mesure. Enfin, la table *valeurs* contient les valeurs mesurées pour l'expression des gènes des différents individus lors de certains événements particuliers ayant lieu à une heure donnée. Chaque expérience de mesure correspond à l'usage d'une puce. La Table *valeurs* possède différents champs identifiant le numéro de l'expérience, le numéro de l'individu sur le-

quel la mesure a été effectuée, l'état biologique de l'individu au moment de la mesure, l'heure de la mesure, le gène mesuré, et le niveau d'expression. La table 3.1 donne un aperçu de ces données. Il est alors possible d'obtenir un contexte comme celui que nous décrivons en réalisant la jointure de ces différentes tables et en codant sous forme de propriétés booléennes les données d'expression. Plus précisément, on aura une nouvelle table où les gènes apparaîtront sous forme d'attributs, pour lesquels une valeur '1' signifiera une surexpression et une valeur '0' une absence de surexpression (sous-expression ou comportement "normal"). Un exemple de tel contexte est donné par la Table 3.2 : par comparaison avec notre définition générique de contexte d'extraction pour le domaine ITEM,nous avons $tuple\_id = exp\_id$, $ind\_attr = name$, $event\_attr = event\_id$, $ordering\_attr = time$ et $\mathcal{A}_l, 1 \leq l \leq a$ correspondent aux différents gènes GH0012, GH0013, ..., GH5068.

ind_id	*Name*	*Birth_Date*
1	Fly_001	26/08/2004
2	Fly_002	25/08/2004
3	Fly_003	30/08/2004
...	...	...
n	Fly_n	24/08/2004

Table *individus*

event_id	*Description*
1	État normal
2	Injection Produit a
3	Début division cellulaire
...	...
p	Mort

Table *evenements*

exp_id	*ind_id*	*event_id*	*time*	*gene_name*	*exp_value*
1	1	1	12h27	GH0012	24.3
2	1	1	12h27	GH5068	17.8
3	1	3	12h30	GH0012	5.1
...	...	...	...	...	...
q	n	p	13h15	GH5068	5.3

Table *valeurs*

TAB. 3.1 – Exemple de données en bioinformatique

exp_id	name	event_id	time	GH0012	GH0013	...	GH5068
1	Fly_001	1	12h27	1	0	...	1
2	Fly_001	3	12h30	0	1	...	0
...	...	...	...	...	...	...	...
q	Fly_n	p	13h15	0	1	...	1

TAB. 3.2 – Exemple de contexte d'extraction

Nous pouvons maintenant définir plus précisément les relations qui existent entre notre contexte d'extraction et les différents types de motifs que l'on souhaite trouver. Nous rappelons tout d'abord la notion de descripteurs telle qu'elle a été définie par Imielinski et Virmani dans MSQL [IV99]. Un descripteur est de la forme $(A = value)$ où A est un attribut et $value$ une valeur du domaine de A. La représentation transactionnelle nécessaire pour l'extraction d'ensembles d'items et de règles d'association peut être aisément obtenue en enlevant les attributs

number_id	name	CG_number	E01hunfertilized	E01h	E0515h	E012 h	...
1	106G11	CG9536	0,6	0,45	0,63	0,94	...
2	106G12	CG4920	0,78	1,52	1,47	1,27	...
3	119F10.2	CG4199	0,5	0,57	0,12	0,08	...
4	96B2	CG8884	0,87	0,09	0,37	0,32	...
...	...	...	...	...	...	...	...

FIG. 3.3 – Extrait du jeu de données Drosophile

ind_id, *ordering_attr*, *event_attr* de notre contexte d'extraction générique $\mathcal{C}$, et en conservant donc uniquement les attributs *tuple_id* et $\mathcal{A}_l, 1 \leq l \leq a$. Puis, comme l'algorithme de fouille extrait seulement des ensembles d'items basés sur les valeurs vraies d'une table booléenne, nous considérerons donc un ensemble `Items`$= \{(\mathcal{A}_l = 1), 1 \leq l \leq a\}$ pour l'extraction (c.à.d. les descripteurs basées sur $\mathcal{A}_l, 1 \leq l \leq a$ prenant une valeur vraie). Dans notre exemple jouet de bioinformatique, il suffit de ne conserver que les attributs relatifs à l'expression des gènes (et donc d'enlever les colonnes *name*, *event_id* et *time*) pour avoir un contexte d'extraction pour les ensembles d'items et les règles d'association.

Les séquences d'entrée nécessaires à l'extraction d'épisodes peuvent être obtenues en projetant notre contexte d'extraction général sur les attributs *tuple_id*, *ordering_attr*, et *event_attr*. Un événement est alors donné par la valeur des deux attributs *ordering_attr* et *event_attr*, et la séquence utilisée comme entrée de l'algorithme d'extraction d'épisodes sera donnée par ces deux colonnes. Si nous souhaitons obtenir une base de séquences pour l'extraction de motifs séquentiels nous avons alors juste à ajouter la colonne *ind_attr* au contexte utilisé pour l'extraction d'épisodes. Ainsi, nous aurons une information sur l'individu auquel chaque séquence d'entrée est rattachée ; le contexte pour l'extraction de motifs séquentiels sera donc de la forme (*tuple_id*, *ind_attr*, *ordering_attr*, *event_attr*). Pour les extractions d'épisodes et de motifs séquentiels, nous aurons donc à considérer un ensemble `Items` constitué de tous les descripteurs basés sur *event_attr*. Sur notre exemple dans le domaine de la bioinformatique, nous pouvons réaliser une projection de notre contexte générique en supprimant les attributs relatifs au niveau d'expression des gènes, nous aurons alors un contexte conforme à l'extraction de motifs séquentiels. Les séquences d'entrées seront alors $\{\{(1, 12h27), (3, 12h30)\}, \ldots, \{\ldots, (p, 13h15)\}\}$

Autre exemple. Nous utilisons ici le jeu de données concernant l'analyse d'expression de gènes chez la drosophile et disponible sur le site internet de l'Université de Stanford [1]. Un extrait de ce fichier est présenté sur la Figure 3.3. Nous pouvons obtenir un contexte d'extraction qui corresponde à notre définition à partir de cet exemple. On peut prendre $tuple\_id = number\_id, ordering\_attr = CG\_number, event\_attr = name$, et les attributs $A_l, 1 \leq l \leq a$ peuvent être obtenus en discrétisant les valeurs d'expression des différents gènes (E01hunfertilized, E01h, ...). Il n'y pas d'attributs *ind_attr*, nous ne pourrons donc réaliser que de l'extraction d'ensembles d'items ou d'épisodes.

[1] `http ://genome-www5.stanford.edu/cgi-bin/publication/viewPublication.pl ?pub_no=183`

3.1.5 Domaine de motifs DLOG

Nous présentons maintenant un autre domaine de motifs qui ressemble au domaine ITEM, sauf qu'ici, nous considérons des littéraux de la logique de premier ordre comme élément de base des motifs au lieu des items. Nous utilisons le cadre de travail classique de la logique du premier ordre. Nous supposons donc que nous disposons d'un ensemble $\mathcal{V}ar$ de variables et d'un ensemble $\mathcal{C}st$ de constantes tels que $\mathcal{V}ar \cap \mathcal{C}st = \emptyset$. Les variables peuvent être différenciées des constantes par le fait que leurs caractères sont en majuscules alors que les constantes sont en minuscules. $\mathcal{P}red$ est un ensemble de symboles de prédicats. Si $p \in \mathcal{P}red$ a une arité de n et $\forall i, 1 \leq i \leq n, t_i \in \mathcal{V}ar$ alors $p(t_1, t_2, \ldots, t_n)$ est appelé un *atome libre*. De même, si $p \in \mathcal{P}red$ a une arité de n et $\forall i, 1 \leq i \leq n, t_i \in \mathcal{C}st$ alors $p(t_1, t_2, \ldots, t_n)$ est appelé un *atome clos*. Enfin, nous notons V la fonction qui, étant donné un littéral l, nous renvoie la liste des variables apparaissant dans l.

Exemple. Considérons un ensemble de variables $\mathcal{V}ar = \{X, Y, Z\}$, un ensemble de constantes $\mathcal{C}st = \{a, b, c\}$ et un symbole de prédicat p d'arité 3, on a alors que $p(X, Y, Z)$ et $p(X, Z, X)$ sont des atomes libres et $p(a, b, c)$ et $p(b, b, a)$ sont des atomes clos. De même, soit l le littéral suivant $l = p(X, a, Y)$, alors $V(l) = \{X, Y\}$.

Une substitution θ est un ensemble fini de la forme $\{v_1 \mid t_1, \ldots v_n \mid t_n\}$ où $\forall i, 1 \leq i \leq n, v_i \in \mathcal{V}ar$ et $\forall j, 1 \leq j \leq n, v_i = v_j \Leftrightarrow i = j$ et $\forall k, 1 \leq k \leq n, t_k \in \mathcal{V}ar \cup \mathcal{C}st \; and \; t_k \neq v_k$. Soient t_1 et t_2 deux atomes libres, nous dirons que t_1 s'unifie avec t_2 s'il existe une substitution θ telle que $t_2 = \theta s_1$.

Exemple. Soient $t = p(X, Y, X)$ un atome libre et θ_1 la substitution suivante : $\theta_1 = \{X \mid a, Y \mid b\}$, alors l'application de θ_1 à t sera notée $\theta_1 t$ et sera égale à $p(a, b, a)$. De la même manière, soit $\theta_2 = \{X \mid c, Y \mid Z\}$, alors $\theta_2 t = p(c, Z, c)$. Enfin, prenons deux atomes libres $t_1 = p(X, Y, Z, X)$ et $t_2 = p(A, B, A, A)$, alors t_1 peut s'unifier avec t_2 grâce à la substitution $\theta = \{X \mid A, \; Z \mid A, \; Y \mid B\}$

Soient t et t' deux atomes libres, on dira que t' est un renommage de t (ou de manière équivalente que t est un renommage de t') s'il existe une substitution θ telle que $t' = \theta t$ et une substitution θ' telle que $t = \theta' t'$. On dira que θ' est le renommage réciproque de θ. On remarque, d'après la définition d'une substitution, que θ et θ' ne peuvent avoir en partie droite de leurs substitutions que des variables. De même, avec la définition d'unification que nous avons donnée, si t et t' sont équivalents à un renommage près, cela signifie que t s'unifie avec t' et que t' s'unifie avec t.

Exemple. Considérons un ensemble de variables $\mathcal{V}ar = \{V, W, X, Y, Z\}$, un ensemble de constantes $\mathcal{C}st = \{a, b, c\}$ et un symbole de prédicat p d'arité 4, prenons alors deux atomes libres $t_1 = p(X, Y, X, W)$ et $t_2 = p(Z, Y, Z, X)$. On peut alors dire que t_1 et t_2 sont équivalents à un renommage près. En effet, il suffit de prendre les substitutions $\theta_1 = \{X \mid Z, W \mid X\}$ et $\theta_2 = \{Z \mid X, X \mid W\}$, on a alors $t_2 = \theta_1 t_1$ et $t_1 = \theta_2 t_2$. En revanche, $t_3 = p(Z, Y, V, X)$ n'est pas un renommage de t_2.

Dans [DT99], les auteurs proposent une premier type de motifs sur le domaine DLOG : les requêtes DataLog. Il s'agit de transposer au cadre de la logique du premier ordre la notion

d'ensemble d'items. Une requête DataLog est alors une conjonction de littéraux ayant en paramètre aussi bien des constantes que des variables. Un exemple de requête DataLog pourrait être $p(X,Y) \wedge q(Z,R,b) \wedge r(R,a)$, où a et b sont des constantes et X, Z, R et Y des variables. Il devient alors possible de compter la fréquence d'une requête en calculant le nombre de substitutions qui la font réussir dans une base DataLog. Dehaspe et Toivonen proposent alors d'adapter les algorithmes par niveaux utilisés pour l'extraction d'ensembles d'items fréquents à l'extraction de requêtes Datalog fréquentes.

Motifs séquentiels logiques

Nous considérons en entrée un ensemble $\mathcal{D}$ de *séquences d'entrée*, où chaque séquence est une liste ordonnée d'atomes clos. Dans la suite, la liste ordonnée des éléments d'une séquence d'entrée s sera dénotée $\langle s_1\ s_2 \ldots s_n \rangle$ où s_i est le i^{eme} élement de s. La longueur $|s|$ d'une séquence est égale au nombre d'éléments qu'elle contient. Une séquence de longueur k sera appelée k-séquence.

Exemple. $\langle achete(Mike, dvd)\ offre(Mike, Anne, dvd)\ utilise(Anne, dvd)\ range(Anne, dvd) \rangle$ est un exemple de séquence d'entrée, sa longueur est 4.

Un motif séquentiel logique est une liste ordonnée d'atomes libres et est noté $\langle s_1\ s_2 \ldots s_n \rangle$. Nous noterons $\mathcal{M}(\mathcal{P}red)$ l'ensemble des motifs séquentiels logiques pouvant être construits sur l'ensemble de symboles de prédicats $\mathcal{P}red$. De plus, nous notons V la fonction qui étant donné un motif séquentiel logique s nous renvoie la liste des variables apparaissant dans ce motif. Ainsi $V(s) = \bigcup_{i=1}^{i=n} V(s_i)$. La concaténation de deux motifs séquentiels logiques s et t est notée st.

Exemple. $z = \langle achete(A, X)\ offre(A, B, X)\ utilise(B, X) \rangle$ est un exemple de motif séquentiel logique. Nous avons $V(z) = \{A, B, X\}$

Si nous considérons deux motifs séquentiels logiques $s = \langle s_1 \ldots s_n \rangle$ et $t = \langle t_1 \ldots t_m \rangle$, nous dirons que s est un *sous-motif séquentiel logique* de t (et nous noterons $s \subseteq t$) s'il existe des entiers $j_1 < j_2 < \ldots < j_n$ tels que $s = \langle t_{j_1}\ \ldots\ t_{j_n} \rangle$.

Exemple. Avec l'exemple précédent, nous pouvons dire que $\langle achete(A, X)\ utilise(B, X) \rangle$ est un sous-motif séquentiel logique de z.

Les notions que nous avons vu pour les atomes peuvent être étendues au cadre des motifs séquentiels logiques. Soient $s = \langle s_1 \ldots\ s_n \rangle$ un motif séquentiel logique, et θ une substitution, nous notons θs l'application de la substitution θ au motif séquentiel logique s et nous avons $\theta s = \langle \theta s_1\ \ldots \theta s_n \rangle$. Etant donnés deux motifs séquentiels logiques s et t, nous dirons que s s'unifie avec t s'il existe une substitution θ telle que $t = \theta s$. Les deux motifs seront dits équivalents à un renommage près s'il existe une substitution θ telle que $t = \theta s$ et une substitution θ' telle que $s = \theta' t$. θ' sera alors appelé renommage réciproque de θ. Enfin, on peut combiner des substitutions entre elles. Soient s un motif séquentiel logique et deux substitutions θ_1 et θ_2, le résultat de la combinaison $\theta_1\theta_2$ appliqué à s (noté $\theta_1\theta_2 s$) est le résultat obtenu en appliquant θ_1 au motif séquentiel logique $\theta_2 s$.

Exemple. Soient $s = \langle achete(A, X)\ donne(A, B, X)\ utilise(B, X) \rangle$ un motif séquentiel logique et $\theta = \{A \mid Mike,\ B \mid Alice,\ X \mid dvd\}$ une substitution, alors l'application de la

Identifiant	Séquence
S_1	$p(a,b) \rightarrow q(c,d,a) \rightarrow r(d,e) \rightarrow t(e,c) \rightarrow v(e,a,d)$
S_2	$p(a,b) \rightarrow t(b,d) \rightarrow r(c,e) \rightarrow v(e,b,e)$
S_3	$p(a,b) \rightarrow t(b,e) \rightarrow p(c,d) \rightarrow s(d,a) \rightarrow r(c,e) \rightarrow v(e,a,c) \rightarrow v(e,c,c)$

TAB. 3.3 – Exemple de base de séquences d'entrée pour l'extraction de motifs séquentiels logiques

substitution θ au motif séquentiel logique s est notée θs et est égale à $\langle achete(Mike, dvd)$ $donne(Mike, Alice, dvd)$ $utilise(Alice, dvd)\rangle$. Considérons deux substitutions $\theta_1 = \{A \mid Z, B \mid Y, X \mid C\}$ et $\theta_2 = \{Z \mid N, Y \mid N\}$, alors nous avons que $\theta_2\theta_1 s = \theta_2\langle achete(Z, C)$ $donne(Z, Y, C)$ $utilise(Y, C)\rangle = \langle achete(N, C)$ $donne(N, N, C)$ $utilise(N, C)\rangle$. Ensuite, soient $s = \langle p(X, Y)$ $q(X)$ $r(W, Y, T)\rangle$ et $t = \langle p(A, C)$ $q(A)$ $r(B, C, D)\rangle$, alors s et t sont équivalentes à un renommage près. En effet considérons $\theta = \{X \mid A, Y \mid C, W \mid B, T \mid D\}$ et $\theta' = \{A \mid X,\ B \mid W,\ C \mid Y,\ D \mid T\}$, alors $t = \theta s$ et $s = \theta' t$. Enfin, considérons un motif séquentiel logique $u = \langle p(M, N)$ $q(M)$ $r(O, N, N)\rangle$, alors s s'unifie avec u mais l'inverse n'est pas vrai.

Nous pouvons maintenant définir la relation d'inclusion entre une séquence d'entrée et un motif séquence logique. Nous dirons qu'une séquence d'entrée t *contient* un motif séquentiel logique s s'il existe une substitution θ et des entiers $j_1 < j_2 < \ldots < j_n$ tels que $\theta s_1 = t_{j_1}$, $\ldots$, $\theta s_n = t_{j_n}$. Si on note $\mathcal{D}$ la base de séquences d'entrée et que l'on note Θ l'ensemble des substitutions telles que s soit inclus dans une des séquences d'entrée, alors nous dirons que Θ fait réussir s sur $\mathcal{D}$.

Exemple. Considérons la séquence d'entrée $S = \langle achete(Mike, dvd)$ $offre(Mike, Anne, dvd)$ $utilise(Anne, dvd)$ $range(Anne, dvd)\rangle$ ainsi que le motif séquentiel logique $s = \langle achete(A, X)$ $utilise(B, X)\rangle$, alors s est inclus dans S par la substitution $\theta = \{A \mid Mike, B \mid Anne, X \mid dvd\}$.

Dans le cas de l'extraction de motifs séquentiels logiques, nous travaillerons avec des bases de séquences en entrée, c'est-à-dire des ensembles de séquences. La Figure 3.3 donne un exemple d'une telle base (nommons la $\mathcal{D}$). Considérons le motif séquentiel logique $s = \langle p(X, A)$ $r(Y, B)$ $v(C, X, Y)\rangle$. Alors s est inclus dans les séquences S_1 (en utilisant la substitution $\theta_1 = \{X|a,\ Y|d,\ A|b,\ B|e,\ C|e\}$ et S_3 (en utilisant la substitution $\theta_3 = \{X|a,\ Y|c,\ A|b,\ B|e,\ C|e\}$ ou bien $\theta_3 = \{X|c, Y|c, A|d, B|e, C|e\}$). $\Theta = \{\theta_1, \theta_3\}$ fait réussir s sur $\mathcal{D}$.

3.1.6 Fonctions d'évaluation

La forme générale d'une fonction d'évaluation qui calcule une certaine valeur pour un motif donné γ est $f_I(\gamma)$. f est le nom de la fonction et I un ensemble de paramètres relatifs à f. Par exemple, dans le cas de la fonction $Frequency$, l'ensemble des paramètres est $I = \mathcal{D}$, où $\mathcal{D}$ est la table transactionnelle dans laquelle la fréquence est calculée. La liste suivante peut être facilement étendue avec les différentes fonctions d'évaluation présentées dans la littérature.

Ensembles d'items

Définition 2 (Support d'un ensemble d'items) *Le* support *d'un ensemble d'items s dans une table transactionnelle $\mathcal{D}$ est dénoté $Support_{\mathcal{D}}(s)$ et est définie comme l'ensemble de toutes les transactions de $\mathcal{D}$ supportant s (e.g., $Support_{\mathcal{D}}(\emptyset) = \mathcal{D}$).*

Définition 3 (Fréquence d'un ensemble d'items) *Etant donnés un ensemble d'items s et une table transactionnelle $\mathcal{D}$, la* fréquence absolue *de s dans $\mathcal{D}$ est définie par $AbsFreq_{\mathcal{D}}(s) = |Support_{\mathcal{D}}(s)|$ où $|.|$ est la cardinalité de l'ensemble. La fréquence relative de s dans $\mathcal{D}$ est $Freq_{\mathcal{D}}(s) = |Support_{\mathcal{D}}(s)|/|Support_{\mathcal{D}}(\emptyset)|$. Quand il n'y a pas d'ambiguïté par rapport au contexte, la fréquence désigne la fréquence relative.*

Définition 4 (Fermeture d'un ensemble d'items) *La fermeture d'un ensemble d'items s dans une table transactionnelle $\mathcal{D}$ (dénotée $Closure_{\mathcal{D}}(s)$) est le sur-ensemble maximal (par rapport à la relation d'inclusion) de s qui a le même support que s.*

Définition 5 (δ-fermeture d'un ensemble d'items) *Soient δ un entier et s un ensemble d'items, la δ-fermeture de s, dans une table transactionnelle $\mathcal{D}$ est dénotée $Closure_{\delta,\mathcal{D}}(s)$ et est le sur-ensemble maximal (par rapport à la relation d'inclusion) Y de s tel que pour tout item $A \in Y - s$, $|Support_{\mathcal{D}}(s \cup \{A\})|$ est plus grand que $|Support_{\mathcal{D}}(s)| - \delta$.*

Règles d'association

Définition 6 (Support d'une règle d'association) *On appelle* support *d'une règle d'association $X \Rightarrow Y$ dans une table transactionnelle $\mathcal{D}$ le support de l'ensemble d'items $X \cup Y$ dans $\mathcal{D}$. Plus formellement, $Support_{\mathcal{D}}(X \Rightarrow Y) = Support_{\mathcal{D}}(X \cup Y)$*

Définition 7 (Fréquence d'une règle d'association) *La fréquence absolue d'une règle d'association $X \Rightarrow Y$ dans une table transactionnelle $\mathcal{D}$ est donnée par $AbsFreq_{\mathcal{D}} = |Support_{\mathcal{D}}(X \Rightarrow Y)|$. La fréquence relative de $X \Rightarrow Y$ dans $\mathcal{D}$ est donnée par $Freq_{\mathcal{D}} = |Support_{\mathcal{D}}(X \Rightarrow Y)|/|Support_{\mathcal{D}}(\emptyset)|$. Quand cela est clair par rapport au contexte, le terme fréquence désigne la fréquence relative.*

Définition 8 (Confiance) *La confiance d'une règle d'association $X \Rightarrow Y$ dans une table transactionnelle $\mathcal{D}$ est la probabilité qu'une transaction de $\mathcal{D}$ supporte Y quand elle supporte déjà X. Cela revient donc à calculer une probabilité conditionnelle.*
$conf_{\mathcal{D}}(X \Rightarrow Y) = \frac{support_{\mathcal{D}}(X \Rightarrow Y)}{support_{\mathcal{D}}(X)}$

Épisodes

Formellement, étant donné un ensemble E de *types d'événements*, une séquence d'événements $S = (s, T_s, T_e)$ est une suite ordonnée d'événements event_i telle que $\text{event}_i \leq \text{event}_{i+1}$ pour tout $i = 1, \ldots, n-1$, et $T_s \leq \text{event}_i < T_e$ pour tout $i = 1, \ldots, n$. Une *fenêtre* sur une séquence d'événements S est une séquence d'événements $S_w = (w, t_s, t_e)$, où $t_s < T_e, t_e > T_s$, et w est constitué des paires $(event, t)$ de s où $t_s \leq t < t_e$. La valeur $t_e - t_s$ est appelée la *largeur* de la fenêtre et est notée W. Par définition, la première et la dernière fenêtre d'une séquence s'étendent au delà de la séquence, de telle sorte que la première fenêtre contient seulement le premier événement de la séquence, et la dernière le dernier point de la séquence.

Définition 9 (Support) *Etant donnés une largeur d'épisodes définie par l'utilisateur W et une séquence d'événements S, on définit le support d'un épisode α comme l'ensemble des fenêtres de largeur W de S dans lesquelles l'épisode α apparaît. Pour un épisode α, le support est dénoté $Support_{S,W}(\alpha)$.*

Définition 10 (Fréquence) *Etant donné un épisode α, nous définissons la fréquence de l'épisode α comme la proportion de fenêtres de S dans lesquelles l'épisode apparaît, i.e. :*

$$Freq_{S,W}(\alpha) = \frac{|Support_{S,W}(\alpha)|}{|Support_{S,W}(\emptyset)|} = \frac{|S_w \in \mathcal{W}(S,W) \mid \alpha \text{ apparaît dans } S_w|}{|\mathcal{W}(S,W)|}$$

où $\mathcal{W}(S,W)$ est l'ensemble de toutes les fenêtres S_w de la séquence S telles que la largeur de la fenêtre est W.

Règles d'épisodes

Intuitivement, les règles d'épisodes ressemblent aux règles d'association, mais elles ajoutent un aspect temporel. Si les événements satisfaisant la partie gauche de la règle apparaissent dans le bon ordre à l'intérieur de W unités de temps, alors les événements de la partie droite de la règle apparaissent aussi à l'intérieur de W unités de temps dans la partie de la séquence ordonnée par la relation $\leq$. Formellement, une règle d'épisode est une expression $\beta \Rightarrow \gamma$, où β et γ sont des épisodes tels que β est un sous-épisode de γ.

Définition 11 (Confiance) *La confiance de la règle d'épisode $\beta \Rightarrow \gamma$ peut être interprétée comme la probabilité conditionnelle que la totalité de γ apparaisse dans une fenêtre, sachant que β y apparaît aussi. Elle est définie par :*
$Conf_{S,W}(\beta \Rightarrow \gamma) = Freq_{S,W}(\gamma)/Freq_{S,W}(\beta)$

Motifs séquentiels

Définition 12 (Support) *Nous avons déjà défini la relation d'inclusion entre motif séquentiel et séquence d'entrée. Le support d'un motif séquentiel M dans un ensemble de séquences d'entrée T est l'ensemble des séquences d'entrée qui contiennent M. Il est dénoté $Support_T(M)$. La définition de la fréquence est immédiate : $Freq_T(M) = |Support_T(M)|/|T|$*

Définition 13 (Similarité d'un motif séquentiel par rapport à un autre) *Supposons que nous disposions d'un motif séquentiel de référence M', la similarité d'un motif séquentiel M par rapport à M' est donnée par $Sim(M, M')$. La fonction Sim calcule une mesure de similarité entre deux motifs. Plusieurs mesures de similarité différentes ont été proposées dans la littérature [Moe00]. Nous reviendrons plus en détail sur ce point dans le Chapitre 4.*

Motifs séquentiels logiques

Définition 14 (Fréquence d'un motif séquentiel logique) *Nous définissons la fréquence d'un motif séquentiel logique s dans un ensemble de séquences d'entrée $\mathcal{D}$ comme la proportion de séquences d'entrée de $\mathcal{D}$ qui contiennent s.*

3.1.7 Contraintes

Dans cette partie, nous présentons certaines contraintes pouvant pour la plupart être définies à partir des fonctions d'évaluation présentées précédemment. Certaines d'entre elles sont propres à un type de motifs particuliers, alors que d'autres sont communes à plusieurs types de motifs, comme les contraintes d'aggrégats ou d'optimisation. La forme générale des contraintes est

$C_{name(Q)}$ où *name* est la nom de la contrainte relative à une certaine fonction d'évaluation f_I, et Q est l'ensemble des paramètres de la contrainte. Naturellement, les paramètres de la fonction f_I sont *de facto* des paramètres de C_{name}, i.e. $I \subseteq Q$. Par exemple, la contrainte de fréquence minimale $MinFreq$, basée sur la fonction d'évaluation $Freq_{\mathcal{D}}$ et qui prend pour paramètre additionnel le seuil minimal de fréquence spécifié par l'utilisateur t sera dénotée $C_{MinFreq(\mathcal{D},t)}$. Enfin, il est important de préciser que les contraintes présentées ici se concentrent sur les motifs locaux étudiés dans le cadre du projet cInQ (ensembles d'items, règles d'association, motifs séquentiels, etc), mais que pour la généricité de l'approche, cette liste devrait être étendue avec d'autres contraintes sur d'autres types de motifs, comme les arbres de décision ou les clusters par exemple.

Ensembles d'items

Définition 15 (Support) *On dira qu'une transaction t d'une table transactionnelle $\mathcal{D}$ supporte un ensemble d'items s si chaque item de s appartient à t. Il est alors possible de définir une contrainte booléenne $C_{Satisfy(\mathcal{D})}(s)$ qui est vraie si toutes les transactions de $\mathcal{D}$ supportent s. Nous pouvons aussi définir une contrainte booléenne $C_{Contain(\mathcal{D})}(s)$ qui est vraie si au moins une transaction de $\mathcal{D}$ supporte s. Finalement, nous pouvons définir une contrainte $C_{Violate(\mathcal{D})}(s)$ qui est vraie si aucune des transactions de $\mathcal{D}$ ne supporte s.*

Définition 16 (Contraintes de fréquence) *Soient s un ensemble d'items, $\mathcal{D}$ une table transactionnelle et $t \in [0,1]$ un seuil de fréquence minimale. La contrainte de fréquence minimale $\mathcal{C}_{MinFreq(\mathcal{D},t)}$ est définie comme suit : $C_{MinFreq(\mathcal{D},t)} \equiv Freq_{\mathcal{D}}(s) \geq t$. De tels ensembles d'items sont dits fréquents. D'une manière duale, on peut définir une contrainte de fréquence maximale $\mathcal{C}_{MaxFreq(\mathcal{D},t)}$ satisfaite par les ensembles d'items non fréquents.*

Définition 17 (Fermeture) *Un ensemble d'items clos est égal à sa fermeture dans $\mathcal{D}$. Il est alors possible de définir une contrainte de fermeture $C_{Close(\mathcal{D})}(s) \equiv Closure_{\mathcal{D}}(s) = s$*

Définition 18 (Liberté) *Etant donnée une table transactionnelle $\mathcal{D}$, un ensemble d'items s est libre si aucune règle logique ne tient entre ses items, i.e. s'il n'existe pas deux ensembles d'items distincts X et Y tels que $S = X \cup Y$, $Y \neq \emptyset$ et $X \Rightarrow Y$ est une règle logique. Les contraintes associées peuvent alors être définies comme suit [BJ01] : $C_{Free(\mathcal{D})}(s) \equiv (\forall s' \subset s) \Rightarrow s \not\subseteq closure_{\mathcal{D}}(s')$. En terme de fonctions d'évaluation, s est libre si s n'est pas inclus dans le fermeture d'un des ses sous-ensembles.*

Définition 19 (δ-liberté) *Soient δ un entier et s un ensemble d'items, s est δ-libre si aucune règle d'association avec au plus δ exceptions ne tient entre ses sous-ensembles.La contrainte associée peut être définie comme suit [BJ01] : $C_{\delta-free(\mathcal{D},\delta)}(s) \equiv (\forall s' \subset s) \Rightarrow s \not\subseteq closure_{\delta,\mathcal{D}}(s')$.*

Règles d'association

Définition 20 (Contraintes sur la confiance) *Soient $\mathcal{D}$ une table transactionnelle, t un seuil minimal de confiance, et $X \Rightarrow Y$ une règle d'association, la contrainte de fréquence minimale pour les règles d'association est définie comme suit : $\mathcal{C}_{MinConf(\mathcal{D},t)(X \Rightarrow Y)} \equiv Conf_{\mathcal{D}}(X \Rightarrow Y) \geq t$. D'une manière duale, nous pouvons définir une contrainte de confiance maximale $\mathcal{C}_{MaxConf(\mathcal{D},t)}$.*

Définition 21 (Minimalité) *Étant donné un ensemble de règles d'association Σ, nous dirons qu'une règle d'association $X \Rightarrow Y$ est minimale s'il n'existe pas dans Σ une autre règle possédant la même partie droite mais une partie gauche X' telle que $X' \subseteq X$. Plus formellement : $\mathcal{C}_{Minimal(\Sigma)}(X \Rightarrow Y) \equiv \nexists (X' \Rightarrow Y) \in \Sigma, X' \subseteq X$*

Définition 22 (Contrainte d'intérêt minimal) *Etant donnés une table transactionnelle $\mathcal{D}$, un seuil t et une règle d'association $X \Rightarrow Y$, la contrainte d'intérêt minimal est définie comme suit : $\mathcal{C}_{MinInt(\mathcal{D},t)}(X \Rightarrow Y) \equiv Interest_{\mathcal{D}}(X \Rightarrow Y) \geq t$. Cette contrainte est basée sur la fonction d'intérêt d'une règle d'association, pour laquelle il existe différentes définitions [SA95, BMUT97].*

Episodes

Définition 23 (Contraintes de fréquence) *Comme pour les ensembles d'items, on peut définir des contraintes de fréquence minimale et maximale pour les épisodes. Ces contraintes sont dénotées : $\mathcal{C}_{MinFreq(\mathcal{D},min\_t)}$ et $\mathcal{C}_{MaxFreq(\mathcal{D},max\_t)}$.*

Règles d'épisodes

Définition 24 (Contraintes de confiance) *Comme pour les règles d'association, on peut définir des contraintes de confiance minimale et maximale. Elles sont dénotées $\mathcal{C}_{MinConf(\mathcal{D},min\_t)}$ et $\mathcal{C}_{MaxConf(\mathcal{D},max\_t)}$.*

Motifs séquentiels

Définition 25 (Contraintes de fréquence) *Les contraintes de fréquences déjà présentées pour les ensembles d'items peuvent être facilement adaptées au cadre des motifs séquentiels. Ainsi, nous disposons de contraintes $\mathcal{C}_{MinFreq(\mathcal{D},min\_t)}$ et $\mathcal{C}_{MaxFreq(\mathcal{D},max\_t)}$ pour ce type de motifs.*

Définition 26 (Contrainte de similarité) *Grâce à la fonction de similarité Sim déjà présentée, il est possible de définir une contrainte de similarité $\mathcal{C}_{Min(M')}$ par rapport à un motif séquentiel M'.*

Définition 27 (Contrainte d'expression rationnelle) *Etant donnée une expression rationnelle e spécifiée par l'utilisateur, nous dirons qu'un motif séquentiel M de largeur 1 satisfait une contrainte d'expression régulière $\mathcal{C}_{ER(e)}$ si la chaîne obtenue en prenant les éléments de M dans l'ordre correspond à une chaîne du langage décrit par e. Ce type de contrainte est utilisée dans la famille d'algorithmes SPIRIT [GRS02].*

Motifs séquentiels logiques

Définition 28 (Contraintes de fréquence) *Les contraintes de fréquence déjà présentées pour les ensembles d'items et les motifs séquentiels peuvent être facilement adaptées au cadre des motifs séquentiels logiques étant donnée la définition de l'inclusion d'un motif séquentiel logique dans une séquence d'entrée. Ainsi, nous disposons de contraintes $\mathcal{C}_{MinFreq(\mathcal{D},min\_t)}$ et $\mathcal{C}_{MaxFreq(\mathcal{D},max\_t)}$ pour ce type de motifs, où t est un seuil de fréquence précisé par l'utilisateur..*

Définition 29 (Maximalité d'un motif dans un ensemble de motifs) *Etant donné un ensemble de motifs séquentiels logiques $\mathcal{S}$ et un motif séquentiel logique $s \in \mathcal{S}$, nous dirons que s vérifie la contrainte de maximalité $\mathcal{C}_{Max(\mathcal{S})}$ s'il n'existe aucune séquence $t \in \mathcal{S} - \{s\}$ telle que s est un sous-motif séquentiel logique de t. On dira aussi que s est maximal dans $\mathcal{S}$.*

Définition 30 (Contrainte d'expression rationnelle) *Nous souhaitons adapter la contrainte syntaxique sous forme d'expression rationnelle utilisée dans [GRS02] au cadre de l'extraction de motifs séquentiels logiques. Nous considérons une expression rationnelle $\mathcal{R}$ définie par l'utilisateur (et construite sur l'alphabet des symboles de prédicats $Pred$). Nous dirons qu'un motif séquentiel logique s de largeur 1 satisfait la contrainte d'expression régulière $\mathcal{C}_{ER(\mathcal{R})}$ si la chaîne obtenue en prenant dans l'ordre les symboles de prédicats sur lesquels sont construits les atomes de s correspond à une chaîne du langage décrit par $\mathcal{R}$.*

Contraintes communes

Contraintes d'optimisation. Parmi les contraintes basées sur les fonctions d'évaluation, nous pouvons citer les contraintes d'optimisation. Étant donnés une fonction d'évaluation f_I, un entier N un motif ϕ et un ensemble de motifs Σ, la contrainte d'optimisation $C_{OptMost(f_I,N,\Sigma)}(\phi)$ est vraie seulement si ϕ a l'une des plus grandes valeurs possibles selon la fonction f_I. De manière duale, nous pouvons définir une contrainte $C_{OptLeast(f_I,N,\Sigma)}$ qui est vraie pour les motifs ayant l'une des N plus faibles valeurs selon la fonction f_I.

Contraintes d'aggrégats. Il est aussi possible de définir des contraintes qui sont basées sur des fonctions d'aggrégats. Un exemple typique de ce genre de contraintes peut être trouvé dans le problème classique dit de "l'analyse du panier", où l'on cherche des ensembles de produits fréquemment achetés par le client. On peut ainsi souhaiter extraire des ensembles d'items pour lesquels la somme des prix est plus grande qu'un certain seuil t ($Avg(Price(s)) \geq t$). Étant donnés les aggrégats fournis par les langages standards, on peut définir relativement facilement de nouvelles contraintes liées à ces aggrégats. Néanmoins, l'évaluation et l'exploitation active de telles contraintes restent généralement difficiles [BGKW03].

3.1.8 Propriétés des contraintes

Nous rappelons ici quelques propriétés fondamentales des contraintes qui sont utilisées de manière active lors de l'extraction de motifs par des algorithmes par niveaux [AS94, PHL01, NLHP98]. Elles peuvent en effet avoir un impact sur le processus de génération des candidats.

Contrainte anti-monotone. On dira qu'une contrainte C_{am} est anti-monotone si lorsqu'un un motif s viole C_{am}, alors tous les motifs contenant s violent aussi C_{am}. De manière plus formelle, cela signifie aussi que :
s satisfait $C_{am} \Rightarrow \forall s' \subseteq s$, s' satisfait C_{am}

Cette propriété est la plus communément utilisée dans les algorithmes du type APriori [AS94] pour l'extraction d'ensembles d'items : la contrainte de fréquence minimale $C_{MinFreq}$ est un exemple classique de contrainte anti-monotone.

Contrainte monotone. On dira qu'une contrainte C_m est monotone si et seulement si $\neg C_{am}$ est anti-monotone. Dès qu'un motif s viole C_m, alors tous les motifs s' contenu dans s violent aussi C_m. De manière plus formelle :
s satisfait $C_m \Rightarrow \forall s'$ tel que $s \subseteq s'$, s' satisfait C_m.

Cette propriété est moins connue, alors qu'on dispose pourtant de techniques efficaces pour pousser les contraintes monotones, notamment lorsqu'elles sont conjointes à des contraintes anti-monotones [Jeu02, Bon03]. La contrainte de fréquence maximale est un exemple classique de contrainte monotone [dRK01]. De nombreuses contraintes syntaxiques sont monotones, comme par exemple la présence de certains items dans les ensembles que l'on cherche à découvrir, la taille minimale des motifs, ou certaines contraintes d'aggrégat du type $Sum(s) \geq seuil$ (où s est un ensemble d'items) qui signifie que l'on cherche les motifs pour lesquels la somme d'une certaine fonction positive appliquée aux éléments du motif (par exemple le prix des articles dans le cas de l'analyse du panier) excède un certain seuil [BGMP03a].

Contrainte convertible. Les notions de monotonie et d'anti-monotonie sont souvent trop restrictives pour caractériser certaines contraintes. La notion de contraintes convertibles introduites par Pei et al. dans [PHL01] permet de relaxer ces définitions et de caractériser un plus grand nombre de contraintes dans l'optique de leur exploitation active dans l'extraction. Dans la suite, on suppose que l'on dispose d'une relation d'ordre préfixe R sur les motifs.

Une contrainte C_{cam} est **convertible anti-monotone** si pour tout motif s satisfaisant C_{cam}, tous les préfixes de s par R la satisfont aussi. Une contrainte C_{cm} est **convertible monotone** si pour tout motif s violant C_{cm}, tous les préfixes de s par R la violent aussi. Une contrainte est dite convertible si elle est convertible anti-monotone ou convertible monotone. Enfin, une contrainte est succinte si elle est soit monotone, soit anti-monotone, soit convertible.

De telles contraintes peuvent être caractérisées à l'aide de fonctions préfixes monotones dont la définition est la suivante. Etant donné un ensemble de motifs I, une fonction $f : I \rightarrow \mathbb{R}$ est une fonction préfixe monotone croissante par rapport à R si et seulement si pour tout motif s et son préfixe s' par R, nous avons $f(s') \leq f(s)$. De même, une fonction $g : I \rightarrow \mathbb{R}$ est appelée préfixe monotone décroissante par rapport à R si et seulement si pour tout motif s et son préfixe s' par R, nous avons $g(s') \geq g(s)$.

Dans [PHL01], les auteurs ont caractérisé les contraintes convertibles comme suit. Soient h un fonction d'évaluation, s un motif et v un seuil, alors une contrainte de la forme $h(s) \geq v$ (resp. $h(s) \leq v$) est convertible anti-monotone (resp. monotone) si et seulement si h est une fonction préfixe monotone décroissante. De la même manière, une contrainte de la forme $h(s) \geq v$ (resp.$h(s) \leq v$) est convertible monotone (resp. anti-monotone) si est seulement si h est préfixe croissante.

Exemple : On considère des motifs séquentiels correspondant à des suites décroissantes de nombres entiers et R la relation d'ordre préfixe. On considère la contrainte convertible anti-monotone $\mathcal{C}$ suivante : un motif satisfait $\mathcal{C}$ si la moyenne de ses éléments est supérieure à une constante c. Alors, si s' satisfait la contrainte, tous les motifs qui lui sont équivalents (i.e. ses préfixes) la satisfont aussi. Par exemple, en prenant $c = 8$ si la séquence $s' = \{14, 10, 5, 4\}$ vérifie la contrainte $avg(s) \geq c$ alors ses préfixes la vérifient aussi (on a ainsi $avg(\{14, 10, 5\}) \geq c$, $avg(\{14, 10\}) \geq c$, $avg(\{14\}) = 14 \geq c$). En revanche, $avg(\{14, 5, 4)\}) < c$.

3.1.9 Autres primitives

Opération de croisement entre données et motifs

Afin de gérer les opérations de post-traitement qui impliquent un croisement entre les données et les motifs, nous étendons le langage présenté précédemment avec les deux opérateurs suivants. Si $\mathcal{C}$ est un contexte d'extraction et P un ensemble de motifs, $\alpha(\mathcal{C}, P)$ retourne l'ensemble des entrées de $\mathcal{C}$ qui contiennent au moins un motif de P. Par exemple, si P est une collection d'ensembles d'items, l'opérateur retournera les transactions de $\mathcal{C}$ contenant au moins un ensemble d'items de P. Dans le cas des motifs séquentiels, il retournera des séquences d'entrée. L'opération duale $\beta(\mathcal{C}, P)$ retourne les éléments de P apparaissant au moins une fois dans le contexte d'extraction $\mathcal{C}$.

Exemple. Reprenons l'ensemble T donné dans l'exemple de la Figure 3.1 et considérons la collection d'ensembles d'items $R = \{\{A, D\}, \{B, C\}\}$, alors la Figure 3.4 nous donne le résultat de l'opération $\alpha(T, R)$. De même, si on considère $S = \{\{A, C\}, \{A, E\}, \{B, D\}\}$, alors $\beta(T, S) = \{\{A, E\}\}$.

$$T = \begin{array}{c|l} t_1 & \texttt{ABCD} \\ t_2 & \texttt{BC} \\ t_3 & \texttt{AC} \\ t_4 & \texttt{AC} \\ t_5 & \texttt{ABCD} \\ t_6 & \texttt{ABC} \end{array} \qquad \alpha(T, R) = \begin{array}{c|l} t_1 & \texttt{ABCD} \\ t_2 & \texttt{BC} \\ t_5 & \texttt{ABCD} \\ t_6 & \texttt{ABC} \end{array}$$

FIG. 3.4 – Exemple de croisement entre les données et les motifs

Typage des motifs et variables

Comme nous souhaitons écrire des requêtes qui peuvent impliquer différentes types de motifs, il est important de disposer de moyens pour différencier les variables qui peuvent apparaître dans une requête. Ainsi, il sera possible de spécifier dans la requête qu'une variable représente un type particulier de motifs. Si nous considérons un ensemble d'items `Items`, cela peut être fait en disant que la variable appartient à l'ensemble $\mathcal{I}(\texttt{Items}), \mathcal{P}(\texttt{Items}), \mathcal{E}(\texttt{Items})$.

Par exemple, dans la requête suivante :
$r \in \mathcal{I}(\texttt{Items}) \wedge s \in \mathcal{E}(\texttt{Items}) \wedge C_{MinFreq(\mathcal{D},t)}(r) \wedge C_{Sim(M')}(s)$

nous pouvons facilement remarquer que r est un ensemble d'items et s un épisode.

Remarquons également que le fait de typer un motif peut donner de nouvelles informations sur ses propriétés, et parfois impliquer de nouvelles contraintes. Prenons l'exemple des concepts [RBCB03] qui sont formellement représentés par un couple (G, T) où G est un ensemble d'objets et T un ensemble de propriétés. Par définition, ces deux ensembles sont liés par les opérateurs de Galois f et g, qui sont équivalents par définition à une propriété de fermeture à la fois sur les ensembles d'items et sur les ensembles d'identifiants de transactions. Ainsi, lorque nous cherchons à extraire ce genre de motifs, cela implique automatiquement deux contraintes $C_{Close(\mathcal{D})}(G)$ et $C_{Close(\mathcal{D}^{\mathcal{T}})}(T)$. (où $\mathcal{D}^{\mathcal{T}}$ est la matrice transposée de $\mathcal{D}$).

3.1.10 Écriture des requêtes

Dans cette partie, nous donnons quelques exemples de requêtes que nous pouvons écrire avec le langage précédemment présenté. Nous considérons un contexte d'extraction générique $\mathcal{C}$ comme celui précédemment présenté. De plus, nous considérons que $\mathtt{I}_1 = \{(A_l = 1), 1 \leq l \leq a\}$ et $\mathtt{I}_2$ est l'ensemble des descripteurs que l'on peut construire sur l'attribut *event_attr*.

- Premièrement, nous donnons un exemple de requête simple pour l'extraction des ensembles d'items clos. Le seuil de fréquence minimale est t.
  create pattern set P **as** $r \in \mathcal{I}(\mathtt{I}_1) \wedge r \in P \wedge C_{MinFreq(\mathcal{C},t)}(r) \wedge C_{Close(\mathcal{C})}(r)$
- Ici, nous donnons un exemple d'opération de post-traitement : imaginons que nous ayons extraits certains ensembles d'items intéressants dans P depuis un contexte d'extraction $\mathcal{C}$ et que nous souhaiterions obtenir les tuples de $\mathcal{C}'$ (qui a le même schéma que $\mathcal{C}$) contenant au moins un ensemble d'items de P.
  create data set D **as** $\alpha(\mathcal{C}', P)$
- Voici maintenant un exemple de requête dans laquelle nous extrayons des ensembles d'items fréquents (seuil t) et où nous réalisons ensuite une extraction de motifs séquentiels fréquents (seuil t') dans le contexte obtenu en conservant seulement les transactions de $\mathcal{C}$ contenant au moins un ensemble d'items fréquent de taille supérieure à 4.
  create pattern set P_1, P_2 **as** $r \in \mathcal{I}(\mathtt{I}_1) \wedge\ s \in \mathcal{S}(\mathtt{I}_2) \wedge\ r \in P_1 \wedge\ s \in P_2 \wedge\ (C_{MinFreq(\mathcal{C},t)}(r) \wedge C_{MinSize(4)}(r)) \wedge (C_{MinFreq(\alpha(\mathcal{C},P_1),t')}(s))$

3.2 Scénario prototypique

Il est tout d'abord important de préciser ce que nous entendons par scénario. De notre point de vue, un scénario est un ensemble de manipulations sur les données et les motifs qui peuvent être décrites en terme de requêtes inductives. Une requête inductive peut être une requête de prétraitement, une requête d'extraction ou bien une requête de post-traitement (ce qui inclue les opérations de croisement -ou crossing-over - entre les données et les motifs). Les requêtes de prétraitement peuvent être réalisées à l'aide de langages standards (typiquement SQL pour les bases de données relationnelles) et certaines extensions comme PL/SQL pour l'écriture de fonctions plus complexes (comme la discrétisation d'attributs). Ecrire un langage pour le post-traitement des motifs est plus délicat parce que cette tâche est profondément liée avec le type de motifs (cf. par exemple [TL02] pour une proposition de langage pour le post-traitement des règles d'association).

Un scénario prototypique est une description abstraite et non ambigüe de ce que l'utilisateur fait ou peut faire. Typiquement, quand on analyse le comportement réel de l'utilisateur, nous pouvons observer qu'il réalise souvent les mêmes opérations en changeant seulement les valeurs des paramètres. Dans un scénario prototypique, toutes ces phases de "règlages" intermédiaires vont être généralisées en une seule requête, en utilisant le formalisme des scénarios d'ECD pour réécrire ces manipulations à un plus haut niveau. Une fois abstraites, ces requêtes deviendront des objets formels, sur lesquels il sera possible de raisonner. Par exemple, nous pourrons mettre en valeur certaines propriétés importantes des contraintes entrant en jeu pour l'extraction (comme la monotonie ou l'anti-monotonie). De plus, en tirant partie des différentes valeurs qui peuvent être données à un paramètre spécifique, nous pourrons mettre en évidence certaines dépendances entre les ensembles de résultats. En effet, lorsque l'on modifie les valeurs

de seuil dans des algorithmes d'extraction par niveau, il arrive fréquemment que l'on soit amener à recalculer au moins partiellement certains ensembles de motifs déjà découverts lors des précédentes extractions. Enfin, nous pourrons analyser les scénarios prototypiques en tant que séquences de requêtes et proposer des techniques de compilation pour des séquences de requêtes pour lesquelles nous disposons d'outils adaptés.

Un autre avantage des scénarios prototypiques est le transfert d'expertise entre l'utilisateur final et les experts en matière d'extracteurs de motifs. Cela permet à l'utilisateur de décrire avec une technique formelle les opérations que le spécialiste devra réaliser pour lui. Dans la pratique actuelle, le spécialiste en fouille de données est souvent amené à décider à la place de l'expert sur certains points particuliers du processus d'extraction. Ecrire des requêtes d'extraction et de post-traitement à l'aide d'un formalisme unifié permet ainsi à l'utilisateur de mieux décrire son problème et de s'échanger les consignes plus facilement.

Dans le domaine du génie logiciel, UML propose déjà un formalisme graphique pour la représentation de scénarios. Néanmoins, cette notion placée dans le contexte de la modélisation d'applications reste relativement éloignée de ce qui nous intéresse ici. En UML, trois types de diagrammes interviennent dans la représentation des interactions entre l'utilisateur et le système : les use cases, le diagramme de collaborations et le diagramme de séquences. Les use cases ont avant tout pour but l'identification des acteurs du système et la modélisation à un haut niveau des fonctionnalités à implémenter. Mais cette description reste trop imprécise par rapport à nos besoins, car nous voulons pouvoir spécifier précisément les types de contraintes et les paramètres utilisés. Le diagramme de collaborations a pour but la description des interactions entre objets d'un point de vue statique. Cela ne nous convient pas non plus car nous nous intéressons ici aux interactions entre l'utilisateur et le système et non pas aux interactions entre les objets du système, à part éventuellement pour les échanges entre les parties données et motifs d'une base de données inductive, mais cela ne recouvre qu'une partie de ce que l'on souhaite faire. Enfin, le diagramme de séquences, orienté vers la gestion événementielle de l'exécution des processus, apporte la prise en compte de la dimension temporelle dans les échanges de messages entre les objets composant l'application à spécifier. D'une manière générale, l'emploi d'UML semble inadéquat pour le problème auquel nous nous intéressons car ce formalisme graphique reste avant tout tourné vers la description d'applications logicielles alors que nous nous intéressons avant tout à la modélisation formelle des interactions entre l'utilisateur et le système de base de données inductive.

3.3 Exemples de scénarios prototypiques

3.3.1 Contexte applicatif

Nous présentons ici quelques exemples de scénarios prototypiques afin d'illustrer ce concept. Pour cela, nous avons choisi le contexte applicatif du Web Usage Mining. En effet, dans ce domaine, on peut typiquement avoir aussi bien de l'information de type "ensemble d'items" (par exemple des ensembles de pages web visitées), que de l'information du type "motifs séquentiels" (par exemple, la suite des pages visitées par un utilisateur). Dans cette partie, nous détaillons précisément le type de données jouet que nous allons considérer pour nos exemples et nous précisons la terminologie.

On considère un ensemble d'utilisateurs pour lesquels on va logguer une partie de l'activité Internet (Web et messagerie instantanée). Ces utilisateurs pourront se connecter depuis un

ensemble de PC situés dans une même salle. Les PC ne sont pas nominatifs et n'importe quel utilisateur peut se connecter depuis n'importe quel PC. Néanmoins, chaque utilisateur doit s'identifier sur le PC, ce qui permet de savoir à tout moment qui est connecté sur quelle machine. De plus, nous caractérisons les pages web visitées à l'aide de mots clés (grâce par exemple à une méthode de comptage de la fréquence des mots).

Ces différentes informations sont stockées dans des tables relationnelles. La table *users* contient ainsi des informations sur les utilisateurs, comme le nom, l'âge, les revenus, les centres d'intérêt, etc. Nous avons ensuite une table *slots* qui contient les informations sur les utilisations des machines. Elle comporte cinq champs : un identificateur de slot, un identificateur d'utilisateur, un identifiant de machine (son adresse IP), la date de début du slot, la date de fin du slot. On loggue les sites web visités par les utilisateurs, on a donc une table *logs_web* contenant l'identifiant du slot visiteur, le chemin complet de la page visitée, la date et l'heure de la consultation. Pour caractériser les pages web visitées, nous avons une table *keywords* contenant sur chaque ligne le chemin de la page, ainsi qu'un mot clé caractéristique de la page. Nous avons également une table *weights* qui associe un poids à chaque mot clé (en fait, un entier entre 1 et 10), afin de quantifier leur pouvoir discriminant. Ainsi, le terme "constitution" est plus discriminant du texte qu'un article comme "le" ou qu'un mot aux significations multiples comme "note". Cette table *weights* est donc de schéma $(keyword, weight)$. Enfin, nous logguons les messages envoyés par les systèmes de messagerie instantanée entre les PC de la salle. Nous avons pour cela une table *logs_mess* qui contient les identifiants des PC source et destinataire d'un message ainsi que l'heure à laquelle cet envoi a été effectué. Pour résumer, nous avons donc les tables suivantes :

- $Users(user\_id, name, age, etc)$
- $Slots(slot\_id, user\_id, IP, begin, end)$
- $Logs\_web(log\_id, slot\_id, chemin, time)$
- $Logs\_mess(log\_id, slot\_id\_source, slot\_id\_dest, time)$
- $Keywords(chemin, keyword)$
- $Weights(keyword, weight)$

La Figure 3.5 montre un exemple de ce genre de données.

Dans les exemples suivants, nous allons utiliser ces données pour spécifier différentes manipulations d'ECD. Nous montrerons également comment construire différents contextes d'extraction à partir de ces données. En effet, il sera nécessaire de transformer les données brutes initiales afin d'obtenir des contextes d'extraction conformes à la définition donnée précédemment et sur lesquels nous pourrons travailler.

3.3.2 Un premier exemple

Pour présenter le concept de scénario d'ECD de manière simple, nous prendrons un premier exemple qui contient peu de manipulations et qui ne s'intéresse qu'à la seule problématique de l'extraction d'ensembles d'items fréquents, un problème classique en ECD.

Nous supposons que la personne chargée de gérer la salle dans laquelle se trouvent les machines souhaite mieux connaître les habitudes des utilisateurs. Pour cela, il voudrait trouver les ensembles fréquents de domaines visités au cours des slots. Cela pourrait lui permettre par exemple de configurer automatiquement les favoris des navigateurs en y incluant ces sites. Pour cela, il faut donc construire un contexte d'extraction WUM_1 qui contiendra les identifiants de slots sur les lignes et les noms des domaines en colonnes. Les noms des domaines visités peuvent être facilement obtenus en tronquant les chemins des pages webs visitées au premier caractère

user_id	Name	Age	Income	...
1	Emilie	20	Medium	...
2	John	22	Low	...
3	Antoine	26	High	...
...	...	...	...	...
n	Julie	19	Medium	...

Table *users*

Keyword	Weight
institutions	8
juridique	6
Constitution	9
...	...
élèves	4

Table *Weights*

slot_id	*user_id*	IP	begin	end
1	1	192.168.0.4	10/08/04 - 17h22	10/08/04 - 18h14
2	3	192.168.0.1	10/08/04 - 17h40	10/08/04 - 19h02
3	5	192.168.0.6	10/08/04 - 17h58	10/08/04 - 19h09
4	6	192.168.0.5	10/08/04 - 18h12	10/08/04 - 19h30
5	1	192.168.0.4	10/08/04 - 18h17	10/08/04 - 19h50
...	...	...	...	...
p	n	192.168.0.4	18/08/04 - 14h25	18/08/04 - 14h45

Table *slots*

log_id	*slot_id*	chemin	time
1	1	www.lemonde.fr/article189.html	10/08/04 - 17h23
2	1	www.linuxfr.org/article090804.html	10/08/04 - 17h28
3	2	www.libe.fr/0804/1234.html	10/08/04 - 18h02
4	2	www.caramail.com/connect.php	10/08/04 - 18h49
5	2	www.lemonde.fr/article18.html	10/08/04 - 19h00
6	3	fr.news.yahoo.com/index.html	10/08/04 - 19h01
7	3	www.insee.fr/etude99.html	10/08/04 - 19h04
...	...	...	...
m	p	www.insa-lyon.fr/formation/page1.html	18/08/04 - 14h42

Table *logs_web*

log_id	*slot_id_source*	*slot_id_dest*	time
1	1	2	10/08/04 - 17h53
2	3	4	10/08/04 - 19h05
...	...	...	...
q	$p-2$	p	18/08/04 - 14h30

Table *logs_mess*

chemin	keyword
www.lemonde.fr/edition/article189.html	institutions
www.lemonde.fr/edition/article189.html	juridique
www.lemonde.fr/edition/article189.html	Constitution
...	...
www.insa-lyon.fr/formation/page1.html	pluricompétents
www.insa-lyon.fr/formation/page1.html	innovation
www.insa-lyon.fr/formation/page1.html	élèves

Table *keywords*

FIG. 3.5 – Extraits des données utilisées pour l'exemple

$Slot\_id$	$domain_1$	$domain_2$	...	$domain_l$
1	1	0	...	1
2	0	1	...	0
3			...	
...	...	...	...	...
p	0	1	...	1

TAB. 3.4 – Extrait du contexte d'extraction WUM_1

slash('/') et en conservant la partie gauche. On considèrera donc un nouvel ensemble d'attributs $Domains = \{domain_1, domain_2, \ldots, domain_l\}$ contenant l'ensemble des domaines visités par les utilisateurs dans la table initiale *logs_web*. Pour chaque slot, un 1 dans la case correspondant à l'attribut $domain_i$ signifiera que l'utilisateur du slot correspondant à visité le domaine $domain_i$ durant sa navigation. Par rapport à notre définition générique de contexte d'extraction, nous avons les correspondances suivantes : $tuple\_id = slot\_id$, $A_i = domain_i$ pour tout $i \in \{1, \ldots, n\}$. La Table 3.4 donne un aperçu de ce contexte booléen pour l'extraction d'ensembles d'items et de règles d'association.

L'administrateur souhaite connaître les ensembles de sites apparaissant dans une certaine proportion de sessions (paramètre min_freq1). Puis comme il souhaite voir l'évolution des résultats lorsqu'on abaisse le seuil de fréquence, il désire baisser la valeur du seuil à une nouvelle valeur min_freq2 ($< min\_freq1$) et ne souhaite extraire que des ensembles clos de domaines visités. Ce scénario prototypique peut donc se résumer grâce aux deux requêtes suivantes :

1. **create pattern set** P_1 **as** $r \in \mathcal{I}(\texttt{Domains}) \wedge\ r \in P_1 \wedge\ C_{MinFreq(WUM_1, min\_freq1)}(r)$
2. **create pattern set** P_2 **as** $r \in \mathcal{I}(\texttt{Domains}) \wedge\ r \in P_2 \wedge\ C_{MinFreq(WUM_1, min\_freq2)}(r) \wedge\ C_{Close(WUM_1)}(r)$

Le problème de l'extraction d'ensembles d'items fréquents est un problème classique en ECD, néanmoins, on sait que l'extraction de tels motifs n'est pas forcément simple, et ce, malgré l'anti-monotonie de la contrainte de fréquence minimale. Ainsi, dans le cas où la matrice est dense (c'est-à-dire qu'elle comporte beaucoup de 1 et relativement peu de 0), le nombre d'ensembles d'items vérifiant la contrainte de fréquence minimale est très important et ce, même pour des valeurs de seuils élevées. Cette explosion du nombre de motifs peu rendre les extractions irréalisables aves l'approche APriori traditionnelle. Dans de tels cas, il faudra donc optimiser le traitement de ces requêtes inductives afin de réussir à éxécuter ce scénario d'ECD. Nous remarquons enfin sur cet exemple que certains résultats de la première requête font partie des résultats de la deuxième, ce qui permet de mettre en évidence une dépendance utile et exploitable entre les deux requêtes.

On voit sur cet exemple qu'un scénario prototypique d'ECD ne correspond pas forcément à un ensemble d'opérations que l'on sait réaliser, mais que l'on souhaite réaliser. Cela permet de mettre en évidence le fait que les scénarios prototypiques sont surtout utiles pour le transfert d'expertise. Ils permettent à un expert du domaine de formaliser ses besoins en termes de manipulations afin de pouvoir ensuite transférer ses problèmes à des spécialistes en ECD.

page_id	adresse	institutions	juridique	SCO	...	élèves
1	www.lemonde.fr/edition/article189.html	1	1	0	...	0
2	www.linuxfr.org/article090804.html	0	1	1	...	0
...	...	...	...	...	...	...
q	www.insa-lyon.fr/formation/page1.html	0	0	0	...	1

TAB. 3.5 – Extrait du contexte WUM_2

3.3.3 Exemple de scénario

Nous présentons ici un scénario plus complexe dans le cadre du Web Usage Mining. Le but est de découvrir des chemins fréquents et similaire à un chemin de référence parmi les sites visités par les utilisateurs. Pour déterminer la similarité entre les pages web, nous réaliserons une recherche de concepts booléens [RBCB03, BBR04] reliant les pages web et les mots clés. Ensuite, nous essaierons de caractériser les ensembles d'utilisateurs ayant réalisé de tels cheminements et ceux qui ne l'ont pas fait. Cela permettra par exemple à l'administrateur de proposer des bookmarks personnalisés à de nouveaux utilisateurs en fonction de leur profil. Nous donnons donc ici, étape par étape, le scénario prototypique correspondant à de telles manipulations. Ce scénario fait intervenir des extractions de concepts, d'ensembles d'items, de motifs séquentiels classiques (constitués d'items) et logiques (constitués d'atomes).

Étape 1 : construction d'un contexte pour les mots clés On souhaite tout d'abord trouver des ensembles de mots clés fréquents parmi les différentes pages visitées. Plus précisément, on souhaite extraire des concepts, c'est-à-dire des couples d'ensembles d'items de la forme (T,G) où T correspond à un ensemble de pages et G à un ensemble de mots clés. Pour cela, il faut construire un contexte d'extraction booléen à partir de la table *keywords*. Nous obtiendrons un contexte WUM_2 de schéma $(page\_id, adress, keyword_1, \ldots, keyword_n)$ où $keyword_i, 1 \leq i \leq n$ correspond aux différents valeurs que peut prendre l'attribut *keyword* dans la table initiale *keywords*. Le champ adresse est obtenu en faisant la concaténation des deux champs *domaine* et *chemin*. On remarque au passage que ce problème de représentation des données pour l'extraction des ensembles d'items est le même que celui déjà évoqué dans le Chapitre 2 avec `MSQL` et `MINE RULE`. Par rapport à la définition générique de contexte d'extraction donnée précédemment, nous avons les équivalences suivantes : $tuple\_id = page\_id$, $ind\_attr = adresse$, $A_l = keyword_l, 1 \leq l \leq n$. La Table 3.5 montre un exemple d'un tel concept booléen. Nous considérons alors un alphabet `Kw` correspondant à l'ensemble des descripteurs que l'on peut construite sur les attributs booléens de WUM_2 et un alphabet `Adr` correspondant aux différentes descripteurs construits sur l'attribut *adresse*.

Étape 2 : extraction de concepts Une fois le contexte obtenu, on souhaite découvrir des concepts associant pages web et mots clés. On espère ainsi trouver des ensembles de pages web au contenu similaire, car chaque concept regroupera des pages partageant les mêmes termes. Cependant, afin de limiter la recherche aux concepts les plus pertinents, on ne souhaite conserver que ceux qui font intervenir un nombre minimal t_1 de pages web et un nombre minimal t_2 de mots clés. En effet, les petits concepts, constitués d'un faible nombre de pages web et de mots clés risquent plus d'être le fruit du hasard que de la réelle proximité entre les objets qu'ils

contiennent. En outre, on utilise une contraine d'aggrégat afin de mieux discriminer les concepts, on ne souhaite ainsi extraire que des concepts où la somme du poids des mots clés contenus dans T soit supérieure à un seuil t_3.

create pattern set P_1 **as** $(T,G) \in P_1 \wedge (T,G) \in \mathcal{K}(\texttt{Kw},\texttt{Adr}) \wedge C_{MinSize(t_1)}(T) \wedge C_{MinSize(t_2)}(G) \wedge Sum(\sigma(Weights;T), weight) \geq t_3$

Une fois les concepts extraits, il devient possible de définir une mesure de similarité entre les pages web. En effet, si on choisit de noter *Concepts* la fonction qui pour une page web renvoie l'ensemble des concepts auxquels elle appartient, il devient alors possible de quantifier la similarité de deux pages web p_1 et p_2. Par exemple, on peut prendre :
$Sim(p_1,p_2) = \frac{|Concepts(p_1) \cap Concepts(p_2)|}{|Concepts(p_1) \cup Concepts(p_2)|}$

Étape 3 : Construction d'un contexte pour les parcours Maintenant, l'administrateur de la salle de PC souhaite rechercher les parcours web fréquents réalisés par les utilisateurs durant leurs slots d'utilisation des PC. Les parcours recherchés correspondent donc à des motifs séquentiels, il faut donc un contexte qui conviennent à ce genre d'extractions. La table *logs_web* correspond à ces besoins, nous pouvons donc considérer un contexte $WUM_3 = logs\_web$ en prenant comme équivalences avec notre contexte générique les égalités : $tuple\_id = log\_id$, $ind\_attr = slot\_id$, $ordering\_attr = time$, $event\_attr = chemin$. Les séquences d'entrée correspondront alors à la suite ordonnée des pages web visitées à l'intérieur d'un slot. Nous pouvons alors considérer un alphabet `Pages` constitué des descripteurs basés sur l'attribut *event_attr*.

Étape 4 : Extraction de motifs séquentiels Comme il a été dit dans la paragraphe précédent, l'administrateur souhaite rechercher des parcours Web fréquents apparaissant dans au moins t_4 des séquences d'entrée. Cependant, il ne s'intéresse qu'à des parcours qui sont aussi similaires à un parcours $surf\_ref$ de référence qu'il connaît (et qui correspond à une succession de pages web bien précise). Pour quantifier la similarité entre les parcours extraits et $surf\_ref$, il souhaite utiliser une distance semblable à celle utilisée dans [CMB02]. Pour déterminer les coûts d'insertion, de suppression et de substitution d'un élément du motif $surf\_ref$, il pourra se servir de la fonction Sim définie au point numéro 2 de ce scénario.

La requête d'extraction correspondante peut alors s'écrire :

create pattern set P_2 **as** $s \in P_2 \wedge s \in \mathcal{S}(\texttt{Pages}) \wedge C_{MinFreq(WUM_3,t_4)}(s) \wedge C_{Sim(surf\_ref)}(s)$

Étape 5 : Croisement entre données et motifs Une fois l'extraction de parcours web fréquents et similaires à $surf\_ref$ réalisée, l'administrateur souhaite séparer le contexte WUM_3 en deux sous-contextes : l'un contenant les parcours web contenant au moins un motif de P_2 et l'autre contenant les parcours web ne contenant aucun motif de P_2. Il s'agit donc de deux opérations de croisement qui se résument par les requêtes :
create data set WUM_4 **as** $\alpha(WUM_3, P_2)$
create data set WUM_5 **as** $\neg\alpha(WUM_3, P_2)$

Ensuite, l'utilisateur veut récupérer les profils des utilisateurs qui ont occupé au moins une fois un des slots figurant dans WUM_4, et ceux des utilisateurs qui ne correspondent qu'à des slots apparaissant dans WUM_5 (c'est-à-dire ceux qui n'ont jamais effectué un parcours de P_2). Cela peut être réalisé en SQL comme suit :

user_id	$age \leq 20$	$20 < age \leq 25$	$age > 25$	low_income	medium_income	high_income	...
1	1	0	0	0	1	0	...
2	0	1	0	1	0	0	...
3	0	1	0	0	0	1	...
...	...	...	...	...	...	...	...
n	1	0	0	0	1	0	...

TAB. 3.6 – Exemple de contexte booléen pour les utilisateurs

create data set WUM_6 **as** $users \bowtie_{user\_id} slots \bowtie_{slot\_id} WUM_4$
create data set WUM_7 **as** $(users \bowtie_{user\_id} slots \bowtie_{slot\_id} WUM_5) \backslash WUM_6$

Étape 6 : Caractérisation des utilisateurs L'administrateur souhaite maintenant caractériser les utilisateurs des deux ensembles WUM_6 et WUM_7. Plus précisément, il souhaite essayer de les caractériser à l'aide d'ensembles d'items les décrivant. Pour cela, il faut transformer les ensembles WUM_6 et WUM_7 afin d'obtenir des contextes booléens : on peut par exemple discrétiser les attributs continus. La Figure 3.6 donne un aperçu d'un tel contexte booléen (par rapport à la table initiale *users*, la colonne des noms a été supprimée). Cette discrétisation sera effectuée par la fonction *Discretize* et les contextes WUM_8 et WUM_9 auront donc le même schéma. On notera `Items` l'ensemble des descripteurs que l'on peut construire sur les attibuts booléens de ce schéma.

L'administrateur souhaite trouver des ensembles d'items concernant les caractéristiques des utilisateurs qui permettent de discriminer les deux contextes. Afin de limiter la taille des ensembles retournés, il ne s'intéresse qu'au ensemble d'items clos. Il considère deux seuils t_5 et t_6 avec $t_5 \leq t_6$. Un ensemble d'items sera discriminant si sa fréquence dans un contexte est supérieure à t_6 et inférieure à t_5 dans l'autre contexte. Ce processus de fouille de données peut donc s'écrire à l'aide de la requête suivante :

create pattern set P_3 **as** $r \in P_3 \wedge\ r \in \mathcal{I}(\texttt{Items}) \wedge\ (C_{MinFreq(WUM_8,t_5)} \wedge\ C_{Close(WUM_8)} \wedge\ C_{MaxFreq(WUM_9,t_6)}) \vee (C_{MinFreq(WUM_9,t_5)} \wedge\ C_{Close(WUM_9)} \wedge C_{MaxFreq(WUM_8,t_6)})$

On notera que cette extraction de motifs fait intervenir une disjonction de contraintes primitives.

Étape 7 : Recherche de parcours sous forme relationnelle L'administrateur s'intéresse ensuite à la découverte de parcours d'utilisateurs mélangeant les pages web visitées et les messages transmis. Plus précisément, il s'intéresse à la présence dans ses jeux de données de parcours fréquents qui sont dus à la communication entre deux utilisateurs. Il est en effet courant que, lors de séances de surf, certaines personnes se connaissent dans la salle où se trouvent les PC, et lorsqu'une personne visite un site internet, elle envoie un message à une de ses connaissances se trouvant dans la salle, afin que le destinataire jette lui aussi un œil sur cette page. Ce phénomène peut parfois concerner plusieurs pages et plusieurs messages, et on peut mettre en évidence des "cascades" parmi les parcours utilisateurs (le destinataire réalise le même parcours que l'expéditeur avec un décalage temporel). La recherche de parcours de ce type met en évidence le besoin d'avoir une information exprimée sous forme relationnelle, il faut donc transformer la représentation des données sources afin de faire apparaître ces relations dans les séquences d'entrées sur lesquelles on appliquera un algorithme d'extraction de motifs de nature

séquentielle.

Pour chaque journée d'ouverture de la salle, on crée une séquence d'entrée constituée d'atomes clos de la logique du premier ordre : nous ordonnons dans le temps les connexions des utilisateurs, les pages web visitées et les messages transmis (avec l'expéditeur et les destinataires). Le début d'un slot correspondant à la connexion d'un utilisateur sur l'une des machines sera notifiée par le prédicat *begin* (d'arité 1), la fin du slot par le prédicat *end* (d'arité 1), la visite d'une page par le prédicat *visit* (d'arité 2) et l'envoi du message par le prédicat *message* (d'arité 2). Nous utiliserons donc un alphabet $\mathcal{P}$ de symboles de prédicats. Ainsi, avec les données d'exemple fournies dans la Figure 3.5 nous pouvons construire un nouveau jeu de données WUM_{10} contenant un ensemble de séquences d'entrée, chacune d'elles correspondants à un jour particulier. La Figure 3.6 donne la séquence d'entrée pour la journée du 10/08/2004.

```
begin(1)
visite(1,www.lemonde.fr/article189.html)
visite(1,www.linuxfr.org/article090804.html)
begin(2)
message(1,2)
begin(3)
visite(2,www.libe.fr/0804/1234.html)
begin(4)
end(1)
begin(5)
visite(2,www.caramail.com/connect.php)
visite(2,www.lemonde.fr/article18.html)
end(2)
visite(3,fr.news.yahoo.com/index.html)
visite(3,www.insee.fr/etude99.html)
message(3,4)
visite(4,univ-lyon1.fr/index.html)
visite(4,www.insee.fr/etude99.html)
end(3)
end(4)
end(5)
```

FIG. 3.6 – Exemple de séquence d'entrée

On disposera d'une séquence de ce type par jour. L'administrateur est donc intéressé par la découverte de séquences de la forme : deux utilisateurs se connectent, puis ils visitent les mêmes pages en s'échangeant des messages puis se déconnectent. Ils peuvent visiter ensemble plusieurs pages différentes. Si on représentait cela sous la forme d'une expression rationnelle, on pourrait dire que l'administrateur recherche des motifs du type $begin(X)$ $begin(Y)$ $visite(X, S)$ $message(X, Y)$ $visite(Y, S)$ $end(X)$ $end(Y)$ Ici, X et Y désignent des utilisateurs quelconques et S correspond à une page web quelconque.

Ici, la présentation du jeu de données n'est pas compatible avec le contexte d'extraction pour les requêtes sur le domaine ITEM que nous avons présenté, ce qui est normal puisque nous ne cherchons pas ici à extraire des motifs de cette nature. En fait ce que nous souhaitons extraire

ici, ce sont des motifs séquentiels où les items ont été remplacés par des littéraux de la logique du premier ordre qui permettent d'exprimer de l'information en relation. La contrainte sur l'aspect syntaxique de ces séquences pourra être utilisée afin de faciliter l'extraction de tels motifs à l'aide d'un algorithme par niveaux. Nous noterons cette contrainte $C_{ER(ref)}$ où ref sera l'expression rationnelle spécifiant l'aspect des séquences de littéraux à extraire. Dans l'exemple précédent, nous aurons ref = *begin begin* (*visite message visite*)* *end end*, l'étoile (*) de l'expression rationnelle servant à montrer le fait que ce processus visite d'un site par un utilisateur/message de cet utilisateur à un autre/ visite du site par l'autre utilisateur peut se répéter. Nous pourrons naturellement la combiner à une contrainte de fréquence minimale par rapport à un seuil t_7. La requête d'extraction que souhaite réaliser l'administrateur pourra alors se noter comme suit :
create pattern set P_4 **as** $s \in P_4 \wedge\ s \in \mathcal{M}(\mathcal{P}) \wedge\ C_{ER(ref)(s)} \wedge\ C_{MinFreq(WUM_{10},t_7)}$

Cette requête porte donc sur l'extraction de motifs séquentiels logiques et contient une conjonction de contraintes syntaxiques et de fréquence minimale.

3.4 Scénario pour l'évaluation

Un autre avantage important de l'abstraction de processus d'ECD est la description de scénarios d'évaluation pouvant être utilisés pour évaluer différentes implémentations de systèmes de bases de données inductives. D'un point de vue pratique, un scénario d'évaluation a la même apparence qu'un scénario prototypique. L'idée n'est pas ici de résumer dans un scénario des expérimentations pouvant concerner différents domaines de motifs sur un contexte donné, mais avant tout de mettre en avant certaines difficultés algorithmiques qu'il est intéressant d'étudier et pour lesquels on souhaite mesurer l'efficacité de différentes solutions. Nous pourrons ainsi tester divers systèmes sur ces scénarios d'évaluation, et faire ainsi une comparaison quantitative et qualitative en observant comment ils gèrent ces difficultés algorithmiques. En effet, pour l'exécution d'un même scénario, ces systèmes peuvent utiliser différents outils d'extraction et différentes stratégies, en utilisant par exemple, de l'information sur la caractérisation des données (comme la densité des contextes booléens) pour choisir d'utiliser un algorithme plutôt qu'un autre, ou bien encore réutiliser même partiellement le résultat de requêtes précédentes pour en résoudre de nouvelles.

Étant donné un scénario d'évaluation, un utilisateur va essayer de le résoudre en utilisant ses propres outils. Remarquons que cela ne signifie pas forcément que l'utilisateur possède les extracteurs qui permettent de répondre exactement à chaque requête et qui prennent en compte exactement les contraintes apparaissant dans celles-ci. En effet, pour une requête donnée, il est probable que l'utilisateur ne puisse extraire que les motifs qui vérifient un sous-ensemble des contraintes apparaissant dans la requête inductive. Dans ce cas, il lui est toujours possible d'écrire des programmes pour le traitement des contraintes restantes par post-traitement des résultats (cela est d'ailleurs généralement réalisé à l'aide de quelques scripts *ad hoc*). Ainsi, un scénario d'ECD n'implique pas une unique manière de traiter les requêtes inductives qu'il contient, c'est aux différents systèmes d'élaborer des stratégies pour traiter efficacement ces suites de requêtes.

En utilisant ces scénarios comme "benchmarks", l'évaluation des solutions d'ECD peut alors se faire de manière quantitative (consommation mémoire, ressources utilisées, nombres d'opérations d'Entrée/Sortie), mais il est plus intéressant d'analyser les stratégies d'évaluation et les techniques d'optimisations mises en œuvre afin d'outrepasser les difficultés algorithmiques du benchmark. Cette comparaison qualitative peut être réalisée à deux niveaux :

- Au niveau de la requête inductive, en analysant quels sont les algorithmes mis en œuvre afin de la résoudre. En effet, il est intéressant de voir qu'en fait, comme pour les requêtes SQL, il est possible de concevoir des plans d'exécution pour les requêtes inductives. Si aucun algorithme n'est disponible pour traiter une conjonction ou une disjonction de contraintes particulières, il faut analyser les méthodes mises en œuvre pour contourner le problème (par exemple, quelles contraintes ou conjonctions de contraintes sont poussées dans un algorithme par niveaux et quelles contraintes sont gérées par post-traitement). Si plusieurs solveurs sont utilisés pour résoudre une requête inductive, il peut être intéressant de voir comment ils collaborent, et quel plan d'exécution est utilisé. De plus, si un système dispose de plusieurs techniques pour traiter une même requête inductive, il est également pertinent de s'intéresser aux critères pris en compte pour choisir un plan d'exécution plutôt qu'un autre : il pourra ainsi s'agir de la densité d'un complexe booléen, de la corrélation des données, de la sélectivité des contraintes (que l'on peut essayer d'évaluer de manière heuristique).
- Au niveau du scénario, en regardant quelle stratégie globale est utilisée pour résoudre l'ensemble des requêtes. Par exemple, si nous savons qu'il existe certaines relations d'inclusion entre les ensembles résultats de deux requêtes différentes, le résultat d'une d'entre elles peut être utilisé afin de réduire le coût calculatoire de la résolution de l'autre. Ce type de dépendance peut être exploitée par la mise en place de cache [JB02b] et l'analyse formelle des relations entre les requêtes inductives, comme cela a été fait pour la langage `MINE RULE` dans [Meo03], où l'auteur met en évidence trois types de dépendances entre des requêtes `MINE RULE` (équivalence, inclusion, dominance) qui permettent de résoudre au moins partiellement une requête à partir des résultats d'une autre requête. Une solution d'ECD devrait être capable de caractériser certaines propriétés d'un scénario d'évaluation pour tirer partie de telles dépendances entre les requêtes.

3.4.1 Caractérisation des scénarios

Les performances d'un système implémentant une base de données inductive seront donc directement liées à sa capacité à traiter de manière efficace les requêtes qui composent le scénario d'évaluation. Pour cela, il est donc nécessaire que le système dispose de bons outils afin de caractériser au mieux possible les problèmes posés par le scénario. Cette caractérisation peut se faire à deux niveaux :

- Au niveau des données : on peut utiliser différentes mesures pour évaluer les propriétés des données. Certaines sont des mesures classiques de statistiques (moyenne et variance d'un attribut numérique par exemple) et d'autres des mesures adaptées à l'évaluation de la pertinence de l'utilisation d'une technique particulière. Ces mesures objectives aideront ensuite à mettre en place un plan d'exécution efficace pour le scénario considéré. Ainsi, lorsque l'on recherche des ensembles d'items fréquents dans des contextes booléens, il est utile de calculer la densité du contexte (c'est-à-dire la proportion de '1' dans la matrice). En effet, cela permet d'avoir une indication sur le nombre d'ensembles d'items candidats qui seront générés lors de l'utilisation d'un algorithme par niveau du style d'APriori : si cette densité est élevée, on peut s'attendre à des problèmes d'explosion combinatoire même pour des seuils élevés de fréquence minimale. Il peut alors être nécessaire d'utiliser des extracteurs à base de représentations condensées qui sont moins sensibles à ces problèmes de densité du contexte d'extraction initial.

– Au niveau du scénario : cela peut passer par l'analyse de la sélectivité des contraintes. Cela permet d'évaluer le capacité d'élagage des contraintes dans un algorithme par niveaux. Ainsi, une contrainte sélective permettra de supprimer de nombreux motifs dont la vérification de la validité par rapport à une autre contrainte (typiquement la fréquence minimale) aurait été inutile. Cependant, les contraintes les plus sélectives ne présentent pas toujours les propriétés idéales pour être poussées dans un algorithme d'ECD par niveaux. Dans le cas de l'extraction d'ensembles d'items, une contrainte syntaxique sous forme d'expression rationnelle sera très sélective, car elle dicte à la fois le contenu et la forme du motif recherché, alors qu'une contrainte syntaxique imposant juste la présence d'un item quelque part dans le motif est moins contraignante. Malheureusement, les contraintes à base d'expression rationnelle sont plus difficiles à exploiter que les contraintes d'inclusion "simples". La caractérisation du scénario peut aussi passer par la mise en évidence les dépendances entre les requêtes. Ainsi si l'on remarque que le résultat d'une requête est un sous-ensemble du résultat d'une autre requête, on peut faire l'économie d'une extraction à l'aide d'un simple post-traitement. La découverte de telles relations nécessite une analyse poussée des requêtes, mais aussi des liens de dépendances fonctionnelles ou d'inclusion entre les attributs de différents contextes sur lesquels sont réalisés les extractions.

3.4.2 Aspect d'un scénario idéal

Nous avons vu précédemment l'utilité des scénarios d'évaluation comme outil de comparaison de systèmes implémentant des BDI. Cela nous a permis de mettre en évidence le rôle fondamental d'une bonne caractérisation du problème aussi bien au niveau des données qu'au niveau des requêtes inductives qu'il met en jeu. Ceci étant, on peut se demander quelle doit être concrètement l'aspect d'un scénario d'évaluation et ce qu'il contient. De notre point de vue, un scénario prototypique doit posséder les caractéristiques suivantes pour sa formulation :

– Il doit tout d'abord être basé sur des données librement accessibles à tout ceux qui souhaitent tester leur système. Il s'agit là d'une condition découlant du concept même d'un benchmark. En revanche, même si cela est très souhaitable, il n'est pas absolument nécessaire que les données soient des données réelles. En effet, lorsque l'on souhaite éprouver le comportement de certaines algorithmes d'extraction sur des cas limites qui ne peuvent correspondre qu'à des distributions de données très particulières. Il est en effet parfois impossible de trouver des jeux de données réels qui possèdent les caractéristiques requises. Un exemple de ces cas extrêmes est l'extraction d'ensembles d'items fréquents dans des contextes booléens extrêmement denses. Trouver des jeux de données réels très denses est difficile, il peut alors être préférable de synthétiser un contexte booléen très dense sur lequel on sait qu'un algorithme comme APriori aura du mal à s'éxécuter.
– Il faut que le problème posé par le scénario d'évaluation soit pertinent et relativement complexe. La complexité ne se mesure pas au nombre de requêtes inductives contenues dans le scénario d'évaluation mais à la pertinence des difficultés algorithmiques que le scénario met en évidence. Ainsi, un scénario contenant une seule requête d'extraction de motifs séquentiels fréquents peut sembler simple au premier abord. Mais si le jeu de données utilisé pour le test contient de nombreuses répétitions, on sait que les techniques d'extraction classiques peuvent devenir très difficiles à cause notamment de phases de comptage coûteuses. Il faudra alors utiliser des approches comme celle décrite dans [LRBE03] qui sont plus adaptées à ce type d'extraction. L'extraction d'ensembles d'items fréquents dans

des contextes booléens denses est également un exemple de problème qui peut sembler simple dans sa formulation mais qui est finalement complexe à traiter.
- On doit savoir quel est le résultat recherché. En effet, si l'on souhaite comparer les différents systèmes de BDI, il faut quand même s'assurer que ceux-ci retournent bien les bons résultats. Ce point soulève le problème de la nature des jeux de données. En effet, si l'on choisit d'utiliser des données synthétiques pour le scénario d'évaluation, il faudra alors faire une génération "intelligente", comme par exemple utiliser l'ensemble de motifs que l'on cherche comme racine du générateur du jeu de données. Un exemple d'une telle génération est celle utilisée par le générateur Quest d'IBM, où les motifs à découvrir sont générés en premier et sont ensuite utilisés pour la synthèse du jeu de données. Si l'on travaille sur des données réelles, connaître le résultat peut être plus délicat, car cela signifie que l'on dispose au moins d'une technique pour traiter le scénario. Si ce n'est pas le cas, le scénario ne pourra pas servir de base à une évaluation de solutions BDI et restera au stade de scénario prototypique (on décrit ce que l'on souhaite réaliser).
- Enfin, le scénario peut contenir certaines données de caractérisation, comme des mesures statistiques sur les valeurs des attributs d'un contexte pour le domaine ITEM par exemple. Néanmoins, une telle information n'est pas nécessaire si le but premier du scénario d'évaluation est justement de tester la capacité des systèmes à caractériser les données.

Dans cette partie du mémoire, nous ne présentons pas de tels scénarios "idéaux". En effet, pour l'instant, les scénarios d'évaluation que nous avons présentés concernent des contextes contenant des données purement fictives. Néanmoins, dans le Chapitre 5, nous présenterons un premier scénario d'évaluation dans le domaine de la bioinformatique sur un jeu de données concernant les données d'expression de gènes.

3.4.3 Exemple

Nous donnons d'abord ici l'exemple d'un scénario d'évaluation relativement simple sur l'extraction des ensembles d'items fréquents. Nous reprenons le contexte du Web Usage Mining présenté dans la Section 3.3.2.

Ce scénario d'évaluation a pour but de mettre en évidence les capacités des extracteurs d'ensembles d'items fréquents à travailler à de faibles seuils de fréquence minimale et à réutiliser les résultats de requêtes précédentes. Pour cela, on réalise un première extraction des ensembles d'items fréquents avec un seuil de 40 %, une deuxième avec un seuil de 10 % et la troisième avec un seuil de 20 %. De manière formelle, cela peut s'écrire à l'aide des requêtes inductives suivantes :

1. **create pattern set** P_1 **as** $r \in \mathcal{I}(\texttt{Domains}) \wedge\ r \in P_1 \wedge\ C_{MinFreq(\mathcal{C}_{WUM_1},40\%)}(r)$
2. **create pattern set** P_2 **as** $r \in \mathcal{I}(\texttt{Domains}) \wedge\ r \in P_2 \wedge\ C_{MinFreq(\mathcal{C}_{WUM_1},10\%)}(r) \wedge\ C_{Close(\mathcal{C}_{WUM_1})}(r)$
3. **create pattern set** P_3 **as** $r \in \mathcal{I}(\texttt{Domains}) \wedge\ r \in P_3 \wedge\ C_{MinFreq(\mathcal{C}_{WUM_1},20\%)}(r)$

Si on analyse le scénario précédent, on s'aperçoit assez facilement d'un certain nombre de dépendances entre les ensembles de résultats renvoyés par les différentes requêtes. Ainsi, le résultat de la requête 1 est un sous-ensemble de celui de la requête 3. De même, une partie des clos fréquents calculés par la requête 2 font partie des résultats des requêtes 1 et 3.

Supposons que nous disposons de deux prototypes de BDI pour l'extraction d'ensembles d'items que l'on souhaite comparer. Le premier ne dispose que de l'algorithme APriori pour

l'extraction d'ensembles d'items fréquents. Le deuxième sait gérer l'extraction de représentations condensées d'ensembles d'items fréquents [BBR03, BR03].

Supposons que le contexte d'extraction soit relativement dense et que le seuil de fréquence en dessous duquel la technique APriori ne fonctionne plus soit 30% (à cause d'une explosion du nombre de candidats). A ce moment là, un implémentation de BDI qui ne peut utiliser que la seule technique APriori originale ne pourra pas exécuter la requête 2 ou 3. En effet, dans le cas de la requête 2, ce système sera bloqué par le faible seuil de fréquence et par le fait que la gestion de la contrainte de clôture devra être déférée à une manipulation de post-traitement. Pour la requête 3, le seuil de fréquence empêchera l'exécution de la requête. L'implémentation utilisant les représentations condensées pourra elle passer par la représentation sous forme de clos pour optimiser ses temps de traitement.

Ceci montre l'intérêt de l'évaluation quantitative des différentes implémentations. En effet, le but d'un scénario d'évaluation n'est pas simplement de tester les capacités d'un système à résoudre un problème donné, mais aussi de mettre en évidence les stratégies utilisées pour y parvenir. Ainsi, le premier système ne peut utiliser que l'algorithme APriori, cela implique nécessairement que toute contrainte additionnelle autre que la fréquence minimale sur les ensembles d'items à extraire ne pourra être traitée que dans une phase de post-traitement. Au contraire, le deuxième système, qui repose sur un plus grand nombre de solveurs, peut réaliser une exploitation active de la contrainte de fermeture de la requête N°2 afin de réduire l'espace de recherche des candidats. De même pour résoudre la requête N°3, on peut, si on connait P_2, utiliser un système de mise en cache des résultats [JB02b] ainsi qu'une technique de régénération de tous les ensembles d'items fréquents à partir de leur représentation condensée. La Figure 3.7 présente quelques extracteurs d'ensembles d'items fréquents développés au sein de l'équipe "Data Mining et Bases de Données Inductives" du LIRIS et les relations qu'ils entretiennent entre eux. On voit qu'il est possible d'obtenir des ensembles fréquents à partir des fermés fréquents, des libres fréquents ou des $\vee$-libres fréquents. Ainsi, lorsque le système doit faire face à une nouvelle requête d'extraction d'ensembles d'items fréquents, il dispose de plusieurs chemins possibles pour réaliser la tâche, là où un système basé sur un algorithme unique ne disposera que d'une seule technique. Il faut d'ailleurs noter qu'un bon nombre des langages de requêtes présentés dans la Section 2 sont eux même construits autour d'un faible nombre d'extracteurs.

On en arrive donc à la définition de véritables plans d'exécution des requêtes inductives. Face à une nouvelle requête d'extraction, le SGBDI doit déterminer quels solveurs sont les mieux à même de résoudre le problème en un minimum de temps et/ou d'espace. L'évaluation qualitative d'une BDI se fera donc aussi bien sur les résultats mesurables que sur les prototypes utilisés et les plans d'éxécution mis en œuvre.

Dans le cas de notre exemple sur le Web Usage Mining, voici deux plans d'exécutions possibles avec chacun des systèmes que nous considérons :

- Si on considère un système qui ne possède que l'algorithme APriori
  1. Cette requête est exécutée sans problèmes.
  2. La contrainte de fermeture est post-traitée. Mais l'extraction ne peut se terminer à cause du seuil de fréquence minimale qui est beaucoup trop faible
  3. L'extraction ne peut être réalisée à cause du seuil de fréquence minimale trop faible.
- Si on considère un système basé sur l'ensemble des extracteurs de la Figure 3.7.
  1. Cette requête peut être exécutée directement avec un algorithme du style APriori ou bien en passant par des représentations condensées comme les libres à partir desquels

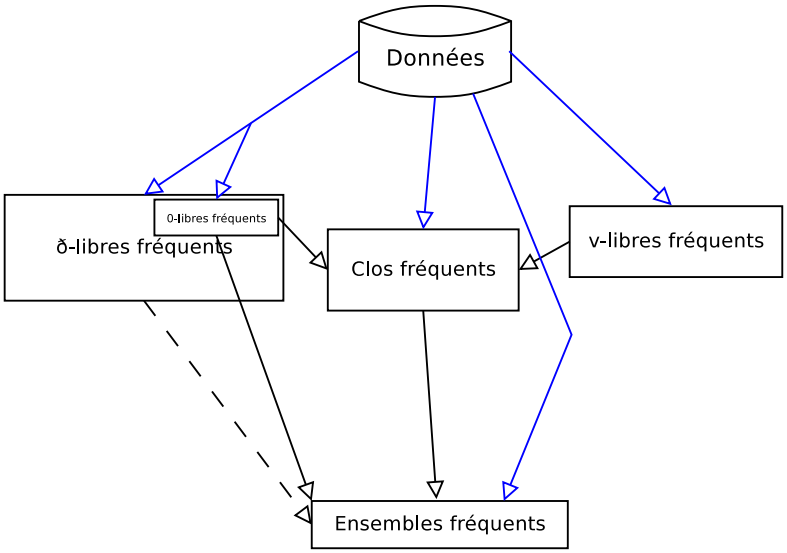

FIG. 3.7 – Solveurs pour les ensembles d'items

on peut régénérer l'ensemble des motifs recherchés.

2. Nous possédons un prototype permettant de pousser la contrainte de fermeture dans un algorithme par niveaux. C'est donc celui que nous utiliserons pour trouver P_2.
3. Enfin, pour résoudre la troisième requête, on peut passer à nouveau par la régénération des motifs fréquents depuis des représentations condensées. On peut aussi utiliser un système de cache afin de résoudre la requête 3 directement à partir des résultats de la requête 2 et sans repasser par une phase d'extraction.

De tels scénarios d'évaluation basé sur l'extraction d'ensembles d'items fréquents peuvent paraître courts, mais peuvent néanmoins être utiles en pratique. Ainsi, dans [HBK+03], nous nous sommes intéressés à l'exploitation des logs de firewall chez NOKIA, leader mondial dans le domaine des télécommunications mobiles. Nous disposions de plusieurs fichiers de logs, dont chaque ligne correspondait à une violation des règles du firewall. Plus précisément, chaque ligne contenait l'adresse IP de la machine émettrice du paquet suspect, l'adresse IP de la machine destination, le port visé, la date et l'heure, etc. Nous voulions rechercher des ensembles d'items fréquents dans ces données, afin d'arriver à compresser les informations contenus dans ces immenses fichiers. En effet, il se trouve que ces fichiers de logs contiennent beaucoup de redondance : ainsi, lors de la propagation d'un ver sur le réseau, on peut enregistrer plusieurs centaines voire milliers de tentatives ciblant un même port émises depuis une même adresse. A cause de cette très forte densité des données, l'utilisation d'un algorithme classique d'extraction d'ensembles d'items basé sur le paradigme APriori était impossible à des seuils relativement bas. L'utilisation du prototype AC Miner [BBR03] conçu pour l'extraction d'ensembles libres fréquents, nous a permis de réaliser des extractions à des seuil bien plus bas dans les fichiers de logs. Il a ainsi été possible de concevoir une méthode de compression des fichiers de logs de firewall basée sur les représentations condensées [HBK+03]. Un tel scénario concernant l'extraction d'ensembles d'items fréquent montre donc l'importance de l'évaluation des outils d'ECD afin de déceler ceux qui sont le plus à même de résoudre un problème donné.

3.4.4 Autre exemple

Un scénario d'évaluation peut être inspiré par un scénario prototypique, et peut être utilisé pour mettre en évidence des conjonctions ou disjonctions de contraintes pertinentes, pour lesquelles il serait souhaitable de disposer d'un outil performant. Pour cette partie, nous allons réutiliser le scénario prototypique que nous avons présenté dans la partie précédente. Nous allons mettre en évidence les problèmes algorithmiques qu'il comporte et nous proposerons des solutions pour les traiter. Néanmoins, pour certaines contraintes ou conjonctions de contraintes, comme il n'existait pas auparavant d'algorithmes pour les traiter, nous avons développé nous-même certains algorithmes qui seront présentés au chapitre suivant.

Nous rappelons donc ici le scénario présenté en Section 3.3.2 sous la forme d'une séquence de requêtes inductives.

1. Construction du contexte booléen WUM_2.
2. **create pattern set** P_1 **as** $(T,G) \in P_1 \wedge (T,G) \in \mathcal{K}(\texttt{Kw},\texttt{Adr}) \wedge C_{MinSize(t_1)}(T) \wedge C_{MinSize(t_2)}(G) \wedge Sum(\sigma(Weights;T), weight) \geq t_3$
3. Construction d'un contexte WUM_3 pour l'extraction de motifs séquentiels.
4. **create pattern set** P_2 **as** $s \in P_2 \wedge s \in \mathcal{S}(\texttt{Pages}) \wedge C_{MinFreq(WUM_3,t_4)}(s) \wedge C_{Sim(surf\_ref)}(s)$
5. **create data set** $WUM4$ **as** $\alpha(WUM_3, P_2)$
6. **create data set** $WUM5$ **as** $\neg\alpha(WUM_3, P_2)$
7. **create data set** $WUM6$ **as** $users \bowtie_{user\_id} slots \bowtie_{slot\_id} WUM_4$
8. **create data set** $WUM7$ **as** $(users \bowtie_{user\_id} slots \bowtie_{slot\_id} WUM_5) \backslash WUM_6$
9. **create pattern set** P_3 **as** $r \in P_3 \wedge (C_{MinFreq(WUM_8,t_5)} \wedge C_{Close(WUM_8)} \wedge C_{MaxFreq(WUM_9,t_6)}) \vee (C_{MinFreq(WUM_9,t_5)} \wedge C_{Close(WUM_9)} \wedge C_{MaxFreq(WUM_8,t_6)})$
10. **create pattern set** P_4 **as** $s \in P_4 \wedge C_{ER(ref)(s)} \wedge C_{MinFreq(WUM_{10},t_7)}$

La requête N°2 concerne une extraction de concepts. Étant donné que les concepts sont des couples (T,G) où T est un ensemble d'items clos et G est un ensemble contenant les identifiants des transactions incluant T, plusieurs stratégies sont possibles pour la résoudre :

- La première consiste à utiliser les extracteurs d'ensembles d'items fréquents décrits sur la Figure 3.7. En effet, ils permettent tous d'extraire de manière directe ou indirecte (par post-traitement) des ensembles d'items clos. On pourra ainsi obtenir la première partie des couples de concepts. Pour obtenir la deuxième partie, il suffira de faire un croisement entre ces ensembles d'items clos et le contexte booléen initial, nous pourrons alors récupérer les ensembles d'identifiants de transactions qui supportent les ensembles d'items clos extraits. Au final, nous aurons ainsi obtenu les concepts recherchés.
- Une deuxième solution consiste à utiliser un extracteur qui tire partie des propriétés particulières des concepts. En effet, pour un concept (T,G) extrait d'un contexte booléen $\mathcal{D}$, l'ensemble d'items T est clos dans D, mais on a aussi la propriété que G est clos dans le contexte booléen transposé $\mathcal{D}^T$. Les algorithmes comme DMiner [RBCB03, BRBR05] sont capables d'utiliser ces propriétés afin de pousser cette double contrainte de fermeture dans un algorithme d'extraction par niveaux. Pour cela, l'algorithme est capable de transposer la matrice si cette représentation permet une extraction plus efficace. De plus, le fait de travailler sur des concepts permet de travailler à de bas niveaux de fréquence minimale, mais aussi lorsque le contexte booléen possède un nombre de colonnes sensiblement plus élevé que le nombre de lignes [BRBR05].

La requête N°4 fait intervenir une conjonction de contraintes $C_{MinFreq} \wedge C_{Sim(ref)}$ sur l'extraction de motifs séquentiels. A l'heure actuelle, il n'existe pas d'algorithmes permettant de prendre directement en compte cette conjonction de contraintes pour en faire une exploitation active dans un algorithme par niveaux. La seule solution serait donc d'extraire d'abord les motifs séquentiels vérifiant la contrainte de fréquence minimale [AS95, Zak98, PHMA+04], puis de filtrer les résultats afin de ne conserver que ceux qui vérifient aussi la contrainte de similarité avec le motif de référence. Cependant, en exploitant certaines propriétés de la contrainte de similarité minimale, il est possible de la pousser dans un algorithme par niveaux, en conjonction avec une contrainte de fréquence minimale. C'est pourquoi dans le Chapitre 4, nous présenterons un algorithme capable de gérer cette conjonction de contraintes.

La requête N°10 est particulière dans le sens où elle cherche à extraire des motifs séquentiels composés de littéraux de la logique du premier ordre. Il serait possible de traiter cette requête avec des extracteurs sur les ensembles d'items en passant par une phase de propositionalisation afin d'obtenir un contexte pour l'extraction de motifs séquentiels. Néanmoins, une telle transformation pourrait s'avérer fastidieuse. De plus, les motifs découverts seraient alors des motifs séquentiels classiques et il faudrait une phase de post-traitement poussée afin de mettre en évidence les relations entre les éléments du motif, comme on cherche à le faire. Il faudrait alors refaire le chemin inverse de la propositionalisation afin de faire réapparaître les relations, puis eventuellement réaliser de nombreux tests de substitution et d'unification afin de trouver les motifs contenant les relations intéressantes. Dans le chapitre 4, nous proposons un algorithme nommé SPIRIT-L$^{\forall}$G pour la gestion d'une telle conjonction de contraintes lors de l'extraction de motifs séquentiels logiques.

3.5 Synthèse

Dans ce chapitre, nous avons présenté un premier cadre théorique ainsi qu'un premier langage pour les bases de données inductives. Nous avons ensuite enrichi ce langage avec des possibilités de typages de variables et de croisement entre les données source et les motifs obtenus par l'application d'algorithmes d'extraction. Après quoi, nous avons présenté une notion novatrice dans le monde de l'ECD : la notion de scénario, c'est-à-dire d'un ensemble de requêtes inductives décrivant une série d'actions composant un processus d'ECD. Nous avons distingué deux types de scénario : les scénarios prototypiques qui ont pour but la description abstraite d'un ensemble d'opérations afin de faciliter le transfert de savoir-faire, et les scénarios d'évaluation, destinés à comparer sur une base commune différentes implémentations de BDI.

Les scénarios prototypiques sont utiles non seulement pour le transfert d'expertise mais aussi pour la mise en évidence de problèmes intéressants, comme l'étude de l'exploitation d'une conjonction ou d'une disjonction de contraintes primitives dans des algorithmes d'extraction. Ainsi, sur l'exemple jouet tiré du domaine du Web Usage Mining, nous avons réussi à mettre en évidence deux conjonctions de contraintes pertinentes pour l'extraction de motifs séquentiels. La première d'entre-elles implique l'usage simultané d'une contrainte de fréquence minimale et de similarité sur les motifs séquentiels que l'on cherche à extraire ($C_{MinFreq} \wedge C_{Sim}$). La deuxième contrainte pertinente que nous avons mis en évidence concerne l'extraction de motifs séquentiels logiques sous une conjonction de contrainte de fréquence minimale et de contrainte syntaxique exprimée sous la forme d'une expression rationnelle ($C_{MinFreq} \wedge C_{ER}$). Pour l'instant, nous ne disposons d'aucun solveur capable de tirer partie de ces conjonctions de contraintes dans un

algorithme d'extraction. Il faut en effet considérer seulement l'une d'entre elles (généralement celle de fréquence minimale qui est anti-monotone) et post-traiter la deuxième. De plus, les deux contraintes syntaxiques ainsi insérées (similarité et expression rationnelle) ne présentent pas le caractère anti-monotone qui rendrait leur exploitation simple. Il faut donc étudier d'autres stratégies afin d'obtenir des algorithmes efficaces pour traiter ces conjonctions de contraintes. C'est justement le but du chapitre suivant qui présente deux algorithmes : Galibot, donc le but est d'extraire des motifs séquentiels vérifiant la conjonction de contraintes $C_{MinFreq} \wedge C_{Sim}$ et SPIRIT-L^{o}G qui extrait des motifs séquentiels logiques vérifiant $C_{MinFreq} \wedge C_{ER}$.

Nous avons vu que les scénarios d'évaluation sont utiles pour l'évaluation qualitative et quantitative d'un problème d'ECD donné. Ainsi, une fois un problème posé, nous pouvons étudier la faisabilité de l'exécution de ce scénario à l'aide de différents plans d'exécution faisant intervenir différents algorithmes d'extraction et scripts de manipulation des données. Dans le Chapitre 5, nous proposerons un scénario d'évaluation complet dans le domaine de la bioinformatique. Pour ce scénario, nous utiliserons un jeu de données réel et nous poserons un problème composé de différentes requêtes inductives. Pour ce problème, nous verrons qu'il est possible d'appliquer deux stratégies différentes impliquant différents extracteurs. Pour l'une d'elles, des opérations ensemblistes entre les ensembles d'items extraits peuvent être exploitées par l'algorithme pendant l'extraction. Dans l'autre technique, nous serons obligés de mettre au point une nouvelle technique de post-traitement des ensembles d'items afin de résoudre plus facilement ces manipulations ensemblistes sur les ensembles d'items lorsque ceux-ci sont stockés dans une base de données relationnelle. Nous en profiterons donc pour détailler plus précisément cette technique.

Chapitre 4

Solveurs pour l'extraction

Dans le chapitre précédent, nous avons vu qu'il était possible, à partir de scénarios prototypiques, de mettre en évidence des problèmes algorithmiques intéressants dans un but de transfert d'expertise. Ceci peut passer par exemple par l'identification d'une conjonction ou d'une disjonction spécifique de contraintes que l'on souhaite exploiter de manière active dans un algorithme par niveaux. A partir d'un exemple inspiré du domaine du Web Usage Mining, nous avons identifié deux conjonctions intéressantes : $C_{MinFreq} \wedge C_{Sim}$ pour les motifs séquentiels d'items et $C_{MinFreq} \wedge C_{ER}$ pour les motifs séquentiels logiques. Nous ne disposons pour l'instant d'aucun d'algorithme permettant d'extraire des motifs vérifiant ces conjonctions. Il faut donc faire appel à d'autres algorithmes classiques sachant gérer l'une des deux contraintes apparaissant dans la conjonction (typiquement la contrainte de fréquence minimale), puis post-traiter les résultats obtenus afin d'éliminer les motifs ne vérifiant pas la deuxième contrainte. Néanmoins, une telle technique présente un intérêt limité : rien ne garantit que la seule contrainte de fréquence minimale sera suffisamment sélective pour assurer la réussite de l'extraction et on n'exploite pas pleinement l'intérêt de la présence de plusieurs contraintes dans l'extraction.

D'une manière générale, il apparaît souvent que, lors d'une extraction de motifs, l'utilisateur possède une connaissance des propriétés des motifs à extraire qui va au delà de la satisfaction d'une seule contrainte (comme la fréquence minimale). Il est alors intéressant d'utiliser des conjonctions ou disjonctions de contraintes primitives dans les requêtes inductives afin de retranscrire cette connaissance *a priori* de l'utilisateur sur ce qu'il recherche. L'utilisation simultanée de plusieurs contraintes peut également permettre de réduire la taille de l'ensemble des résultats. Mais, pour pouvoir exploiter pleinement ces conjonctions de contraintes durant la phase de fouille de données, il est indispensable d'identifier pour les contraintes apparaissant dans la conjonction des caractéristiques intéressantes (comme l'anti-monotonie) permettant de les exploiter de manière active dans un algorithme par niveaux. L'exploitation active de contraintes dans un algorithme par niveaux permet entre autres d'augmenter l'efficacité de l'élagage des motifs candidats générés à chaque itération de ce type d'algorithme. Cela permet ensuite d'accroître la rapidité des calculs de fréquence pour les motifs candidats restants ainsi que de réduire la taille de l'espace de recherche exploré par l'algorithme d'extraction.

Nous proposons donc ici deux nouvelles méthodes afin de traiter les conjonctions de contraintes mises en évidence précédemment. La première de ces méthodes est l'algorithme GALIBOT [CMB02] qui se donne pour but d'extraire des motifs séquentiels vérifiant une conjonction de contraintes $C_{MinFreq} \wedge C_{Sim}$. La deuxième est l'algorithme SPIRIT-L^{0}G [MJ02b] qui lui a pour objectif l'extraction de motifs séquentiels logiques vérifiant $C_{MinFreq} \wedge C_{ER}$.

Afin de valider l'efficacité expérimentale de nos deux algorithmes, nous les testerons sur des jeux de données synthétiques. Ce choix s'explique par le fait que, dans le chapitre précédent , nous avons mis en évidence les conjonctions de contraintes pertinentes sur des scénarios prototypiques pour lesquels nous ne disposons actuellement pas de jeux de données correspondant. Cependant, dans le chapitre suivant, nous montrerons l'utilité du concept de scénario d'évaluation sur un jeu de données réel provenant du monde de la bioinformatique.

4.1 Galibot

4.1.1 Motivations

Dans le chapitre précédent, nous avons mis en évidence l'utilité de l'exploitation d'une conjonction de contraintes de fréquence minimale et de similarité minimale pour l'extraction de motifs séquentiels dans le cadre du Web Usage Mining. Nous rappelons ici la requête 4 du scénario prototypique qui y était présenté :
create pattern set P_2 **as** $s \in P_2 \wedge\ s \in \mathcal{S}(\texttt{Pages}) \wedge\ C_{MinFreq(WUM_3,t_4)}(s) \wedge\ C_{Sim(surf\_ref)}(s)$

Dans ce cas, on cherchait à extraire des parcours, i.e. des motifs séquentiels dont les éléments sont des pages Web, similaires à un certain motif de référence $surf\_ref$ donné par l'administrateur de la salle et ayant une fréquence supérieur à un seuil minimum t_4.

Les contraintes concernant le similarité d'un motif avec un autre motif de références peuvent être utiles dans d'autres domaines d'applications. En biologie, un utilisateur peut ainsi souhaiter rechercher dans une base de données protéiniques des protéines ressemblant à une protéine de référence et susceptibles d'appartenir à la même famille. Une mesure de similarité peut s'appuyer sur une notion de distance, deux séquences pouvant être considérées comme similaires si la distance qui les sépare est faible. De manière informelle, une telle distance sert à quantifier l'effort nécessaire pour "égaliser" deux motifs séquentiels. L'une des plus connues est la distance d'édition introduite dans [Lev66], mais de nombreuses mesures de similarité existent (voir par exemple [Moe00] pour une synthèse récente). Nous proposons donc ici de définir une nouvelle fonction d'évaluation pour le domaine ITEM correspondant à une mesure de similarité entre deux motifs séquentiels. Il sera alors possible de dériver une contrainte de similarité minimale à partir d'une telle fonction.

Comme celle-ci ne présentera pas les propriétés nécessaires pour une exploitation active dans un algorithme par niveaux, nous établirons une version relaxée de cette contrainte, qui présentera l'avantage d'être convertible anti-monotone. Dans [PHL01], Han et al. ont proposé un algorithme par niveau générique pour exploiter de telles contraintes. Nous proposerons donc un algorithme, nommé Galibot, capable d'extraire des motifs séquentiels fréquents dans une base de séquence et similaires à un motif de référence spécifié par l'utilisateur. L'expérimentation de cette technique sur des données synthétiques permet de mettre en évidence les rôles complémentaires que jouent dans l'élagage des candidats les contraintes de fréquence et de similarité minimales.

4.1.2 Définition d'une mesure de similarité

Il est tout d'abord nécessaire de définir une mesure de similarité entre les motifs, l'idée étant d'en étudier ensuite les propriétés afin d'en tirer une contrainte que l'on puisse exploiter activement dans un algorithme d'extraction par niveau. Étant donnés un ensemble d'opérations d'édition possible et des coûts qui leurs sont associés, on souhaite quantifier la similarité de deux

	A	B	C	D	E
A	1	0.01	0.01	0.9	0.01
B	0.01	1	0.01	0.01	0.9
C	0.01	0.01	1	0.01	0.01
D	0.9	0.01	0.01	1	0.01
E	0.01	0.9	0.01	0.01	1

Coûts pour la substitution

	1	2	3
X	0.01	0.75	0.01

Coûts pour la suppression

	0	1	2	3
X	0.01	0.75	0.01	0.9

Coûts pour l'insertion

TAB. 4.1 – Exemples de coûts associés aux différentes opérations

motifs séquentiels. On notera `Items` l'ensemble des items sur lesquels peuvent être construits les motifs séquentiels et $\mathcal{S}(\texttt{Items})$ l'ensemble des motifs séquentiels construits sur `Items`.

Opérations considérées et coûts associés

Soit M_R un motif séquentiel de référence défini par l'utilisateur. L'ensemble des opérations considérés est $O = \{Ins, Del, Sub\}$. Les coûts associés aux opérations dépendent à la fois des symboles sur lesquels elles s'appliquent et de la position relativement au motif de référence. Ces valeurs sont données par l'utilisateur. Les coûts d'insertion et de suppression peuvent être représentés à l'aide d'une matrice. À chaque symbole X d'un motif séquentiel M, pour chaque position $i \in \{0 \dots |M_R|\}$, on associe $c_{Ins}(X, i)$, le coût d'insertion du symbole X à la position i. De même pour la suppression, on associe $c_{Del}(X, i)$ à chaque position $i \in \{1 \dots |M_R|\}$ (il est en effet impossible de supprimer le symbole 0 de M_R). Pour chaque couple ordonné de symboles $(X, Y) \in \Sigma \times \Sigma$, et pour chaque position $i \in \{1 \dots |M_R|\}$, on associe un coût de substitution $c_{Sub}(X, Y, i)$ de X par Y à la position i. Les valeurs des coûts sont comprises dans l'intervalle $[0, 1]$. Par convention, une opération est d'autant plus coûteuse que son coût est proche de 0. Le Tableau 4.1 donne un exemple de coûts associés aux différentes opérations. Dans cet exemple, on considère cependant que les coûts de substitution d'un symbole par un autre sont les mêmes pour toutes les positions et que les coûts d'insertion et de suppression d'un symbole sont indépendants du symbole considéré. Soient $M, M_R \in \mathcal{S}(\texttt{Items})$, un alignement du motif séquentiel M sur le motif de référence M_R est la suite des opérations de substitution, d'insertion ou de suppression qui permettent d'égaliser le motif M_R avec M. Le coût $c(a_{M,M_R})$ d'un alignement possible a_{M,M_R} de M et M_R est donné par le produit des coûts associés à chacune des opérations qu'il comporte.

Calcul du score de similarité d'un motif

Le **score de similarité** d'un motif séquentiel M relativement à un motif de référence M_R, noté $sim(M, M_R)$, est donné par :
$sim(M, M_R) = max\{c(a_{M,M_R}) | a_{M,M_R}$ est un alignement de M et $M_R\}$.

On remarque que la similarité maximale de deux motifs est égale à 1. Nous noterons a^*_{M,M_R} un alignement optimal de M et M_R. L'algorithme permettant de calculer le score de similarité d'un motif M étant donnés un motif de référence M_R et les coûts d'opérations considérés, ressemble beaucoup à celui utilisé pour le calcul de la distance d'édition. Les différences viennent de la prise en compte de coûts dépendant de la position des symboles et de l'initialisation de la cellule (0,0) à 1, étant donné qu'on utilise un critère de maximisation des coûts et non plus de minimisation. Le

		A	B	E	C	D
(0)	1	$\rightarrow$ 0.01	$\rightarrow$ 0.001	$\rightarrow 10^{-4}$	$\rightarrow 10^{-5}$	$\rightarrow 10^{-6}$
A(1)	$\downarrow$ 0.01	$\searrow$ 1	$\rightarrow$ 0.75	$\rightarrow$ 0.56	$\rightarrow$ 0.42	$\rightarrow$ 0.32
B(2)	$\downarrow 7,5.10^{-3}$	$\downarrow$ 0.75	$\searrow$ 1	$\searrow$ 0.67	$\downarrow$ 0.32	$\rightarrow$ 0.24
C(3)	$\downarrow 7,5.10^{-5}$	$\downarrow 7,5.10^{-3}$	$\downarrow$ 0.01	$\searrow$ 0.01	$\searrow$ 0.67	$\rightarrow$ 0.60
sim	$7,5.10^{-5}$	$7,5.10^{-3}$	0.01	0.01	0.67	0.60
sim-pot	1	1	1	0.67	0.67	0.60

TAB. 4.2 – Exemple de calcul du coût de similarité

principe global de l'algorithme consiste à initialiser la matrices des coûts et à calculer ensuite les valeurs de chaque cellule (i, j) sachant qu'elles peuvent se calculer à partir des cellules $(i-1, j-1)$ (en considérant une substitution), $(i-1, j)$ (en considérant une suppression), et $(i, j-1)$ (en considérant une insertion). Pour chaque opération, on tiendra compte des coûts associés avec les symboles concernés. Plus formellement, si M_c est la matrice des coûts considérés pour le calcul du score de similarité, on a pour l'initialisation :

$$M_c(0,0) = 1 \tag{4.1}$$

$$\forall j \in \{1, \ldots, |M|\}, M_c(0,j) = M_c(0,j-1) * C_{ins}(M[j], 0) \tag{4.2}$$

$$\forall i \in \{1, \ldots, |M_R|\}, M_c(i,0) = M_c(i-1,0) * C_{del}(M_R[i], i) \tag{4.3}$$

Et pour le déroulement de l'algorithme, on utilise le fait que :
$\forall i \in \{1, \ldots, |M_R|\}, \forall j \in \{1, \ldots, |M|\}$,

$$M_c(i,j) = max \begin{cases} M_c(i-1,j-1) * C_{sub}(M_R[i], M[j], i) & Substitution(\searrow) \\ M_c(i,j-1) * C_{ins}(M[j], i) & Insertion(\rightarrow) \\ M_c(i-1,j) * C_{del}(M_R[i], i) & Suppression(\downarrow) \end{cases} \tag{4.4}$$

Le score de similarité est alors donné par la cellule $(|M_R|, |M|)$. Un exemple d'un tel calcul est donné dans le tableau 4.2 où l'on voit que le score de similarité de $A \rightarrow B \rightarrow C$ et $A \rightarrow B \rightarrow E \rightarrow C \rightarrow D$ est de 0.6. L'alignement optimal associé est indiqué par les cellules grisées et est constitué des opérations suivantes :

$A \rightarrow B \rightarrow C \overset{Subst}{\Longrightarrow} A \rightarrow B \rightarrow C \overset{Ins}{\Longrightarrow} A \rightarrow B \rightarrow B \rightarrow C \overset{Subst}{\Longrightarrow} A \rightarrow B \rightarrow E \rightarrow C \overset{Subst}{\Longrightarrow} A \rightarrow B \rightarrow E \rightarrow C \overset{Ins}{\Longrightarrow} A \rightarrow B \rightarrow E \rightarrow C \rightarrow D$

Implémentation.

Une implémentation naïve d'un tel algorithme donnerait un coût en $\mathcal{O}(|M|.|M_R|)$ pour le calcul de similarité de chaque motif candidat, mais on sait qu'il peut être implémenté de manière efficace en utilisant un paradigme de programmation dynamique puisque le calcul de la valeur d'une cellule de la matrice ne dépend que de ses trois "voisines". Il est donc possible de calculer de façon incrémentale le score de similarité d'un motif M relativement à M_R en connaissant celui de son préfixe de longueur $|M| - 1$ (pratiquement cela signifie qu'il suffit de connaître la colonne $j-1$ pour déterminer la colonne j de la matrice des coûts). La complexité d'un

tel algorithme est alors en $\mathcal{O}(|M_R|)$. Notons que cet algorithme va être utilisé pour évaluer la mesure de similarité entre les motifs découverts et le motif de référence spécifié par l'utilisateur dans l'algorithme d'extraction que nous proposerons, puisque ce dernier repose sur des classes d'équivalences préfixes.

Pour chaque cellule (i, j), de la matrice des coûts, ($i > 0$ et $j > 0$), nous définissons $o^*(i, j)$ **l'ensemble des opérations optimales élémentaires** permettant de minimiser le coût associé à cette cellule. Nous dirons qu'une opération optimale élémentaire est associée au $(j-1)^{ième}$ symbole d'un motif M et qu'elle s'effectue à la position $i-1$ relativement à M_R. Il existe donc au plus trois opérations optimales élémentaires par cellule. La récursivité d'un algorithme de calcul d'une distance d'édition nous assure que, $\forall i \in \{1, \ldots, |M_R|\}, \forall j \in \{1, \ldots, |M|\}$, chaque élément de $o^*(i, j)$ ne dépend que des éléments de $o^*(i-1, j-1), o^*(i-1, j)$ et $o^*(i, j-1)$. Si $j = 0$ et $i > 0$, alors $o^*(i, 0)$ ne dépend que de $o^*(i-1, 0)$. De même, si $j > 0$ et $i = 0$, alors $o^*(0, j)$ ne dépend que de $o^*(0, j-1)$. Pour j donné, il suffit de connaître les $o^*(i, j-1)$ pour tout $i \in \{0 \ldots |M_R|\}$ pour calculer les $o^*(i, j)$ sous réserve que leurs calculs se fassent dans l'ordre croissant de i. Autrement dit, il nous suffit de connaître les valeurs des coûts associés à chaque cellule du dernier symbole du préfixe de M pour calculer ceux associés à chaque cellule du dernier symbole de M.

4.1.3 Contrainte de Similarité d'un motif

Nous avons défini une fonction d'évaluation sur les motifs séquentiels destinée à quantifier la similarité de deux motifs. Il est maintenant possible d'en dériver une contrainte de similarité.

Définition 31 *Étant donnés un motif de référence M_R , les coûts sur les opérations possibles et un seuil de similarité min-sim $\in [0, 1]$, on dira qu'un motif séquentiel M est **similaire** à M_R si $sim(M, M_R) \geq$ min-sim.*

Nous pouvons alors définir $C_{SIM(M_R)}$, la contrainte de similarité de la façon suivante : un motif séquentiel M satisfait $C_{SIM(M_R)}$ si M est similaire à M_R.

Propriété 1 *$C_{SIM(M_R)}$ n'est ni (convertible) monotone, ni (convertible) anti-monotone.*

Preuve : Reprenons l'exemple du Tableau 4.2, et, pour un seuil de similarité égal à 0,61, prenons $M_R = A \rightarrow B \rightarrow C$ et $M = A \rightarrow B \rightarrow E \rightarrow C \rightarrow D$.

- Pour l'anti-monotonie (convertible) : $M' = A \rightarrow B \rightarrow E \rightarrow C$ est similaire à M_R alors que, par exemple, son préfixe $A \rightarrow B \rightarrow E$ ne l'est pas. Il existe donc un préfixe de M' qui n'est pas similaire à M_R. $C_{SIM(M_R)}$ n'est donc pas (convertible) anti-monotone.
- Pour la monotonie (convertible) : $M' = A \rightarrow B \rightarrow E \rightarrow C$ est similaire à M_R alors que, par exemple, son extension M ne l'est pas. $C_{SIM(M_R)}$ n'est donc pas (convertible) monotone. □

4.1.4 Similarité potentielle d'un motif

La propriété précédente nous assure que $C_{SIM(M_R)}$ ne dispose pas des propriétés suffisantes pour l'exploiter directement et de façon active dans un algorithme par niveau [AIS93, AS94]. Autrement dit, il n'est pas possible de l'utiliser comme critère d'élagage sans porter atteinte à la correction de l'algorithme. Une technique classique pour contourner ce problème consiste à relaxer la contrainte afin d'obtenir une nouvelle contrainte, certes moins sélective, mais qui

présente l'avantage de posséder les propriétés nécessaires pour une exploitation active dans un algorithme par niveaux. Comme cette contrainte relaxée est vérifiée par un plus grand nombre de motifs que ceux qui vérifient la contrainte initiale, il sera donc nécessaire de procéder à un post-traitement des résultats afin d'éliminer les motifs vérifiant la contrainte relaxée mais ne vérifiant pas la contrainte initiale. Il nous faut donc considérer un relâchement de cette contrainte en une contrainte plus faible convertible anti-monotone. C'est pourquoi nous allons introduire la notion de similarité potentielle d'un motif M par rapport à un motif de référence M_R et définir la fonction *sim-pot*(M, M_R), qui calcule la similarité maximale que peut obtenir toute extension équivalente de M.

Définition 32 (Similarité potentielle) *On se donne un motif de référence M_R, les coûts sur les opérations possibles, un seuil de similarité min-sim $\in [0,1]$ et R_p la relation d'ordre "est préfixe de". On définit la similarité potentielle comme suit :*

$$\forall M' \in \mathcal{S}(\texttt{Items}), \textit{sim-pot}(M', M_R) = \max\{sim(M) | M \in \mathcal{S}(\texttt{Items}) \text{ et } R_p(M', M)\}$$

Lemme 1 *Pour tout motif séquentiel M', sim-pot$(M', M_R) \geq sim(M')$.*

Preuve : Il s'agit d'appliquer la définition de cette fonction en considérant le fait que M' est en particulier une extension de M'. □

Propriété 2 *sim-pot est une fonction préfixe monotone décroissante.*

Preuve : Soient M_R, M', M et M'_{pot} des motifs séquentiels tels que : $R_p(M', M), R_p(M', M'_{pot})$ et $sim(M'_{pot}, M_R) =$*sim-pot*(M', M_R).

Raisonnons par l'absurde. Supposons que *sim-pot*$(M, M_R) >$ *sim-pot*(M', M_R). Ceci implique l'existence d'un motif séquentiel M_{pot} tel que : $R_p(M, M_{pot}), sim(M_{pot}, M_R) =$ *sim-pot*(M, M_R) et $sim(M_{pot}, M_R) > sim(M'_{pot}, M_R)$. Or, R_p étant transitive, cette hypothèse est en contradiction avec la définition de la cette fonction. D'où l'absurdité.

Donc, $\forall M_R, M'$ et $M \in \mathcal{S}(\texttt{Items})$, tels que $R_p(M', M)$, on a *sim-pot*$(M, M_R) \leq$ *sim-pot*(M', M_R).

Autrement dit, *sim-pot* est une fonction préfixe monotone décroissante. □

Une manière immédiate de calculer cette fonction *sim-pot* pour un motif M et un motif de référence M_R peut être expliquée de manière très intuitive. Il suffit de prendre la plus grande valeur de la dernière colonne de la matrice des coûts de l'alignement de M et M_R. En effet, il n'est pas possible de faire "augmenter" chacune de ces valeurs puisque l'ajout d'un symbole en queue de M implique la prise en compte d'au moins une opération supplémentaire dont le coût est inférieur ou égal à 1.

Comment assurer la complétude ?

Notre première idée était de définir la similarité potentielle d'un motif comme étant le résultat de la fonction *sim-pot*. Cette fonction étant préfixe monotone décroissante, il est possible d'en dériver une contrainte convertible anti-monotone, dont l'exploitation active au niveau de la phase d'élagage d'un algorithme d'extraction par niveau assurerait sa correction. Cependant, on ne peut pas assurer la complétude. Autrement dit, l'algorithme ne renverrait que des motifs

similaires à un motif de référence mais ne garantirait pas de les renvoyer tous. Pour assurer la complétude de l'algorithme, nous allons définir une nouvelle fonction *sim-pot-comp* basée sur *sim-pot*. L'idée est de considérer la similarité potentielle d'un motif M non plus comme la similarité maximale que peuvent atteindre ses extensions possibles mais comme celle que peuvent atteindre les extensions de son préfixe de longueur $|M| - 1$.

Définition 33 (Nouvelle similarité potentielle) *Soient deux motifs séquentiels M' et M tels que $R_p(M', M)$ et $|M| = |M'| + 1$, la fonction sim-pot-comp est définie par :*
sim-pot-comp(M, M_R) = sim-pot(M', M_R)

Définition 34 *Etant donné un seuil de similarité min-sim, on dira qu'un motif séquentiel M est **potentiellement similaire** à M_R si sim-pot-comp$(M, M_R) \geq$ min-sim. Nous pouvons alors définir la contrainte de similarité potentielle comme suit :*

Un motif séquentiel M satisfait $C_{Sim-pot}(M_R) \Leftrightarrow M$ est potentiellement similaire à M_R.

On en déduit alors les propriétés suivantes :

$$\forall M' \in \mathcal{S}(\texttt{Items}), \textit{sim-pot-comp}(M, M_R) \geq \textit{sim}(M') \quad (1)$$

$$M \text{ est } \textit{similaire} \text{ à } M_R \Rightarrow M \text{ est } \textit{potentiellement similaire} \text{ à } M_R \quad (2)$$

$$C_{SIM}(M_R) \subseteq C_{Sim-pot}(M_R) \quad (3)$$

Preuve :

1. Soient M' et M deux motifs séquentiels tels que $R_p(M', M)$ et $|M| = |M'| + 1$.
   On a *sim-pot-comp*(M, M_R) = *sim-pot*(M', M_R) par définition. Or d'après la propriété 2 et sachant que $R_p(M', M)$, on déduit que : *sim-pot*$(M', M_R) \geq$ *sim-pot*(M, M_R). En appliquant le lemme 1, il vient : $sim - pot(M', M_R) \geq sim - pot(M, M_R) \geq sim(M)$.
2. Immédiat en considérant la propriété précédente.
3. C'est une conséquence immédiate du résultat précédent. □

A partir de la propriété 2, on en déduit très facilement que *sim-pot-comp* est aussi une fonction préfixe monotone décroissante. Cela nous amène immédiatement au fait que $C_{SIM-POT}(M_R)$, qui est une relaxation de la contrainte de similarité, est une contrainte convertible anti-monotone. *A fortiori*, si on considère une base de données de séquences d'entrée D et un seuil de fréquence minimale t, la conjonction de contrainte $C_{MinFreq(D,t)} \wedge C_{Sim-pot(M_R)}$ est elle aussi convertible anti-monotone. Nous pouvons donc réaliser une exploitation active de cette contrainte pour les phases de génération et d'élagage d'un algorithme d'extraction de motifs séquentiels. Dans [PHL01], Han et al. ont proposé un algorithme générique pour utiliser les contraintes convertibles anti-monotone dans les algorithmes par niveaux, nous allons donc l'adapter afin d'obtenir notre algorithme Galibot, qui nous permettra donc, moyennant un petit post-traitement des résultats, de résoudre la requête inductive N° 4 du scénario prototypique en Web Usage Mining présenté au chapitre précédent.

4.1.5 L'algorithme GALIBOT

L'algorithme GALIBOT [1], qui est une adaptation de l'algorithme présenté dans [PHL01], prend en compte la contrainte de similarité définie précédemment, et ressemble dans le principe à l'algorithme cSPADE [Zak00]. En effet, il considère des classes d'équivalence pour réaliser l'extraction, ce qui lui permet de pouvoir parcourir l'espace de recherche en mode "en profondeur d'abord". Nous rappelons rapidement ce que cela signifie. Certains algorithmes d'extraction réalisent un parcours de l'espace de recherche en mode "en largeur d'abord". Cela signifie que l'on analyse en premier lieu tous les motifs de taille 1, puis ceux de taille 2 et ainsi de suite. Dans le mode de recherche "en profondeur d'abord", on explore le domaine de recherche par parties, chacune d'entre elles contenant des motifs appartenant à la même classe d'équivalence (préfixe dans notre cas). Galibot explore donc d'abord tous les motifs séquentiels commenceant par A, puis ceux commenceant par B et ainsi de suite. Par exemple, avec ce mode de recherche, $A \rightarrow C \rightarrow A$ sera analysé avant $B \rightarrow D$ (ce qui n'aurait pas été le cas avec un mode "en largeur d'abord"). De plus, il faut noter que Galibot utilise des listes d'occurrences pour calculer le support des motifs. En revanche, les algorithmes Galibot et c-SPADE diffèrent sur un certain nombre de points fondamentaux :

- Les motifs considérés par GALIBOT sont de largeur 1.
- GALIBOT peut explorer l'espace de recherche en mode "en largeur d'abord" et en mode "en profondeur d'abord".
- GALIBOT ne considère aucune contrainte non préfixe anti-monotone lors de l'élagage. On ne gère donc pas les contraintes *max-gap* et les classes de prédiction. On ne s'intéresse ici qu'aux seules contraintes $C_{MinFreq}$ et $C_{Sim-Pot}$.
- Il utilise la contrainte de similarité potentielle, relaxation de la contrainte de similarité, lors des différentes passes.
- La relation d'ordre considérée est la relation "est préfixe de" et non "est suffixe de". Cela a des conséquences sur la génération des candidats. En effet étant donnés deux motifs $X = M \rightarrow A$ et $Y = M \rightarrow B$ appartenant à la même classe d'équivalence préfixe $[M]$, ainsi qu'un ordre lexicographique sur l'ensemble `Items` (on supposera que pour cet ordre $A < B$), la génération des candidats réalisée via l'équi-jointure à l'intérieur de la classe d'équivalence $[M]$ donnera un nouveau motif candidat : $M \rightarrow A \rightarrow B$.

Pour le comptage des supports des motifs séquentiels, GALIBOT utilise des listes d'occurrences similaires à celles de cSPADE [Zak00]. Nous en rappelons brièvement le principe. Nous associons à chaque motif séquentiel $s \in \mathcal{S}(\texttt{Items})$ une *idlist*, notée $\mathcal{L}(s)$ contenant l'ensemble des couples de la forme (sid, t) correspondant chacun à une occurrence de s dans la séquence d'entrée identifiée par *sid*. Dans chaque couple, t est le temps associé à l'occurrence du dernier item de s pour cette occurrence de s dans la séquence *sid*. Le support de s peut alors être obtenu en comptant le nombre de *sid* différents dans $\mathcal{L}(s)$. La Figure 4.1 montre une base de données composée de 4 séquences d'entrée (en haut à gauche), ainsi que les listes d'occurrences de différents motifs séquentiels. Sur l'exemple, le motif séquentiel $B \rightarrow B$ a un support de 3. Lors de la génération d'un nouveau motif $M \rightarrow X \rightarrow Y$ obtenu à partir de deux motifs $M \rightarrow X$ et $M \rightarrow Y$ partageant un préfixe M en commun, on calcule la nouvelle liste d'occurrence en faisant une jointure temporelle des *idlists* des motifs générateurs. Plus précisément,
$\mathcal{L}(M \rightarrow X \rightarrow Y) = \{(s,t) \in \mathcal{L}(M \rightarrow Y) \mid \exists (s,t') \in \mathcal{L}(M \rightarrow X), t' < t\}$

[1] Nom donné au jeune manoeuvre travaillant au service des voies dans les galeries de mines

SID	EID	Items
1	10	AB
	20	B
	30	AB
2	20	AC
	30	ABC
	50	B
3	10	A
	30	B
	40	A
4	30	AB
	40	A
	50	B

A	
SID	EID
1	10
1	30
2	20
2	30
3	10
3	40
4	30
4	40

B	
SID	EID
1	10
1	20
1	30
2	30
2	50
3	30
4	30
4	50

A→A	
SID	EID
1	30
2	30
3	40
4	40

B→A	
SID	EID
1	10
1	20
3	30
4	30

A → B	
SID	EID
1	20
2	30
2	50
3	30
4	50

B → B	
SID	EID
1	20
1	30
2	50
4	50

A → A → B	
SID	EID
2	50
4	50

FIG. 4.1 – Exemple de listes d'occurrences

Soient D une base de données de séquences d'événements, $C = C_{MinFreq(D,t)} \wedge C_{Sim(M_R)}$ une conjonction de contraintes de fréquence minimale (par rapport à un seuil t défini par l'utilisateur) et de similarité minimale (par rapport à un motif M_R). La tâche d'extraction de l'algorithme Galibot consiste alors à trouver tous les motifs apparaissant dans D qui vérifient C.

Etant donné un motif séquentiel M, voici une liste des notations utilisées dans les algorithmes :

- $[M]$: classe d'équivalence préfixe de M (motifs de $\mathcal{S}(\texttt{Items})$ ayant M pour préfixe).
- $[M]_{=k}$: ensemble de motifs $M' \in \mathcal{S}(\texttt{Items})$ tels que $M' \in [M]$ et $|M| = k$
- $[M]_{\leq k}$: ensemble de motifs $M' \in \mathcal{S}(\texttt{Items})$ tels que $M' \in [M]$ et $|M'| \leq k$
- $C([M])$: ensemble des motifs de $[M]$ apparaissant dans D et vérifiant C
- $C([M]_{=k})$: ensemble des motifs de $[M]_{=k}$ apparaissant dans D et vérifiant C
- $C([M]_{\leq k})$: ensemble des motifs de $[M]_{\leq k}$ apparaissant dans D e vérifiant C

L'algorithme 1 décrit l'algorithme principal de Galibot. On remarque qu'en entrée, on peut spécifier le type de mode de recherche souhaité (en profondeur, DFS, Depth-First Search, ou en largeur, BFS, Breadth-First Search) La première étape consiste à calculer $C([\emptyset]_{=1})$, l'ensemble des 1-motifs vérifiant C. Ce calcul nécessite une première passe sur D pour déterminer les listes d'occurrence (id-lists) de chaque 1-motif. La seconde étape construit E, l'ensemble des classes d'équivalence ayant un 1-motif M pour préfixe, chacune de ces classes ne comporte qu'un motif, le 1-motif préfixe de la classe. Enfin, l'étape suivante appelle la fonction *Calculer-Motifs-Contraints* pour chaque classe d'équivalence de E. Elle est détaillée dans l'algorithme 2.

Algorithme 1: Algorithme principal de Galibot

Données : D une base de séquence, C une conjonction de contraintes
Mode-Rech le type de mode de recherche
Résultat : L'ensemble des motifs séquentiels de D vérifiant C
$C([\emptyset]) = \emptyset$;
Calculer $C([\emptyset]_{=1})$;
$C([\emptyset]) = C([\emptyset]_{=1})$;
Calculer $E = \{[M]_{=1} | M \in C([\emptyset]_{=1})\}$;
pour i allant de 1 à $|E|$ **faire**
  $C([\emptyset]) = C([\emptyset])+$ *Calculer-Motifs-Contraints*$(E(i), C,$ *Mode-Rech*$)$
fin
retourner $C([\emptyset])$

L'algorithme *Calculer-Motifs-Contraints* prend en entrée le mode de recherche, la conjonction de contraintes C et une classe d'équivalence $[M]_{=k}$ préfixe de motifs de longueur k. La première boucle sur i parcourt tous les motifs M_i de $[M]_{=k}$. Pour chaque M_i qui admet au moins une extension de longueur $k+1$ satisfaisant C, on crée une nouvelle classe d'équivalence $[M_i]_{=k+1}$. La génération des candidats (appel à la fonction Genere-Candidat) se fait en suivant la procédure décrite dans [Zak01]. Plus précisément, on ajoute le dernier symbole de M_z à la fin de M_i et on obtient alors un nouveau motif R. Si R n'est pas élagué (la procédure d'élagage est celle donnée par l'algorithme 3) et est fréquent, on peut ajouter la classe d'équivalence $[M_i]_{=k+1}$ à E. On calcule ensuite la liste d'occurrences du motif R à l'aide des listes d'occurrences des deux motifs qui ont permis de le générer. Cela est réalisé grâce à la fonction $Jointure - Contrainte$ qui est l'adaptation directe de la fonction $Constrained - Temporal - Join$ présentée dans C-SPADE [Zak00], puis, on ajoute R à $[M_i]_{=k+1}$. Ensuite, si la contrainte de similarité est vérifiée par R, on ajoute R à $C([M]_{=k+1})$. Notons que si R n'a pas été élagué, cela signifie que R satisfait $C_{Sim-Pot(M_R)}$, nous devons donc garder R pour assurer la complétude de la génération des candidats possibles. De même, si R ne satisfait pas la contrainte de similarité, on ne peut l'inclure dans l'ensemble des motifs satisfaisant C sans porter atteinte à la correction de l'algorithme. Lorsqu'on sort de la deuxième boucle Pour, tous les motifs possibles ayant M_i pour préfixe ont été traités. Si le mode de recherche est en profondeur d'abord, on appelle récursivement la procédure avec la classe $[M_i]_{=k+1}$ en paramètre. Dans le cas contraire, on passe au motif M_i suivant. Lorsqu'on sort de la première boucle, tous les motifs de longueur $k+1$ satisfaisant C ont été traités. On renvoie donc cet ensemble. Si le mode de recherche est en largeur d'abord, on appelle récursivement la procédure sur chaque élément de E.

Pour l'algorithme d'élagage, on regarde si la contrainte $C_{Sim-pot}$ est violée par R. Comme cette contrainte est convertible anti-monotone, cela nous assure que la stratégie d'élagage est correcte [PHL01]. Cependant, elle n'est pas complète car elle n'élague pas tous les motifs de R dont on peut savoir *a priori* qu'ils ne satisfont pas une des contraintes gérées. En effet, nous ne procédons pas à l'élagage sur la contrainte de fréquence, i.e., on ne teste pas s'il existe un sous-motif de R qui ne soit pas fréquent. La justification est double : d'une part, comme tous les sous-motifs de R ne sont pas dans la classe d'équivalence préfixe de R, nous perdons en partie l'utilité de cet élagage, d'autre part, d'après Zaki dans [Zak01], la gestion de cette contrainte quand d'autres contraintes sont présentes pour l'élagage n'améliore pas significativement l'efficacité de l'algorithme en pratique.

Algorithme 2: Pseudo-code de Calculer-Motifs-Contraints

Données : $[M]_{=k}$ une classe d'équivalence préfixe
C une conjonction de contraintes
Mode-Rech le type de mode de recherche
Résultat : $C([M]_{=k})$
pour i *allant de* 1 *à* $|[M]_{=k}|$ **faire**
  $M_i = [M]_{=k}(i)$;
  $[M_i]_{=k+1} = \emptyset$;
  pour z *allant de* 1 *à* $|[M]_{=k}|$ **faire**
    $M_z = [M]_{=k}(z)$;
    $R = Genere - Candidat(M_i, M_z)$;
    si $Elaguer(R, C) = Faux$ **alors**
      $R.idlists = Jointure - Contrainte(M_i, M_z, C)$;
      si $C_{MinFreq(D,t)}(R)$ **alors**
        si $[M_i]_{=k+1} = \emptyset$ **alors** $E = E + [M_i]_{=k+1}$;
        $[M_i]_{=k+1} = [M_i]_{=k+1} + R$;
        si $C_{SIM(M_R)}(R)$ **alors** $C([M]_{=k+1}) = C([M]_{=k+1}) + R$;
      fin
    fin
  fin
  si *Mode-Rech=DFS et* $[M_i]_{=k+1} \neq \emptyset$ **alors**
    Calculer-Motifs-Contraints($[M_i]_{=k+1}$,C,*Mode-Rech*)
  fin
fin
retourner $C([M]_{=k+1})$;
si *Mode-Rech=BFS et* $E \neq \emptyset$ **alors**
  pour $m = 1$ *à* $|E|$ **faire**
    Calculer-Motifs-Contraints($E(i)$,C,*Mode-Rech*)
  fin
fin

4.1.6 Résultats expérimentaux

Les expérimentations ont été effectuées sur un PC sous Windows NT 4.0 équipé d'un processeur Pentium III de 800 MHz et de 512 Mo de mémoire et l'implémentation a été réalisée en JAVA. Les trois jeux de données utilisés lors de l'extraction ont été obtenus à l'aide du générateur aléatoire proposé dans le projet Quest (`www.almaden.ibm.com/cs/quest/index.html`). Chacun d'entre eux correspond à un ensemble de séquences d'entrée. Chacune de ces séquences d'entrée peut par exemple correspondre aux différents achats d'un même client dans un supermarché au fil du temps. La signification des différentes variables (D, C, T, S, N_S, N) est donnée dans le Tableau 4.3. Les paramètres retenus pour les expérimentations sont donnés dans le Tableau 4.4.

Les expérimentations visent à comparer les performances de GALIBOT lorsque l'on exploite les contraintes de façon active et lorsque l'on adopte une stratégie post-traitement. Nous avons retenu trois critères pour l'évaluation des résultats des extractions : le temps d'exécution, le nombre de motifs séquentiels renvoyés et le nombre de candidats générés. Ensuite, nous étudions

Algorithme 3: Pseudo-code de la procédure Elaguer(R,C)

Données : R un motif séquentiel
C une conjonction de contraintes
Résultat : *true* si R peut R peut être élagué
$false$ sinon
$elaguer = faux$;
si $non(C(R))$ **alors** $elaguer = vrai$;
retourner *elaguer*

Variable	Signification
D	Nombre de séquences d'événement en entrée
C	Nombre moyen de transactions par client
T	Nombre moyen de symboles (items) par transaction
S	Longueur moyenne des motifs séquentiels potentiellement fréquents
N	Nombre total de symboles (items)

TAB. 4.3 – Signification des paramètres du générateur

la sélectivité des contraintes de support et de similarité. Étant donné le nombre de degrés de liberté induits par la contrainte de similarité (valeurs des coûts d'opérations, seuils de similarité possibles) nous avons choisi de simplifier les tests expérimentaux en ne considérant que des coûts uniformes (tous les coûts spécifiés sont égaux) et d'étudier le comportement des extractions pour divers seuils de similarité. Les opérations possibles sont l'insertion et la suppression d'un symbole. Les coûts étant égaux, le seuil de similarité correspond alors à un nombre d'opérations tolérées pour chaque motif. Les motifs de référence spécifiés ont été choisis parmi les motifs fréquents retournés lors des extractions afin d'être assuré qu'ils apparaissent dans les données pour un seuil de fréquence donné. Enfin, l'implémentation de Galibot que nous avons réalisée ne prend en compte que le parcours de l'espace de recherche en mode en profondeur d'abord.

Les résultats sont donnés sur la figure 4.2. Lorsque le seuil de similarité vaut 0, cela signifie que la contrainte n'est pas gérée (il s'agit de la base de l'évaluation de la stratégie post-traitement). Une valeur de 0,5 indique que l'on tolère au plus une seule opération d'insertion ou de suppression sur les motifs à retrouver, une valeur de 0,25 correspondant à deux opérations. Enfin, une valeur de 1 indique qu'aucune opération n'est tolérée, i.e., le seul motif qui peut être retourné est le motif de référence.

Le constat général est que plus le seuil de similarité est fort, plus les performances de l'extraction sont améliorées. Nous pouvons remarquer que l'évolution du nombre de candidats générés et

Fichier	$\|D\|$	$\|C\|$	$\|T\|$	$\|S\|$	N_S	$\|N\|$	Taille	Nb. Evts
F1	25k	10	10	10	5k	100	11 Mo	790k
F2	50k	10	10	10	5k	100	23 Mo	1,5M
F3	100k	10	10	10	5k	2k	54 Mo	3,2M

TAB. 4.4 – Fichiers utilisés pour les tests

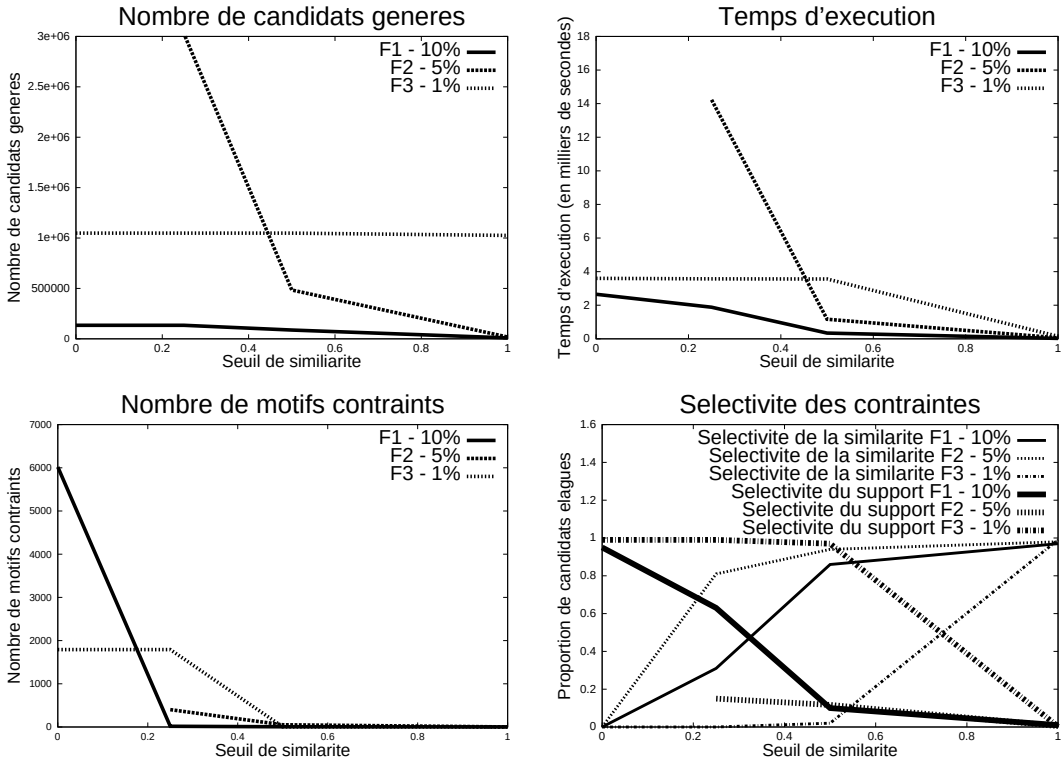

FIG. 4.2 – Résultats des extractions sur les trois fichiers

le temps d'exécution varient de façon similaire. Il est à noter une exception à ce principe lorsque l'on compare les valeurs prises par les extractions sur F3 pour un seuil de similarité de 1. En effet, les performances augmentent significativement alors que le nombre de candidats générés varie peu. Ceci s'explique par la forte chute de la sélectivité de la contrainte de support et l'augmentation de celle de similarité. Or, le coût de l'évaluation de la première est nettement plus important que celui de la seconde. Enfin, on peut remarquer que la sélectivité de la contrainte de similarité est inversement proportionnelle à celle de la contrainte de support. Globalement les résultats obtenus sont très encourageants et démontrent la pertinence de la considération de contraintes fondées sur des notions de similarité.

4.1.7 Synthèse

Nous avons donc vu dans cette première partie un algorithme pour l'extraction de motifs séquentiels fréquents vérifiant une contrainte de similarité. L'utilisation d'une telle conjonction de contraintes peut être utile dans de nombreux contextes applicatifs : recherche de parcours d'utilisateurs similaires à un chemin de référence dans le cadre du Web Usage Mining, recherche de protéines similaires à une structure connue en protéonomique, etc. Malheureusement, les mesures utilisées pour quantifier la similarité de deux motifs séquentiels permettent rarement d'en dériver des contraintes anti-monotones que l'on sait pleinement exploiter dans des algorithmes d'extraction par niveaux. C'est le cas de la mesure de similarité que nous avons définie, où notre conjonction de contraintes $C_{MinFreq} \wedge C_{Sim(M_R)}$ n'était pas anti-monotone. Afin de contourner

cette limitation, nous avons mis en place une stratégie classique : étudier les possibilités de relaxer la contrainte de similarité minimale qui ne possède pas de propriétés "sympathiques". Cela signifie en fait découvrir une nouvelle contrainte C' moins sélective que la contrainte initiale mais possédant de "bonnes" propriétés pour l'exploiter dans un algorithme d'extraction. Il faut bien sûr pour cela que tout motif vérifiant le contrainte initiale $C_{Sim(M_R)}$ vérifie la contrainte C'. Néanmoins, la réciproque n'étant pas nécessairement vraie, il sera nécessaire de post-traiter les motifs obtenus afin d'éliminer ceux vérifiant C' mais pas $C_{Sim(M_R)}$. L'intérêt de l'approche utilisée ici est de permettre d'exploiter au moins partiellement une contrainte (ici celle de similarité minimale) quand bien même celle-ci ne présente pas les propriété requises pour une prise en compte optimale dans un algorithme d'extraction classique. Les résultats expérimentaux confortent cette approche et montrent également que, dans le cas qui nous intéresse, la sélectivité des deux contraintes mises en jeu (similarité minimale et fréquence minimale) se complètent utilement. Néanmoins, il faut noter que d'une manière générale, il n'est pas forcément trivial de trouver de "bonnes" relaxations de contraintes non monotones ou anti-monotones. Dans la deuxième partie de ce chapitre, nous allons présenter un algorithme faisant intervenir des relaxations de contraintes syntaxiques dans le domaine des motifs séquentiels logiques.

4.2 SPIRIT-LoG

4.2.1 Motivations

Dans le Chapitre 3, nous avons présenté un scénario prototypique dans le domaine du Web Usage Mining. Parmi toutes les requêtes inductives qui le composent, nous avons mis en évidence une conjonction de contraintes pertinente à étudier pour l'extraction de motifs séquentiels logiques : celle qui fait intervenir à la fois une contrainte de fréquence minimale et une contrainte syntaxique exprimée sous la forme d'une expression rationnelle $C = C_{MinFreq} \wedge C_{ER}$. Nous rappelons ici la requête inductive N° 10 qui met en oeuvre cette contrainte :
create pattern set P_4 **as** $s \in P_4 \wedge C_{ER(ref)(s)} \wedge C_{MinFreq(WUM_{10},t_7)}$
ref est une expression rationnelle donnée par l'utilisateur qui permet de préciser l'aspect des motifs séquentiels logiques à extraire et t_7 est le seuil de fréquence minimale.

D'une manière générale, il existe de nombreux cas où l'usage de motifs séquentiels logiques est plus intéressant que celui des motifs séquentiels classiques construits sur des items. Ce sera souvent le cas dès qu'on disposera d'une information ordonnée (dans le temps ou l'espace) et exprimée en relation. Or, on peut trouver ce genre d'information dans des jeux de données constitués de plusieurs tables liées par des relations et où certains champs servent à trier les tuples par date ou par position. Le fait d'utiliser des extracteurs de motifs séquentiels d'items classiques (d'ordre 0) pour travailler sur ces données nécessite une forme de propositionalisation et du coup, le lien relationel entre les différents tuples des tables ne peut être exploité durant l'extraction. Il peut toutefois être retrouvé par post-traitement des résultats. Avoir des extracteurs permettant de travailler sur des motifs séquentiels logiques permet de conserver cette information relationnelle durant le processus de fouille. D'une manière générale, l'utilité de la Programmation Logique Inductive [MR94] étant maintenant démontrée dans le domaine de la Fouille de Données Multi-Relationnelles [DT99, BR98], il nous semble pertinent d'essayer de s'attaquer au problème de l'extraction de séquences composées d'atomes, faisant intervenir des variables permettant la généralisation de certaines constantes. Il existe de nombreux domaines d'applications dans lesquels l'utilisation de littéraux de la logique du premier ordre pour expri-

mer de l'information en relation est pertinente. Prenons par exemple la découverte de scripts shell à partir des logs d'un système Unix [JB01] et supposons que nous avons une base de données contenant les commandes tapées par certains utilisateurs comme décrit dans le Tableau 4.5. En transformant ces commandes en littéraux de la logique du premier ordre, nous obtenons l'ensemble de séquences logiques en entrée présenté dans la Table 4.6.

En utilisant un seuil de fréquence minimale fixé à 30 % par l'utilisateur, les algorithmes que nous présentons dans cette partie peuvent extraire des motifs séquentiels logiques de la forme : $\langle mv(F, D)\ cd(D)\ emacs(F)\ gcc(F, E)\ exec(E)\ \rangle$, où F, D, E sont des variables logiques. En effet, ce motif séquentiel logique apparaît deux fois parmi les cinq séquences d'entrée. Il est alors facile de comprendre l'intérêt des motifs séquentiels logiques dans la conception d'un shell intelligent qui pourrait guider l'utilisateur dans une série de commandes qu'il soumet au système d'exploitation. On peut aussi imaginer qu'une séquence de commandes fréquentes puisse être réalisée automatiquement. Néanmoins, comme rechercher tous les motifs séquentiels logiques noierait l'utilisateur sous un trop grand nombre de résultats, il est nécessaire de considérer une autre contrainte comme par exemple une contrainte syntaxique. Une contrainte de ce type très efficace est celle proposée par Garofalakis dans les algorithmes SPIRIT [GRS02, GRS99] qui utilise une expression régulière pour spécifier l'ordre et la nature des items apparaissant dans des motifs séquentiels fréquents. Notre but ici peut donc être présenté comme l'extension de ces algorithmes au domaine des motifs séquentiels logiques.

4.2.2 Formalisation du problème

Plus formellement, le problème de l'extraction de motifs séquentiels logiques sous une contrainte de fréquence minimale et d'expression régulière peut alors se décrire comme suit :

- **Etant donnés** $\mathcal{D}$, une base de données de séquences d'entrée, un seuil de fréquence minimum *minsup* spécifié par l'utilisateur, et une contrainte syntaxique donnée sous forme d'expression rationnelle $\mathcal{R}$ (ou sous la forme de l'automate fini déterministe associé $\mathcal{A}$).
- **Trouver** tous les motifs séquentiels logiques valides et fréquents dans $\mathcal{D}$.

Trouver un tel algorithme permettra alors de résoudre directement la requête N° 10 de notre scénario prototypique en Web Usage Mining.

User 1	User 2	User 3	User 4	User 5
`mv ex.c src/`	`mv f1 f2`	`mail`	`mv mn.c prj/`	`cd docs`
`mail`	`cd /tmp`	`ftp srv1`	`cd prj/`	`gv f1.ps`
`cd src/`	`rm train.dat`	`xcdroast`	`lpq`	`gv f2.ps`
`emacs ex.c`	`cd ~`	`logout`	`emacs mn.c`	`lpr f2.ps`
`talk user2`	`emacs n.tex`		`gcc mn.c -o prj`	`cd ../progs`
`gcc ex.c -o ex`	`latex n`		`./prj`	`make`
`lpq`	`dvips n.tex n.dvi`		`logout`	`proto`
`./ex`	`gv n.ps`			`more res`
`gprof ex`	`lpr n.ps`			`logout`
`logout`	`logout`			

TAB. 4.5 – Exemple de commandes Unix

User 1	User 2	User 3	User 4	User 5
mv(ex.c,src/)	mv(f1,f2)	exec(mail)	mv(mn.c,prj/)	cd(docs)
exec(mail)	cd(/tmp)	ftp(srv1)	cd(prj/)	gv(f1.ps)
cd(src/)	rm(train.dat)	exec(xcdroast)	exec(lpq)	gv(f2.ps)
emacs(ex.c)	cd(∼)	exec(logout)	emacs(mn.c)	lpr(f2.ps)
talk(user2)	emacs(n.tex)		gcc(mn.c,prj)	cd(../progs)
gcc(ex.c,ex)	latex(n)		exec(prj)	exec(make)
exec(ex)	dvips(n.tex,n.dvi)		exec(logout)	exec(proto)
gprof(ex)	gv(n.ps)			more(res)
exec(lpq)	lpr(n.ps)			exec(logout)
exec(logout)	exec(logout)			

TAB. 4.6 – Séquences de littéraux en entrée

4.2.3 Propriétés de la conjonction de contraintes

Nous nous intéressons donc à une conjonction de contraintes $C = C_{MinFreq(\mathcal{D},minsup)} \wedge C_{ER(\mathcal{R})}$. Afin de savoir quelle stratégie algorithmique nous allons utiliser, nous allons d'abord observer les propriétés de ces contraintes.

La contrainte de fréquence minimale est anti-monotone et cette propriété est conservée lorque l'on travaille sur des motifs séquentiels logiques.
Preuve : Soit s un motif séquentiel logique fréquent dans une base de données de séquences d'entrée $\mathcal{D}$ par rapport à un seuil de fréquence minimale t. Soient Θ l'ensemble des substitutions $\{\theta_1, \ldots, \theta_n\}$ qui font réussir s sur $\mathcal{D}$. Si on considère s' un sous-motif séquentiel logique de s, alors l'application de toutes les substitutions contenues dans Θ sur s' permet d'obtenir pour s' une fréquence supérieure ou égale à celle de s dans $\mathcal{D}$. Et comme s vérifiait $C_{MinFreq(\mathcal{D},t)}$, alors s' vérifie aussi $C_{MinFreq(\mathcal{D},t)}$. □

En revanche la contrainte de validité syntaxique exprimée sous forme d'expression rationnelle n'est ni monotone, ni anti-monotone.
Preuve : Prenons par exemple une expression rationnelle $\mathcal{R} = p_1(p_2|p_3)^*p_4$. Pour des raisons de commodité, considérons que tous les symboles de prédicat sont d'arité 1. Soit s le motif séquentiel logique suivant :$\langle p_1(X)\ p_3(Y)\ p_2(T)\ p_2(T)\ p_4(X)\rangle$. Il apparaît que s vérifie $C_{ER(\mathcal{R})}$. Prenons alors le sous-motif séquentiel logique s' de s suivant : $s' = \langle p_1(X)\ p_3(Y)\ p_2(T)\ p_2(T)\rangle$. On constate alors que s' ne vérifie pas $C_{ER(\mathcal{R})}$ Donc $C_{ER(\mathcal{R})}$ n'est pas anti-monotone

De même, si on considère un motif séquentiel logique $s = \langle p_1(X)\ p_4(Y)\rangle$, alors s vérifie $C_{ER(\mathcal{R})}$. Considérons alors $s' = \langle p_1(X)\ p_4(Y)\ p_2(X)\rangle$, alors s' est un sous-motif séquentiel logique de s et pourtant s' ne vérifie pas $C_{ER(\mathcal{R})}$. Donc $C_{ER(\mathcal{R})}$ n'est pas monotone. □

Par extension, la conjonction de contraintes qui nous intéresse ici n'est donc ni monotone ni anti-monotone, il n'est donc pas possible de l'exploiter directement dans un algorithme par niveaux. Une solution possible est alors d'essayer de relaxer la contrainte $C_{ER(\mathcal{R})}$ en une contrainte moins forte mais possédant des propriétés plus sympathiques pour l'extraction de motifs. Dans [GRS02], Minos N. Garofalakis a proposé la famille d'algorithmes SPIRIT pour l'extraction de motifs séquentiels sous une conjonction de contraintes $C_{MinFreq(\mathcal{D},minsup)} \wedge C_{ER(\mathcal{R})}$, qui consistait en l'usage de relaxations plus ou moins forte de la contrainte de validité par rapport à une

expression rationnelle. Nous proposons donc ici d'adapter et d'étendre ces algorithmes au cadre de l'extraction de motifs séquentiels logiques.

4.2.4 SPIRIT-LOG pour l'extraction de motifs séquentiels logiques fréquents

4.2.5 Algorithme générique de SPIRIT-LOG

L'algorithme 4 donne la structure générale du processus d'extraction de motifs séquentiels logiques fréquents vérifiant une contrainte syntaxique sur la structure des motifs spécifiée sous forme d'expression rationnelle. Cet algorithme est l'adaptation directe de celui présenté dans [GRS99, GRS02] qui a pour but d'extraire des motifs séquentiels constitués d'items qui vérifient la même conjonction de contraintes. L'algorithme consiste en un parcours par niveaux de l'espace de recherche qui extrait à chaque étape des motifs séquentiels logiques de plus en plus longs. C_k dénote l'ensemble des k-motifs séquentiels logiques candidats et F_k dénote l'ensemble des k-motifs séquentiels logiques fréquents vérifiant la conjonction de contraintes. $\mathcal{D}$ désigne la base de séquence d'entrée, $C_{ER(\mathcal{R})}$ est le contrainte syntaxique sous forme d'une expression rationnelle $\mathcal{R}$, C' une version relaxée de la contrainte $C_{ER(\mathcal{R})}$ et *minsup* correspond au seuil pour la contrainte de fréquence minimale. En sortie, l'algorithme nous renvoie donc l'ensemble de tous les motifs séquentiels logiques vérifiant la contrainte de fréquence minimale $C_{MinFreq(\mathcal{D},minsup)}$ et la contrainte syntaxique sous forme d'expression rationnelle $C_{ER(\mathcal{R})}$.

Dans SPIRIT, le calcul de l'ensemble F_1 se faisait en vérifiant la conjonction de contrainte $C' \wedge C_{MinFreq(\mathcal{D},minsup)}$ pour tous les motifs séquentiels de longueur 1 (c'est-à-dire les items) apparaissant dans le jeu de données en entrée. Pour adapter cette première phase aux cas des motifs séquentiels logiques, nous avons procédé comme suit : nous considérons tous les symboles de prédicats apparaissant dans le jeu de données en entrée. Pour chaque symbole de prédicat, on considère ensuite tous les atomes libres (i.e. où apparaissent uniquement des variables en paramètres) pouvant être construits dessus. Plus précisément, supposons un symbole de prédicat p d'arité n, nous considérons alors n variables $V = \{X_1, \ldots, X_n\}$. Pour construire un atome libre basé sur p, nous avons alors le choix entre n variables pour le premier paramètre, n pour le second, etc, jusqu'à n pour le dernier. Une telle solution naïve donnerait un résultat en n^n ce qui est beaucoup trop. Si on regarde plus précisément, on s'aperçoit que de nombreux motifs ainsi considérés sont équivalents à un renommage près. En effet, si on prend $n = 3$, les motifs séquentiels $\langle p(X_1, X_1, X_3)\rangle$ et $\langle p(X_2, X_2, X_1)\rangle$ sont équivalents à un renommage près. Pour construite les candidats, il vaut mieux donc considérer les atomes libres que l'on peut construite en prenant 1 variable dans V, puis 2 variables dans V, ..., jusqu'à prendre n variables dans V. Le nombre de candidats que l'on peut construire en prenant k variables dans V ($k \leq n$) est C_n^k. Le nombre de candidats que l'on va construire pour un symbole de prédicat p d'arité n sera donc : $C_n^1 + C_n^2 + \ldots + C_n^n = 2^n - 1$ Pour construire les candidats basés sur un symbole de prédicat d'arité n, il suffira donc de faire l'énumération de toutes les combinaisons possibles de k parmi n variables ($k \leq n$).

Comme la contrainte syntaxique sous forme d'expression régulière $C_{ER(\mathcal{R})}$ n'est généralement pas anti-monotone, il est impossible d'en faire une exploitation active comme dans le paradigme classique des extractions par niveau. C'est pourquoi Garofalakis a proposé d'utiliser différentes relaxations C' de $C_{ER(\mathcal{R})}$ [GRS02] pour lesquels il est possible de concevoir des fonctions de génération et d'élagage *ad hoc* qui permettent de faire un usage efficace de C'. Nous remarquons cependant que nous avons un algorithme qui retourne plus de résultats que ce

Algorithme 4: Algorithme générique de SPIRIT-LºG pour l'extraction de motifs séquentiels logiques

Données : $\mathcal{D}$ une base de données de séquences d'entrée,
$C_{ER(\mathcal{R})}$ une contrainte spécifiée sous forme d'expression rationnelle
$minsup$ un seuil de fréquence minimum

Résultat : F l'ensemble de tous motifs séquentiels logiques vérifiant $C_{ER(\mathcal{R})}$

C' = une relaxation de la contrainte $C_{ER(\mathcal{R})}$;
$F = F_1 = \{$ Les motifs séquentiels logiques fréquents de longueur 1 vérifiant C' $\}$;
répéter
  // Génération des candidats à l'aide de C'
  Utiliser C' et F pour générer $C_k :=$ {motifs séquentiels logiques candidats de longueur k vérifiant C'};
  // Elagage des candidats à l'aide du support
  $P_k = \{s \in C_k, \exists s'$ un sous-motif séquentiel loqique de s qui vérifie C' et qui ne s'unifie avec aucun motif séquentiel logique de $F\}$;
  $C_k = C_k - P_k$;
  // Comptage des candidats
  Parcourir $\mathcal{D}$ pour calculer le support des motifs séquentiels logiques de C_k;
  $F_k =$ motifs séquentiels logiques de C_k;
  $F = F \cup F_k$;
  $k \leftarrow k + 1$;
jusqu'à StopCondition(F, C');
// Post traitement de F : on renforce la contrainte en réintroduisant $C_{ER(\mathcal{R})}$
Retourner les motifs séquentiels logiques de F qui vérifient $C_{ER(\mathcal{R})}$;

qui était initialement attendu, étant donné qu'il considère une conjonction de contrainte $C' \wedge C_{MinFreq(\mathcal{D},minsup)}$. Les motifs séquentiels recherchés qui vérifient la conjonction de contraintes $C_{ER(\mathcal{R})} \wedge C_{MinFreq(\mathcal{D},minsup)}$ sont parmi ces résultats. Une étape de post-traitement supplémentaire est donc nécessaire si l'on souhaite conserver uniquement les résultats pertinents.

Comme pour SPIRIT, nos algorithmes utilisent deux types d'élagage. En effet, compter le support des motifs séquentiels candidats est une phase coûteuse, il est donc nécessaire d'élaguer l'ensemble des motifs candidats, pour cela, nous utilisons :

- *Elagage basé sur C'* qui utilise la relaxation C' de la contrainte $\mathcal{C}_{ER(\mathcal{R})}$ donnée par l'utilisateur, et qui permet de ne générer que des candidats vérifiant cette contrainte.
- *Elagage basé sur le support* qui consiste à s'assurer que tous les sous-motifs séquentiels logiques d'un motif candidat de C_k vérifiant C' font partie de l'ensemble F des motifs séquentiels logiques fréquents déjà découverts. Nous pouvons uniquement vérifier ce point pour les sous-motifs vérifiant eux-mêmes la contrainte C', étant donné que seuls ceux-ci peuvent éventuellement apparaître dans F.

Preuve. Montrons que la procédure d'élagage (représentée par le calcul de l'ensemble de candidats à élaguer P_k dans l'algorithme générique) est correcte et complète.

Montrons tout d'abord la correction de l'élagage. Pour cela, il nous faut montrer qu'à chaque itération k de l'algorithme, P_k ne contient que des motifs séquentiels logiques non fréquents. Raisonnons par l'absurde, soit $s \in P_k$ et supposons s fréquent. Prenons alors s' un sous-motif

séquentiel logique de s vérifiant C'. Par anti-monotonie de la contrainte de fréquence, s' vérifie $C_{MinFreq(\mathcal{D},minsup)}$. Soit $p = |s'|$, nous avons $p \leq k-1$. Comme s' vérifie $C' \wedge C_{MinFreq(\mathcal{D},minsup)}$, cela signifie que s' appartient à F (à un renommage près), i.e. qu'il existe $t \in F$, tel qu'il existe une substitution θ, telle que $t = \theta s$. Par définition de P_k, cela signifie que $s \notin P_k$. Ce qui est absurde. Donc s ne peut être fréquent et P_k ne contient que des motifs séquentiels logiques non fréquents.

Montrons la complétude de la procédure d'élagage des candidats. Soit k un entier et $P(k)$ la propriété suivante : "Tous les sous-motifs séquentiels logiques des éléments de $C_k \backslash P_k$ qui vérifient C' vérifient aussi $C_{MinFreq(\mathcal{D},minsup)}$".

Montrons que $P(2)$ est vraie. Soit $s \in C_2 \backslash P_2$, on note que s vérifie C' car $s \in C_2$ et les candidats générés vérifient C' (ce point sera démontré plus tard lors de la présentation des 4 variantes de génération de candidats de SPIRIT-LºG). Soit s' un sous-motif séquentiel logique vérifiant C' de longueur 1. Alors il existe $t \in F_1$ tel qu'il existe une substitution Θ telle que $t = \Theta s'$. Comme t est fréquent, il existe un ensemble de substitutions $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$ qui font réussir t sur $\mathcal{D}$. Prenons alors l'ensemble des substitutions $\Sigma' = \{X \mid Z, \{X \mid Y\} \in \Theta$ et $\{Y \mid Z\} \in \Sigma\}$, alors Σ' permet de faire réussir s' sur D. Donc s' est fréquent et $P(2)$ est vraie.

Passons à la démonstration de la phase de récurrence, supposons $P(n)$ vraie pour tout entier $k \in \{1, \ldots, n\}$, et montrons que $P(n+1)$ est vraie. Soit $s \in C_{n+1} \backslash P_{n+1}$. Soit s' un sous-motif séquentiel logique de s vérifiant C'. Posons $p = |s'|$ alors $p \leq n$. Par définition de P_{n+1}, il existe $t \in F_p$ tel qu'il existe une substitution Θ tel que $t = \Theta s'$. Comme $t \in F_p$, t est fréquent donc il existe un ensemble de substitutions $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$ qui font réussir t sur $\mathcal{D}$. Prenons alors l'ensemble des substitutions $\Sigma' = \{X \mid Z, \{X \mid Y\} \in \Theta$ et $\{Y \mid Z\} \in \Sigma\}$, alors Σ' permet de faire réussir s' sur D. Donc s' est fréquent et $P(n+1)$ est vraie. □

L'idée de base demeure que l'élagage à base de la contrainte syntaxique essaie de réduire la taille de l'espace de recherche C_k en renforçant la contrainte C', alors que l'élagage à base de support essaie de réduire la taille de C_k en vérifiant que le seuil de fréquence minimal est vérifié pour les sous-motifs séquentiels appropriés.

Nous montrons maintenant comment nous avons adapté les quatre versions de SPIRIT en une nouvelle famille d'algorithmes nommée SPIRIT-LºG pour l'extraction de motifs séquentiels logiques fréquents.

4.2.6 Les quatre variantes de SPIRIT-LºG

Etant donné que nous avons décidé d'adapter la contrainte d'expression rationnelle proposée par Minos N. Garofalakis dans SPIRIT au cas des motifs séquentiels logiques, nous considérons une expression rationnelle $\mathcal{R}$ définie par l'utilisateur sur l'alphabet des symboles de prédicats $\mathcal{P}red$. Nous dénoterons $\mathcal{A}$ l'automate fini déterministe associé à $\mathcal{R}$. Remarquons que, pour la correction des algorithmes présentés ici, le fait que $\mathcal{A}$ soit déterministe est une caractéristique essentielle, car les algorithmes ne peuvent fonctionner qu'avec ce type d'automates. Nous pouvons alors associer une contrainte $C_{ER(\mathcal{R})}$ aux motifs séquentiels logiques. Par exemple, si nous considérons l'expression rationnelle $\mathcal{R} = mv\ cd^*\ (emacs\ gcc|gcc)\ cd^*\ exec$ (dont l'automate fini déterministe associé $\mathcal{A}$ est décrit sur la Figure 4.3), nous pouvons dire que le motif séquentiel logique $\langle mv(F,D)\ cd(D)\ emacs(F)\ gcc(F,E)\ exec(E)\rangle$ vérifie $\mathcal{C}_{ER(\mathcal{R})}$.

Nous proposons maintenant la définition des contraintes de légalité et de validité des motifs adaptées au cadre des motifs séquentiels logiques.

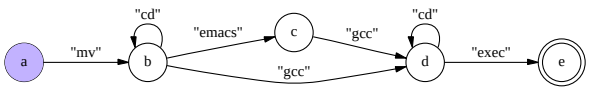

FIG. 4.3 – Automate Fini Déterministe associé à l'expression rationnelle *mv cd* (emacs gcc|gcc) cd* exec*

Définition 35 (Légalité d'un motif séquentiel logique) *Un motif séquentiel logique s est légal par rapport à un état b de l'automate $\mathcal{A}$ si chaque transition de $\mathcal{A}$ est définie à partir de b en suivant la suite de symboles de prédicats sur lesquels les atomes de s sont construits.*

Exemple : Si nous considérons l'automate de la Figure 4.3, nous pouvons dire que le motif séquentiel logique $\langle emacs(X)\ gcc(X,Y)\rangle$ est légal par rapport à l'état b de l'automate associé à l'expression rationnelle.

Définition 36 (Validité d'un motif séquentiel logique par rapport à un état) *Un motif séquentiel logique s est valide par rapport à un état b de $\mathcal{A}$ s'il est légal par rapport à b et si l'état final de la suite de transitions obtenue en suivant dans l'ordre les symboles de prédicat de s à partir de b est un état final de l'automate.*

Exemple : Avec l'automate de la Figure 4.3, nous pouvons dire que le motif séquentiel logique $\langle emacs(X)\ gcc(X,Y)\ cd(D)\ exec(Y)\rangle$ est valide par rapport à b. Enfin, le motif séquentiel logique $\langle mv(X,D)\ emacs(X)\ gcc(X,Y)\ exec(Y)\rangle$ est valide par rapport à l'état initial de $\mathcal{A}$, il est donc valide.

Définition 37 (Validité d'un motif séquentiel logique) *Nous dirons qu'un motif séquentiel logique s est valide si s est valide par rapport à l'état initial de l'automate $\mathcal{A}$.*

Les quatre versions différentes de SPIRIT-LºG que nous présentons ici couvrent l'ensemble des contraintes relaxées possibles C' étant donnée une contrainte $C_{ER(\mathcal{R})}$. Chacune des variantes prend en compte une contrainte C' de plus en plus restrictive dans la boucle d'extraction par niveaux. La première, SPIRIT-LºG(N), nécessite que tous les symboles de prédicat sur lesquels les atomes du motif séquentiel candidat sont construits apparaissent dans l'expression rationnelle $\mathcal{R}$. Le deuxième, SPIRIT-LºG(L), requiert que chaque motif candidat soit légal par rapport à un état de l'automate $\mathcal{A}$, associé à l'expression rationnelle $\mathcal{R}$. Le troisième, nommé SPIRIT-LºG(V), va plus loin en éliminant tout motif séquentiel candidat qui n'est valide par rapport à aucun état de l'automate $\mathcal{A}$. Enfin, SPIRIT-LºG(R) utilise directement la contrainte $C_{ER(\mathcal{R})}$ à l'intérieur de l'algorithme et requiert donc que chaque motif séquentiel logique candidat soit valide. Le Tableau 4.7 montre un récapitulatif des différentes relaxations de C' que nous considérerons pour l'algorithme SPIRIT-LºG. Nous disposons donc ainsi d'un ensemble d'algorithmes permettant de répondre à la requête inductive de notre scénario prototypique, moyennant un éventuel post-traitement dont la complexité dépendra du type de relaxation utilisé.

4.2.7 SPIRIT-LºG(N)

SPIRIT-LºG(N) considère la contrainte C' suivante : un motif séquentiel s vérifie C' si et seulement si tous les symboles de prédicat sur lesquels est construit s apparaissent dans l'expression rationnelle $\mathcal{R}$. Cette contrainte C' est anti-monotone.

Algorithme	**Contrainte Relaxée C'**
SPIRIT-LºG(N)	tous les symboles de prédicat doivent apparaître dans la contrainte $\mathcal{R}$
SPIRIT-LºG(L)	légale par rapport à un état de $\mathcal{A}_\mathcal{R}$
SPIRIT-LºG(V)	valide par rapport à un état de $\mathcal{A}_\mathcal{R}$
SPIRIT-LºG(R)	valide, i.e. $C' = C_{ER(\mathcal{R})}$

TAB. 4.7 – Les 4 variantes de SPIRIT-LºG

Preuve. Soit $s = \langle s_1 \ \ldots \ s_n \rangle$ un motif séquentiel logique vérifiant C'. Par l'absurde, supposons qu'il existe un sous-motif séquentiel logique s' de s ne vérifiant pas C'. Alors, cela signifie qu'il existe dans s' au moins un littéral construit sur un symbole de prédicat n'apparaissant pas dans $\mathcal{R}$. Par extension, puisque s' est un sous-motif séquentiel logique de s, cela est vrai aussi pour s. Donc s ne vérifie pas C', ce qui est absurde. On en déduit que C' est anti-monotone. □

Cette contrainte étant anti-monotone, il suffit donc d'adapter les procédures de génération et d'élagage des algorithmes APriori pour l'extraction de motifs séquentiels d'items au cadre des motifs séquentiels logiques.

Génération des candidats. Nous générons des candidats de taille k pour toute paire (s, t) de motifs séquentiels logiques telle que le suffixe de longueur $(k-2)$ de s s'unifie avec le préfixe de longueur $(k-2)$ de t. Plus formellement, soient $s = \langle s_1 \ \ldots s_{k-1} \rangle$ et $t = \langle t_1 \ \ldots t_{k-1} \rangle$ deux motifs séquentiels logiques, si nécessaire, nous renommons les variables de s et t de telle sorte que $V(s) \cap V(t) = \emptyset$. S'il existe θ telle que $t_j = \theta s_{j+1}$ pour $1 \leq j \leq k-2$, alors nous ajoutons à C_k tous les candidats de la forme $\langle \theta s \ \theta' t_{k-1} \rangle$ où θ' est une substitution d'une partie de $V(t_{k-1})$ avec des variables de $V(\theta s_1)$. Pour trouver toutes les substitutions θ' possibles, nous appelons la procédure ENUM-SUBST$(V(t_{k-1}), V(\theta s_1))$.

La procédure ENUM-SUBST prend en entrée deux ensembles V et W de variables et a pour but de renvoyer toutes les substitutions θ telles que $\forall \{X \mid Y\} \in \theta, X \in V$ et $Y \in W$ On remarque que pour chaque partie $P \in \mathcal{P}(V)$, il y a $|W|^{|P|}$ substitutions possibles. En effet, pour chaque variable de P, on peut choisir parmi $|W|$ variables pour la partie droite de la substitution. Le nombre total de substitutions renvoyées par l'algorithme est donc

$$\sum_{k=0}^{|V|} C_{|V|}^{k} |W|^k = (1 + |W|)^{|V|}$$

Dans notre cas, cela signifie que le nombre de candidats générés peut être exponentiel avec l'arité du dernier symbole de prédicat du motif séquentiel candidat. C'est pourquoi, lors de nos expériences, nous avons travaillé avec des symboles de prédicat de petite arité.

Exemple Supposons que nous ayons découvert les deux motifs séquentiels logiques $s = \langle p(X, R) \ \ q(Y) \ \ r(X, S) \rangle$ et $t = \langle q(L) \ r(N, T) \ s(J, T) \rangle$. Considérons la substitution $\theta = \{Y \mid L, X \mid N, S \mid T\}$, nous appelons la procédure ENUM-SUBST$(\{J, T\}, \{N, R\})$ qui nous retourne l'ensemble suivant de substitutions possibles :

$$\Omega = \{\epsilon, \{J \mid N\}, \{T \mid N\}, \{J \mid R\}, \{T \mid R\}, \{J \mid N, T \mid N\} \\ \{J \mid R, T \mid R\}, \{J \mid N, T \mid R\}, \{J \mid R, T \mid N\}\}$$

Algorithme 5: ENUM-SUBST

Données : V, W deux ensembles de variables
$\Omega = \{\epsilon$ la substitution identité$\}$;
pour chaque *partie* $P \in \mathcal{P}(V)$ **faire**
  $\Omega = \Omega \cup \{\theta, \theta$ est une substitution telle que $\forall\{X \mid Y\} \in \theta, X \in P$ et $Y \in W\}$;
fin
retourner Ω

Ainsi, nous ajoutons 9 candidats à l'ensemble des motifs séquentiels logiques candidats.

θ	Motif séquentiel candidat
ϵ	$\langle p(N,R)\ q(L)\ r(N,T)\ s(J,T)\rangle$
$\{J \mid N\}$	$\langle p(N,R)\ q(L)\ r(N,T)\ s(N,T)\rangle$
$\{T \mid N\}$	$\langle p(N,R)\ q(L)\ r(N,T)\ s(J,N)\rangle$
$\{J \mid R\}$	$\langle p(N,R)\ q(L)\ r(N,T)\ s(R,T)\rangle$
$\{T \mid R\}$	$\langle p(N,R)\ q(L)\ r(N,T)\ s(J,R)\rangle$
$\{J \mid N,T \mid N\}$	$\langle p(N,R)\ q(L)\ r(N,T)\ s(N,N)\rangle$
$\{J \mid R,T \mid R\}$	$\langle p(N,R)\ q(L)\ r(N,T)\ s(R,R)\rangle$
$\{J \mid N,T \mid R\}$	$\langle p(N,R)\ q(L)\ r(N,T)\ s(N,R)\rangle$
$\{J \mid R,T \mid N\}$	$\langle p(N,R)\ q(L)\ r(N,T)\ s(R,N)\rangle$

Preuve. Montrons que cette méthode de génération des candidats est correcte et complète.

La correction découle immédiatement de l'anti-monotonie de la contrainte C' que nous considérons ici. En effet, si nous considérons deux motifs séquentiels logiques de taille $k-1$ vérifiant C' et que nous les fusionnons par la méthode décrite ci-dessus, nous obtenons nécessairement un candidat vérifiant C'. □

Intéressons nous maintenant à la complétude de cette méthode de génération des candidats. Faisons une démonstration par récurrence. Soit $P(k)$ la propriété suivante : "Les motifs séquentiels logiques de taille k de C_k vérifient C'"

Montrons d'abord que $P(1)$ est vraie. Pour construire C_1, on fait une énumération exhaustive de tous les atomes libres pouvant être construits sur les symboles de prédicat apparaissant dans $\mathcal{D}$ et apparaissant dans $\mathcal{R}$. Donc $P(1)$ est vraie par construction. On remarque qu'on élimine ensuite les non fréquents toujours par construction de C_1.

Passons à la phase de récurrence. Supposons que $P(k)$ soit vraie pout tout $k \in \{1, \ldots, n\}$, montrons que $P(n+1)$ est vraie.

Pour cela, raisonnons par l'absurde, supposons qu'il existe un motif séquentiel logique s de longueur $n+1$ vérifiant C' qui n'appartient pas à C_{n+1}. Notons le $s = \langle s_1, s_2, \ldots, s_{n+1}\rangle$. Considérons alors les deux sous-motifs suivants : $A = \langle s_1\ s_2\ \ldots\ s_n\rangle$ et $B = \langle s_2\ s_3\ \ldots\ s_{n+1}\rangle$. Nous avons donc que A et B sont deux motifs séquentiels logiques vérifiant C' (par anti-monotonie de C'). Par récurrence, nous avons donc que A et B appartiennent à C_n (au renommage de variables près). Renommons les variables de A et B de telle sorte que $V(A) \cap V(B) = \emptyset$. Pour cela, nous pouvons appliquer un substitution θ_A à A. Comme il s'agit d'un simple renommage, θ_A est telle qu'il n'existe pas deux variables X et Y telles que $\theta_A X = \theta_A Y$. Nous avons donc $V(\theta_A A) \cap V(B) = \emptyset$. Notons θ'_A le renommage réciproque de θ_A.

Considérons désormais notre procédure de génération avec nos deux motifs $\theta_A A$ et B. Ces deux motifs peuvent être utilisés pour générer un nouveau candidat car il existe une substitution

permettant d'unifier les $n-1$ derniers éléments de $\theta_A A$ avec B. En particulier, considérons la substitution θ'_A. On obtient alors un ensemble de nouveaux candidats de taille $n+1$ dont les éléments sont de la forme $\langle s_1\ s_2\ \ldots\ s_n\ q\rangle$ où q est un atome libre basé sur le même symbole de prédicat que s_{n+1} et obtenu en considérant le renommage des variable de s_{n+1}. Plus précisément, d'après notre méthode de génération, nous devons considérer le renommage de $V(s_{n+1})$ avec $V(\theta'_A(\theta_A s_1))$, i.e. de $V(s_{n+1})$ avec $V(s_1)$. Parmi toutes ces substitutions possibles, on peut par exemple prendre la substitution identité ϵ. On génère alors le motif $\langle s_1, s_2, \ldots, s_{n+1}\rangle$, c'est-à-dire s. D'où l'absurdité, donc $P(n+1)$ est vraie. □

Élagage des candidats. Un motif séquentiel logique candidat s est élagué de C_k s'il existe un sous-motif séquentiel logique de longueur $k-1$ qui ne s'unifie avec aucun motif séquentiel logique de F_{k-1}.

Condition d'arrêt. L'ensemble des k-motifs séquentiels logiques fréquents F_k est vide. Nous ne pouvons alors plus générer de candidats à l'étape $(k+1)$.

4.2.8 SPIRIT-L^{o}G**(L)**

Dans SPIRIT-L^{o}G(L), C' spécifie que chaque motif séquentiel candidat doit être légal par rapport à un état de l'automate $\mathcal{A}$. L'algorithme utilise donc $\mathcal{A}$ pour réaliser l'élagage des k-motifs séquentiels logiques candidats qui ne sont légaux par rapport à aucun état de celui-ci. $F_k(b)$ est l'ensemble des k-motifs séquentiels fréquents et légaux par rapport à un état b de $\mathcal{A}$.

Cette contrainte C' n'est ni monotone, ni anti-monotone.

Preuve. Pour l'anti-monotonie, prenons par exemple $\mathcal{R} = p^*qu$, et soit $\mathcal{A}$ l'automate fini déterministe associé à $\mathcal{R}$. Prenons un motif séquentiel $s = \langle p(X,Y)\ q(Y)\ u(X,T)\rangle$. Alors s vérifie C'. Considérons son sous-motif séquentiel logique $s' = \langle p(X,Y)\ u(X,T)\rangle$, alors s' ne vérifie pas C'. Donc C' n'est pas anti-monotone. De même considérons un motif $s'' = \langle p(X,Y)\ q(Y)\ u(X,T)\ p(X,T)\rangle$, alors s est un sous-motif de s'' et s'' ne vérifie pas C'. On en déduit que C' n'est donc pas monotone non plus. □

Génération des candidats. Pour chaque état b de l'automate $\mathcal{A}$, nous ajoutons à C_k les k-motifs séquentiels logiques candidats qui sont légaux par rapport à b et potentiellement fréquents.

Avant de présenter la procédure de génération, nous avons besoin du lemme suivant :

Lemme 2 *Soit $s = \langle s_1 \ldots s_k\rangle$ un motif séquentiel logique légal par rapport à un état b de $\mathcal{A}$, où $b \xrightarrow{p} c$ est une transition de $\mathcal{A}$ et où p est le symbole de prédicat sur lequel s_1 est construit. Pour que s soit fréquent, il faut que $\langle s_1 \ldots s_{k-1}\rangle$ soit fréquent et légal par rapport à b, et $\langle s_2 \ldots s_k\rangle$ soit fréquent et légal par rapport à c.*

Preuve. Soit $s = \langle s_1 \ldots s_k\rangle$ un motif séquentiel logique fréquent légal par rapport à un état b de $\mathcal{A}$. On suppose que $b \xrightarrow{p} c$ est une transition de $\mathcal{A}$ et où p est le symbole de prédicat sur lequel s_1 est construit.

Comme s vérifie C', cela signifie que la chaîne de caractères obtenue en prenant dans l'ordre les symboles de prédicat de s décrit un chemin de l'automate $\mathcal{A}$ depuis l'état b. Cela reste vrai pour son préfixe de longueur $n-1$. Prenons $s' = \langle s_1 \ldots s_{k-1}\rangle$, alors s' vérifie C'. Considérons ensuite Σ l'ensemble des substitutions qui font réussir s sur $\mathcal{D}$, alors l'application de Σ à s' nous

garantit que s' est fréquent (anti-monotonie de la contrainte de fréquence minimale). Donc s' est fréquent et légal par rapport à b.

De même, si s_1 est bâti sur le symbole de prédicat p est que nous avons une transition $b \xrightarrow{p} c$, alors le suffixe de longueur $n-1$ de s permet de donner une chaîne de caractères décrivant un chemin dans l'automate $\mathcal{A}$ à partir de l'état c. Prenons alors $s'' = \langle s_2 \dots s_n \rangle$, nous avons alors que s'' est légal par rapport à c. Par anti-monotonie de la contrainte de fréquence minimale, nous avons que s'' est fréquent. □

Les motifs séquentiels logiques candidats pour un état b seront construits comme suit. Pour tout motif s dans $F_{k-1}(b)$, si $b \xrightarrow{p} c$ est une transition de $\mathcal{A}$ et s_1 est construit sur le symbole de prédicat p, alors, pour tout motif séquentiel t dans $F_{k-1}(c)$ tel que :

- $V(s) \cap V(t) = \emptyset$ (on peut renommer les variables de s et de t de telle sorte que ce soit le cas).
- $\langle s_2 \dots s_{k-1} \rangle$ s'unifie avec $\langle t_1 \dots t_{k-2} \rangle$ (nous notons θ la substitution telle que $t_j = \theta s_{j+1}$, pour $1 \leq j \leq k-2$).

nous ajoutons à C_k tous les motifs séquentiels fréquents de la forme $\langle \theta s \ \theta' t_{k-1} \rangle$, où θ' est une substitution d'une partie de $V(t_{k-1})$ avec des variables de $V(\theta s_1)$. Pour trouver toutes les substitutions possibles, nous appelons la procédure ENUM-SUBST($V(t_{k-1})$, $V(\theta s_1)$).

On remarque en fait que cette procédure de génération est la même que celle mise en place dans le cas de SPIRIT-LºG(N). La preuve de complétude est donc la même. La correction de la méthode de génération découle immédiatement quand à elle du lemme précédent. □

Élagage des candidats. Pour élaguer un motif candidat s, nous devons trouver un sous-motif séquentiel logique de s qui soit légal par rapport à un état de $\mathcal{A}$ et qui ne s'unifie avec aucun motif séquentiel logique fréquent déjà découvert. Pour cela, nous calculons les sous-motifs maximaux de s de longueur inférieure à k et qui sont légaux par rapport à un état de $\mathcal{A}$. Si aucune d'entre elle ne s'unifie avec un motif séquentiel logique de F, alors s est élagué de C_k.

Nous devons trouver tous les sous-motifs logiques maximaux légaux d'un motif candidat s. Pour cela, nous pouvons utiliser l'algorithme FINDMAXSUBSEQ présenté dans [GRS02]. L'algorithme 6 en rappelle le principe. Étant donné un automate $\mathcal{A}$ dont les ensembles d'états finaux et initiaux (S et E) sont donnés en paramètres, il retourne tous les sous-motifs séquentiels logiques valides de s en calculant d'abord tous les sous-motifs séquentiels logiques de s qui sont valides par rapport à un état de $\mathcal{A}$ et en retournant ensuite uniquement ceux qui sont valides par rapport à un état apparaissant dans l'ensemble S. La seule différence par rapport à l'algorithme initialement présenté dans [GRS02] est que nous raisonnons ici sur des symboles de prédicats plutôt que sur des items, ce qui n'a aucun impact sur le fonctionnement de l'algorithme. De plus, l'aspect "variables" des motifs séquentiels logiques n'influe pas sur la satisfaction de C'. Nous rappelons les principaux faits sur lesquels cet algorithme est basé. Soit maxSeq(b, s) l'ensemble des sous-motifs maximaux de $s = \langle s_1 \dots s_{|s|} \rangle$ qui sont valides par rapport à un état b de $\mathcal{A}$ et $t = \langle s_2 \dots s_{|s|} \rangle$, un sur-ensemble de maxSeq(b, s) peut être calculé à partir de maxSeq(b, t) en utilisant le fait que :

- $maxSeq(b, s) \subseteq maxSeq(b, t) \cup \{\langle s_1 \ u \rangle \text{ avec } u \in maxSeq(c, t)\} \cup \{s_1\}$ où s_1 est construit sur $p \in \mathcal{P}red$, et où $b \xrightarrow{p} c$ est une transition de $\mathcal{A}$.
- $maxSeq(b, s) \subseteq maxSeq(b, t)$ sinon.

Un post-traitement est alors nécessaire pour conserver uniquement les sous-motifs séquentiels maximaux. Ainsi, pour trouver tous les sous-motifs séquentiels logiques maximaux d'un motif

Algorithme 6: FINDMAXSUBSEQ pour trouver les sous-motifs séquentiels maximaux

Data : $\mathcal{A}$ un automate, S un ensemble d'états initiaux, E un ensemble d'états finaux, s un motif séquentiel logique

pour chaque état b de $\mathcal{A}$ **faire** maxSeq[b] $\leftarrow \emptyset$;
pour k de $|s|$ à 1 **faire**
  pour chaque état b de $\mathcal{A}$ **faire**
    tmpSeq[b] $\leftarrow \emptyset$;
    si $\exists$ une transition $b \xrightarrow{p} c$ dans $\mathcal{A}$ et que s_k est construit sur le symbole p **alors**
      si $c \in E$ **alors** tmpSeq[b] $\leftarrow \{s_k\}$
      tmpSeq[b] $\leftarrow$ tmpSeq[b] $\cup \{\langle s_k\ t\rangle : t \in$ maxSeq[c]$\}$;
  fin
  pour chaque état b de l'automate $\mathcal{A}$ **faire**
    maxSeq[b] $\leftarrow$ maxSeq[b] $\cup$ tmpSeq[b];
    pour chaque motif séquentiel logique t dans MaxSeq[b] **faire**
      si $\exists\ u \in$ maxSeq[b]-$\{\langle s_k \dots s_{|s|}\rangle\}$ tel que $t \subseteq u$ **alors** Elaguer t de maxSeq[b];
    fin
  fin
fin
retourner $\bigcup_{b\in S}$ maxSeq[b]-$\{s\}$ (après l'élagage des motifs non maximaux)

candidat s, nous appelons la fonction FINDMAXSUBSEQ avec des ensembles S et E tous les deux égaux à l'ensemble des états de $\mathcal{A}$.

Condition d'arrêt. L'ensemble des motifs séquentiels logiques fréquents de longueur k qui sont légaux par rapport à l'état initial a de l'automate $\mathcal{A}$ est vide, i.e. $F_k(a)$ est vide. Nous ne pourrons ainsi plus générer de nouveaux motifs logiques candidats de longueur $(k+1)$ qui soient légaux par rapport à a.

4.2.9 SPIRIT-L$^{\text{O}}$G(V)

SPIRIT-L$^{\text{O}}$G(V) utilise une contrainte C' plus forte que dans SPIRIT-L$^{\text{O}}$G(L) étant donné que nous imposons que tous les candidats générés soient valides par rapport à un état b de l'automate $\mathcal{A}$. $F_k(b)$ dénote l'ensemble des motifs séquentiels logiques de longueur k fréquents et qui sont valides par rapport à b.

Génération des candidats. Pour présenter notre méthode de génération des candidats, nous aurons besoin du lemme suivant. Nous supposons toujours disposer d'une expression rationnelle $\mathcal{R}$ dont l'automate fini déterministe associé est $\mathcal{A}$, ainsi que d'une base de séquences en entrée $\mathcal{D}$.

Lemme 3 *Soient un motif séquentiel logique fréquent $s \in C_k$ est valide par rapport à un état b de $\mathcal{A}$, et $b \xrightarrow{p} c$ est une transition de $\mathcal{A}$ et s_1 est un atome libre construit sur $p \in \mathcal{P}red$, alors le sous-motif séquentiel logique $\langle s_2 \dots s_k\rangle$ qui est le suffixe de longueur $(k-1)$ de s est fréquent et valide par rapport à c, i.e. il appartient à C_{k-1} à un renommage près.*

Preuve. Soit s un motif séquentiel logique fréquent valide par rapport à un état b de l'automate $\mathcal{A}$, notons $s = \langle s_1 \dots s_n \rangle$. Soient p le symbole de prédicat sur lequel est construit s_1, alors il existe un état c de l'automate tel que $b \xrightarrow{p} c$ une transition de $\mathcal{A}$. Considérons le motif séquentiel logique $s' = \langle s_2 \dots s_n \rangle$ constitué en fait du suffixe de longueur $n-1$ de s. Par anti-monotonie de la contrainte de fréquence minimale, nous avons que s' est fréquent dans $\mathcal{D}$. De plus, comme le chemin obtenu en partant de b et en suivant la séquence des symboles de prédicat de s conduit à un état final de $\mathcal{A}$ et que $b \xrightarrow{p} c$ est une transition de $\mathcal{A}$, le chemin obtenu en partant de c et en suivant la séquence des symboles de prédicat de s' conduit au même état final. Donc s' est fréquent et valide par rapport à c, i.e. $s' \in F_{n-1}(c)$ (à un renommage de variables près). □

Ce lemme nous permet de déduire un processus intéressant pour la génération des motifs candidats. Pour toutes les transitions $b \xrightarrow{p} c$ où p est un symbole de prédicat d'arité n, pour tout motif séquentiel logique $t \in F_{k-1}(c)$, tel que $V(t) \cap \{X_1, \dots, X_n\} = \emptyset$ (quitte pour cela de renommer les variables de t), nous ajoutons à l'ensemble des candidats pour un état b les motifs séquentiels logiques de longueur k de la forme $\langle \theta u\ t \rangle$ où u est un littéral construit sur p et $\{X_1, \dots, X_n\}$ et θ est une substitution d'une partie de $V(u)$ avec certaines variables de $V(t)$. Pour trouver l'ensemble de ces substitutions, nous appelons la procédure Enum-Subst$(V(u), V(t))$. L'union de tous ces ensembles de candidats pour tous les états de $\mathcal{A}$ donne C_k.

Exemple Supposons que nous ayons découvert le motif séquentiel logique $s = \langle q(X,T)\ r(T,K) \rangle$ dans F_2, et qu'il soit valide par rapport à un état c. S'il existe une transition $b \xrightarrow{p} c$ et que l'arité de p est 2, nous allons considérer les littéraux suivants construits sur p : $p(X_1, X_1)$ et $p(X_1, X_2)$.

– Pour $p(X_1, X_2)$, appeler Enum-Subst$(\{X_1, X_2\}, \{X, T, K\})$ nous renvoie :

$$\begin{aligned}\Omega = \{&\epsilon, \{X_1/X\}, \{X_1/T\}, \{X_1/K\}, \{X_2/X\}, \{X_2/T\}, \{X_2/K\}, \{X_1/X, X_2/X\},\\ &\{X_1/X, X_2/T\}, \{X_1/X, X_2/K\}, \{X_1/T, X_2/X\}, \{X_1/T, X_2/T\},\\ &\{X_1/T, X_2/K\}, \{X_1/K, X_2/X\}, \{X_1/K, X_2/T\}, \{X_1/K, X_2/K\}\}\end{aligned}$$

Nous ajoutons alors à C_3 les candidats suivants :
– Avec $\theta = \epsilon$, nous ajoutons $\langle p(X_1, X_2)\ q(X,T)\ r(T,K) \rangle$
– Avec $\theta = \{X_1/X\}$, nous ajoutons $\langle p(X, X_2)\ q(X,T)\ r(T,K) \rangle$
– Avec $\theta = \{X_1/T\}$, nous ajoutons $\langle p(T, X_2)\ q(X,T)\ r(T,K) \rangle$
– Avec $\theta = \{X_1/K\}$, nous ajoutons $\langle p(K, X_2)\ q(X,T)\ r(T,K) \rangle$
– Avec $\theta = \{X_2/X\}$, nous ajoutons $\langle p(X_1, X)\ q(X,T)\ r(T,K) \rangle$
– Avec $\theta = \{X_2/T\}$, nous ajoutons $\langle p(X_1, T)\ q(X,T)\ r(T,K) \rangle$
– Avec $\theta = \{X_2/K\}$, nous ajoutons $\langle p(X_1, K)\ q(X,T)\ r(T,K) \rangle$
– Avec $\theta = \{X_1/X, X_2/X\}$, nous ajoutons $\langle p(X, X)\ q(X,T)\ r(T,K) \rangle$
– Avec $\theta = \{X_1/X, X_2/T\}$, nous ajoutons $\langle p(X, T)\ q(X,T)\ r(T,K) \rangle$
– Avec $\theta = \{X_1/X, X_2/K\}$, nous ajoutons $\langle p(X, K)\ q(X,T)\ r(T,K) \rangle$
– Avec $\theta = \{X_1/T, X_2/X\}$, nous ajoutons $\langle p(T, X)\ q(X,T)\ r(T,K) \rangle$
– Avec $\theta = \{X_1/T, X_2/T\}$, nous ajoutons $\langle p(T, T)\ q(X,T)\ r(T,K) \rangle$
– Avec $\theta = \{X_1/T, X_2/K\}$, nous ajoutons $\langle p(T, K)\ q(X,T)\ r(T,K) \rangle$
– Avec $\theta = \{X_1/K, X_2/X\}$, nous ajoutons $\langle p(K, X)\ q(X,T)\ r(T,K) \rangle$
– Avec $\theta = \{X_1/K, X_2/T\}$, nous ajoutons $\langle p(K, T)\ q(X,T)\ r(T,K) \rangle$
– Avec $\theta = \{X_1/K, X_2/K\}$, nous ajoutons $\langle p(K, K)\ q(X,T)\ r(T,K) \rangle$

– Pour $p(X_1, X_1)$, appeler ENUM-SUBST($\{X_1\}$, $\{X, T, K\}$) nous renvoie :

$$\Omega = \{\epsilon, \{X_1/X\}, \{X_1/T\}, \{X_1/K\}\}$$

Nous ajoutons alors à C_3 les candidats suivants :
– Avec $\theta = \epsilon$, nous ajoutons $\langle p(X_1, X_1)\ q(X, T)\ r(T, K)\rangle$
– Avec $\theta = \{X_1/X\}$, nous ajoutons $\langle p(X, X)\ q(X, T)\ r(T, K)\rangle$
– Avec $\theta = \{X_1/T\}$, nous ajoutons $\langle p(T, T)\ q(X, T)\ r(T, K)\rangle$
– Avec $\theta = \{X_1/K\}$, nous ajoutons $\langle p(K, K)\ q(X, T)\ r(T, K)\rangle$

Le lemme précédent nous assure que cette procédure de génération est correcte, i.e. que nous générons des candidats valides par rapport à l'état b de l'automate. Montrons maintenant qu'elle est complète. Raisonnons par récurrence et définissons pour tout entier n la propriété $P(n)$ suivante : "C_n contient tous les motifs séquentiels logiques valides par rapport à un état de l'automate $\mathcal{A}$ et fréquents"

Montrons d'abord que $P(1)$ est vraie. Notons $\mathcal{F}$ l'ensemble des états finaux de $\mathcal{A}$ et $\mathcal{E}$ l'ensemble des états de $\mathcal{A}$. Pour construire C_1, on fait une énumération exhaustive de tous les atomes libres pouvant être construits sur un symbole de prédicat p tel qu'il existe un état $c \in \mathcal{F}$, un état $b \in \mathcal{E}$, et une transition dans $\mathcal{A}$ de la forme $b \xrightarrow{p} c$. Par construction, $P(1)$ sera donc vraie.

Passons maintenant à la phase de récurrence. Supposons $P(n)$ vraie pour un certain entier $n \geq 1$ et montrons que $P(n+1)$ est vraie. Raisonnons par l'absurde : soit s un motif séquentiel logique de longueur $n+1$ valide par rapport à un état b de $\mathcal{A}$ et fréquent. Supposons que $s \notin C_{n+1}$. Notons $s = \langle s_1 \dots s_{n+1}\rangle$ et notons p le symbole de prédicat sur lequel est construit s_1, alors il existe un état de c de $\mathcal{A}$ tel que $b \xrightarrow{p} c$. Posons alors $s' = \langle s_2 \dots s_{n+1}\rangle$ le suffixe de longueur n de s. D'après le lemme, nous avons que s' est valide par rapport à c et fréquent. Par récurrence, cela signifie que $s' \in C_n$ à un renommage près. Plus précisément, il existe $t \in C_n$ et un renommage Ω tel que $t = \Omega s'$. Notons Ω' son renommage réciproque et a le motif séquentiel logique de longueur 1 correspondant à $\langle \Omega s_1\rangle$. Soit V l'ensemble des variables communes à a et t et W l'ensemble des variables de a n'apparaissant pas dans t. Formellement, $V = V(a) \cap V(t)$ et $W = V(a) \backslash V(t)$.

Nous avons donc t un motif de C_n fréquent et valide par rapport à c et une transition $b \xrightarrow{p} c$, nous pouvons donc considérer notre procédure de génération de candidats. Nous allons considérer tous les atomes libres pouvant être construits sur le symbole de prédicat p. Parmi ceux-ci, il y a un atome libre b équivalent à a par un simple renommage Θ et tel que $V(b) \cap V(t) = \emptyset$. Nous avons donc $b = \Theta a$ et $a = \Theta' b$ (où Θ' est le renommage réciproque de Θ). Considérons Θ_V le renommage Θ restreint aux seules variables apparaissant dans V (et Θ'_V son renommage réciproque) et Θ_W le renommage Θ restreint aux seules variables de W (et Θ'_W son renommage réciproque). Nous avons alors $\Theta_W \cap \Theta_V = \emptyset$ et $\Theta = \Theta_W \cup \Theta_V$.

L'appel de la fonction ENUM-SUBST nous renvoie tous les renommages possibles d'une partie d'une partie de $V(b)$ avec des variables de $V(t)$. Parmi celles-ci, il y a donc Θ'_V. Nous allons donc générer un candidat de la forme $\langle \Theta'_V b\ t\rangle$. Nous allons maintenant montrer que ce candidat est équivalent à s à un renommage près. Appliquons tout d'abord le renommage Θ'_W à $\langle \Theta'_V b\ t\rangle$. Comme Θ_W ne porte que sur des variables n'apparaissant pas dans t, Θ'_W ne porte aussi que sur des variables de $\Theta'_V b$ n'apparaissant pas dans t. On peut alors appliquer cette substitution réciproque à notre candidat. On obtient alors $\Theta'_W\langle \Theta'_V b\ t\rangle = \langle \Theta'_W \Theta'_V b\ t\rangle = \langle \Theta' b t\rangle = \langle a t\rangle$. Nous

Algorithme 7: Algorithme de génération de candidats de longueur k pour SPIRIT-L^{0}G(R)

Données : s un motif séquentiel logique,
b l'état courant,
B l'ensemble des états traversés
Résultat : C_k l'ensemble des motifs séquentiels logiques valides de taille k

pour chaque *transition* $b \xrightarrow{p} c$ *de* $\mathcal{A}$ **faire**
  pour chaque *atome libre* l *pouvant être construit sur le symbole de prédicat* p *et tel que* $V(l) \cap V(s) = \emptyset$ **faire**
    $\Omega = Enum\text{-}Subst(V(l), V(s))$
    si *(*$|s| = k-1$ **et** c est un état final*)* **alors**
      pour chaque $\theta' \in \Omega$ **faire** $C_k = C_k \cup \{\langle s\ \theta' l\rangle\}$;
    si *(*$|s| \neq k-1$ **and** *(*c n'est pas un état final **ou** *(*$\exists t \in F, \exists$ une substitution Θ telle que $t = \Theta\langle s\ l\rangle$*))* **alors**
      pour chaque $\theta' \in \Omega$ **faire** $\mathrm{GenR}(\langle s\ \theta' l\rangle, c, B \cup \{b\})$
  fin
fin

pouvons alors appliquer la substitution Ω' et nous obtenons $\Omega'\langle at\rangle = \langle \Omega' a \Omega' t\rangle = \langle s_1 s'\rangle = s$. Donc $s \in C_{n+1}$ (au renommage $\Omega'\Theta'_W$ près), d'où l'absurdité. □

Élagage des candidats. Cette étape est similaire à celle de SPIRIT-L^{0}G(L). L'idée est d'élaguer de C_k tous candidat s possédant un sous-motif séquentiel valide par rapport à un état de $\mathcal{A}$ et qui ne peut s'unifier avec aucun motif séquentiel logique fréquent déjà dans F. Ainsi, étant donné un motif séquentiel logique, nous avons besoin de calculer ses sous-motifs qui sont valides par rapport à un état de l'automate $\mathcal{A}$. Pour cela, nous appelons la procédure FindMaxSubseq avec S égal à l'ensemble de tous les états de $\mathcal{A}$ et E contenant l'ensemble des états finaux de $\mathcal{A}$.

Condition d'arrêt. L'ensemble F_k des motifs séquentiels logiques fréquents de longueur k valides par rapport à un état de $\mathcal{A}$ est vide. Nous serons donc incapables de générer des candidats à l'étape suivante.

4.2.10 SPIRIT-L^{0}G**(R)**

Dans cette variante, nous ne relaxons pas la contrainte spécifiée à l'aide de l'expression rationnelle $\mathcal{R}$. Nous demandons donc à chaque motif séquentiel logique candidat d'être valide, on a donc l'égalité i.e. $C' = C_{ER(\mathcal{R})}$ dans l'Algorithme 4.

Génération des candidats. Pour générer les candidats de C_k, nous devons énumérer tous les motifs séquentiels logiques candidats de longueur k. L'algorithme 7 est une adaptation de la procédure proposée dans [GRS02] pour générer des candidats. Nous parcourons toutes les transitions de $\mathcal{A}$ et nous énumérons tous les motifs séquentiels logiques qui peuvent être construits sur les séries de symboles de prédicats constituant un chemin d'un état initial à un état final

de $\mathcal{A}$. Pour optimiser la génération des candidats, nous utilisons une optimisation : si un motif séquentiel logique s est valide mais pas fréquent, alors il n'est pas nécessaire de l'étendre car il ne peut générer un motif séquentiel logique fréquent.

Exemple. Considérons l'automate de la Figure 4.4. Supposons que nous souhaitions générer des candidats valides de longueur $k = 3$ et que nous ayons déjà fait deux itérations. Plus particulièrement, supposons que nous ayons déjà obtenu dans F_2 le motif séquentiel logique $\langle p(X,Y)\ q(Z,Y)\rangle$. Nous allons alors pratiquer un appel récursif avec les paramètres suivants : $s = \langle p(X,Y)\ q(Z,Y)\rangle$, $b = 2$ et $B = \{0,1\}$. Nous devons alors considérer toutes les transitions partant de l'état numéro 2. Parmi celles-ci, il y a :

- $2 \xrightarrow{r} 3$. On suppose r d'arité 1. Comme $|s| = k-1$ et que 3 n'est pas final, les deux conditions ne sont pas vérifiées. Aucun motif n'est ajouté à C_k et aucun nouvel appel récursif n'est généré.
- $2 \xrightarrow{t} 4$. On suppose t d'arité 2, on va alors considérer tous les atomes pouvant être construits sur t. Il y en a deux : $t(X_1, X_2)$ et $t(X_1, X_1)$.
  - pour $t(X_1, X_1)$, nous appelons *Enum-Subst* avec les paramètres $\{X_1\}$ et $\{X, Y, Z\}$ et nous obtenons $\Omega = \{\epsilon, \{X_1 \mid X\}, \{X_1 \mid Y\}, \{X_1 \mid Z\}\}$. Comme 4 est final, nous ajoutons à C_k les élements suivants :
    - avec $\theta' = \{\epsilon\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)t(X_1,X_1)\rangle$
    - avec $\theta' = \{X_1 \mid X\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)t(X,X)\rangle$
    - avec $\theta' = \{X_1 \mid Y\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)t(Y,Y)\rangle$
    - avec $\theta' = \{X_1 \mid Z\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)t(Z,Z)\rangle$
  - pour $t(X_1, X_2)$, nous appelons *Enum-Subst* avec les paramètres $\{X_1, X_2\}$ et $\{X, Y, Z\}$ et cela nous renvoie :

$$\Omega = \{\epsilon, \{X_1/X\}, \{X_1/Y\}, \{X_1/Z\}, \{X_2/X\}, \{X_2/Y\}, \{X_2/Z\}, \{X_1/X, X_2/X\}, \{X_1/X, X_2/Y\}, \{X_1/X, X_2/Z\}, \{X_1/Y, X_2/X\}, \{X_1/Y, X_2/Y\}, \{X_1/Y, X_2/Z\}, \{X_1/Z, X_2/X\}, \{X_1/Z, X_2/Y\}, \{X_1/Z, X_2/Z\}\}$$

    Comme 4 est final, nous ajoutons à C_k les élements suivants :
    - avec $\theta' = \{\epsilon\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)\ t(X_1,X_2)\rangle$
    - avec $\theta' = \{X_1/X\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)\ t(X,X_2)\rangle$
    - avec $\theta' = \{X_1/Y\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)\ t(Y,X_2)\rangle$
    - avec $\theta' = \{X_1/Z\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)\ t(Z,X_2)\rangle$
    - avec $\theta' = \{X_2/X\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)\ t(X_1,X)\rangle$
    - avec $\theta' = \{X_2/Y\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)\ t(X_1,Y)\rangle$
    - avec $\theta' = \{X_2/Z\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)\ t(X_1,Z)\rangle$
    - avec $\theta' = \{X_1/X, X_2/X\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)\ t(X,X)\rangle$
    - avec $\theta' = \{X_1/X, X_2/Y\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)\ t(X,Y)\rangle$
    - avec $\theta' = \{X_1/X, X_2/Z\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)\ t(X,Z)\rangle$
    - avec $\theta' = \{X_1/Y, X_2/X\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)\ t(Y,X)\rangle$
    - avec $\theta' = \{X_1/Y, X_2/Y\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)\ t(Y,Y)\rangle$
    - avec $\theta' = \{X_1/Y, X_2/Z\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)\ t(Y,Z)\rangle$
    - avec $\theta' = \{X_1/Z, X_2/X\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)\ t(Z,X)\rangle$
    - avec $\theta' = \{X_1/Z, X_2/Y\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)\ t(Z,Y)\rangle$
    - avec $\theta' = \{X_1/Z, X_2/Z\}$, on ajoute $\langle p(X,Y)\ q(Z,Y)\ t(Z,Z)\rangle$

Montrons la correction et la complétude de la procédure de génération

Commençons par la complétude. Par l'absurde, supposons qu'un motif séquentiel logique valide s de longueur k ne soit pas généré. Notons $s = \langle s_1 \dots s_k\rangle$. Comme s est valide, nous

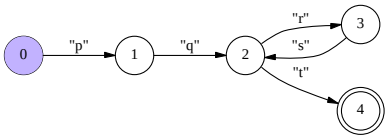

FIG. 4.4 – Automate correspondant à l'expression rationnelle $pq(rs)^*t$

savons que la séquence ordonnée des symboles de prédicats de s constitue un mot décrit par l'expression rationnelle $\mathcal{R}$ donnée par l'utilisateur. Notons alors $E_0, \dots E_k$ la suite d'états de $\mathcal{A}$ ainsi traversés. Si l'on note p_i, $1 \leq i \leq k$ le symbole de prédicat sur lequel est construit l'atome s_i, nous avons qu'il existe alors un ensemble de transitions dans $\mathcal{A}$ de la forme $E_{i-1} \xrightarrow{p_i} E_i$. De plus, comme s est valide, nous avons que E_0 est un état initial de $\mathcal{A}$ et E_k est un état final de $\mathcal{A}$. Notons S_i la fonction qui renvoie le préfixe de longueur i de s. Notons r_n, b_n, B_n les paramètres de l'algorithme à l'itération n. Pour tout $n \in \{1, \dots, k-1\}$, nous allons maintenant montrer la propriété $P(n)$ suivante : "A chaque itération n, soit l'algorithme ajoute à C_k un motif équivalent à S_k à un renommage près, soit il fait un appel récursif avec r_{n+1} équivalent à S_n à un renommage près, légal par rapport à E_0 et $b_{n+1} = E_n$".

Montrons que $P(1)$ est vraie. Lors du premier appel, nous avons $r_1 = \emptyset$, $B_1 = \emptyset$, $b_1 = E_0$. Nous savons qu'il existe une transition $E_0 \xrightarrow{p_1} E_1$. Nous allons considérer tous les atomes libres pouvant être construits sur p_1, Parmi ceux-ci, il existe l qui est un renommage de S_1 par Θ, i.e. $l = \Theta s_1$ Nous avons deux cas :

- Si $k = 1$ et que E_1 est final, nous ajoutons $\langle l \rangle$ à C_k. Comme E_0 est initial et E_1 final, nous avons que $\langle l \rangle$ est valide.
- Si $k \neq 1$ et que E_1 n'est pas final, nous faisons un appel récursif avec $r_2 = \langle l \rangle$, $b_2 = E_1$, $B_2 = \{E_0\}$. Comme E_0 est un état initial de l'automate $\mathcal{A}$, nous avons que r_2 est légal par rapport à E_0.

Donc $P(1)$ est vraie.

Passons à la phase de récurrence, supposons $P(n)$ vraie pour $n \geq 1$, et montrons que $P(n+1)$ est vraie. Comme $P(n)$ est vraie, cela signifie que r_n est équivalent à S_{n-1} à un renommage Ω près, légal par rapport à E_0 et $b_n = E_{n-1}$. Nous savons qu'il existe une transition étiquettée $E_{n-1} \xrightarrow{p_n} E_n$. Nous allons considérer tous les atomes libres pouvant être construits sur p_n. Parmi ceux-ci, il y a l qui est un renommage de s_n et $V(l) \cap V(r_n) = \emptyset$. Nous avons alors deux cas :

- Si $n - 1 = k - 1$, i.e. si $n = k$ et E_n final, nous ajoutons des candidats à C_k. Nous avons que $r_n = \Omega S_{n-1}$, posons alors $a = \langle \Omega s_n \rangle$. Considérons maintenant les deux ensembles $V = V(a) \cap V(r_n)$ et $W = V(a) \cup V(r_n)$. l est un renommage de s_n donc c'est aussi un renommage de a par une substitution Θ, i.e. $l = \Theta a$. Notons Θ_V la substitution Θ restreinte aux seuls éléments de V et Θ_W celle restreinte aux seuls éléments de W. Alors $\Theta = \Theta_V \cup \Theta_W$ et $\Theta_V \cup \Theta_W = \emptyset$. Notons Θ' (resp. Ω', Θ'_V, Θ'_W)le renommage réciproque de Θ (resp. Ω, Θ_V, Θ_W). On considère alors tous les renommages d'une partie des variables de l avec des variables de r_n. Parmi ceux-ci se trouve Θ'_V. On va donc ajouter à C_k un candidat X de la forme $\langle r_n \Theta'_V l \rangle$. Montrons maintenant que ce candidat est équivalent à s à un renommage près. Comme Θ_W ne porte que sur des variables n'apparaissant pas dans r_n et que $V(l) \cap V(r_n) = \emptyset$, on a que Θ'_W ne porte pas non plus sur des variables de r_n. On peut donc appliquer Θ'_W à X. On obtient alors $\Theta'_W X = \langle r_n \Theta'_W \Theta'_V l \rangle = \langle r_n \Theta' l \rangle = \langle r_n a \rangle$. Il

ne nous reste alors plus qu'à appliquer Ω', d'où $\Omega'\langle r_n\ a\rangle = \langle \Omega' r_n\ \Omega' a\rangle = S_{n-1}\langle s_n\rangle = S_n$. Dans ce cas où $n = k$, nous ajoutons S_n (i.e. s) à C_k. Comme r_n est légal par rapport à E_0, que $E_{n-1} \xrightarrow{p_n} E_n$ est une transition de $\mathcal{A}$, et que E_n est final, nous avons que r_n est valide.

– Si $n \neq k$ et E_n non final, Nous avons que $r_n = \Omega S_{n-1}$, posons alors $a = \langle \Omega s_n\rangle$. Considérons maintenant les deux ensembles $V = V(a) \cap V(r_n)$ et $W = V(a) \cup V(r_n)$. l est un renommage de s_n donc c'est aussi un renommage de a par une substitution Θ, i.e. $l = \Theta a$. Notons Θ_V la substitution Θ restreinte aux seuls éléments de V et Θ_W celle restreinte aux seuls éléments de W. Alors $\Theta = \Theta_V \cup \Theta_W$ et $\Theta_V \cup \Theta_W = \emptyset$. Notons Θ' (resp. Ω', Θ'_V, Θ'_W)le renommage réciproque de Θ (resp. Ω, Θ_V, Θ_W). On considère alors tous les renommages d'une partie des variables de l avec des variables de r_n. Parmi ceux-ci se trouve Θ'_V.
On va donc faire un appel récursif avec les paramètres $b_{n+1} = E_n$, $B_{n+1} = B_n \cup E_{n-1}$, et un motif séquentiel logique r_{n+1} de la forme $\langle r_n \Theta'_V l\rangle$. Montrons maintenant que r_{n+1} est équivalent à S_n à un renommage près. Comme Θ_W ne porte que sur des variables n'apparaissant pas dans r_n et que $V(l) \cap V(r_n) = \emptyset$, on a que Θ'_W ne porte pas non plus sur des variables de r_n. On peut donc appliquer Θ'_W à r_{n+1}. On obtient alors $\Theta'_W r_{n+1} = \langle r_n \Theta'_W \Theta'_V l\rangle = \langle r_n \Theta' l\rangle = \langle r_n a\rangle$. Il ne nous reste alors plus qu'à appliquer Ω', d'où $\Omega'\langle r_n\ a\rangle = \langle \Omega' r_n\ \Omega' a\rangle = S_{n-1}\langle s_n\rangle = S_n$. Comme r_n est légal par rapport à E_0, et que $E_{n-1} \xrightarrow{p_n} E_n$ est une transition de $\mathcal{A}$, nous avons que r_n est légal par rapport à E_0.

D'où $P(n+1)$ vraie. Donc s a bien été ajouté à C_k d'où la complétude.

La preuve précédente montre également la correction de cet algorithme. En effet, on voit qu'à chaque fois que l'on ajoute un motif à C_k il s'agit d'un motif séquentiel logique valide de taille k. □

Élagage des candidats. Nous élaguons un motif séquentiel logique candidat $s \in C_k$ s'il contient un sous-motif séquentiel logique valide qui ne s'unifie avec aucun motif séquentiel logique de F qui est valide et fréquent. Pour trouver les sous-motifs séquentiels logiques valides d'un motif candidat s, nous utilisons l'algorithme FindMaxSubseq avec S contenant l'état initial de l'automate $\mathcal{A}$ et E égal à l'ensemble des états finaux de $\mathcal{A}$.

Condition d'arrêt. Pour une étape j de l'algorithme 4, nous nous arrêtons si tous les ensembles $F_j, \ldots, F_{j+|\mathcal{A}|-1}$ sont vides ($|\mathcal{A}|$ est le nombre d'états de l'automate $\mathcal{A}$). En effet, si nous considérons un motif séquentiel logique valide et fréquent s de longueur supérieure à $j + |\mathcal{A}| - 1$, alors il doit contenir une boucle de longueur au plus $|\mathcal{A}|$ et ainsi un sous-motif séquentiel logique valide et fréquent de longueur au moins égale à j. Ainsi, si aucun motif valide de longueur supérieure ou égale à j n'est fréquent (étant donné que nous avons supposé que $F_j, \ldots, F_{j+|\mathcal{A}|-1}$ sont vides), alors s ne peut être fréquent.

4.3 Résultats expérimentaux

Nous avons réalisé quelques expériences sur les différentes versions de SPIRIT-LoG avec des données obtenues grâce à un générateur aléatoire. Nous détaillons le fonctionnement de ce dernier. Il génére tout d'abord un certain nombre de constantes, puis il génère des symboles de prédicat d'arité diverses, ce qui permet de construire des atomes clos. Etant donné un nombre

de séquences d'entrée à construire et un nombre maximal d'atomes par séquence d'entrée, le générateur peut alors construire la base de données de séquences d'entrée.

Pour chaque variante, des expériences ont montré l'impact des différentes contraintes et nous ont permis de mesurer l'équilibre entre l'élagage à base de support et l'élagage dû à la prise en compte d'une contrainte sous forme d'expression rationnelle. Conformément à ce qui était attendu, les résultats obtenus suivent la même tendance que ceux obtenus avec des motifs séquentiels d'items [GRS02]. Cependant, nous pouvons remarquer que l'usage de motifs séquentiels logiques amplifie sensiblement certains résultats. En effet, hormis le cas de l'utilisation de très petits automates (aussi bien en terme de nombre d'états que de transitions), SPIRIT-LºG(R) devient très rapidement inintéressant, parce que le nombre de candidats générés explose rapidement. Cela est dû au fait que l'étape de génération des candidats appelle deux fois la fonction Enum-Subst, qui est très coûteuse dans le cas de cette variante.

Comme c'était le cas avec SPIRIT, nous pouvons dire que plus la contrainte relaxée est faible, plus l'élagage basé sur la contrainte de fréquence est important et moins l'élagage basé sur la contrainte syntaxique sous forme d'expression rationelle est efficace. De plus, plus la contrainte syntaxique est sélective (petit automate) ; plus les versions peu relaxées de la contrainte initiale sont efficaces. Généralement, SPIRIT-LºG(V) et SPIRIT-LºG(L) permettent d'obtenir les meilleurs équilibres entre les deux types d'élagage. En fait, dans de nombreuses expériences, SPIRIT-LºG(L) a donné des résultats légèrement meilleurs que SPIRIT-LºG(V). Cela est dû au fait que le nombre de renommage de variables utilisés dans l'étape de génération des candidats de SPIRIT-LºG(L) est généralement plus faible que celui constaté pour la même étape dans SPIRIT-LºG(V) (Dans SPIRIT-LºG(L), nous avons seulement à considérer des renommages avec les variables apparaissant dans le premier atome, alors que dans SPIRIT-LºG(V), nous devons considérer des renommages avec toutes les variables apparaissant dans le motif séquentiel logique à étendre.

Sur une base de données de 3000 séquences contenant 50 éléments au maximum chacunes, et où l'arité des prédicats ne dépasse pas 3, nous avons obtenu les courbes de la Figure 4.5. Comme on s'y attendait, pour de faibles valeurs de support, le nombre de candidats générés par SPIRIT-LºG(R) explose. Avec une fréquence minimale de 0.1%, plus de 150 000 motifs séquentiels logiques candidats de longueur 6 ont été générés et nous avons manqué de mémoire. SPIRIT-LºG(L) et SPIRIT-LºG(V) ont fourni les meilleurs résultats. Malgré un nombre de candidats générés plus important, l'élagage utilisé par SPIRIT-LºG(V) est très efficace. Ainsi, le nombre de candidats dont le support est effectivement compté reste relativement faible pour les variantes L et V.

La Figure 4.5 montre que les contraintes sont utiles pour augmenter l'efficacité de SPIRIT-LºG. En effet, le nombre de candidats dont le support est compté pour SPIRIT-LºG(L) et SPIRIT-LºG(V) reste plus petit que celui induit par SPIRIT-LºG(N).

4.4 Synthèse

Dans le Chapitre 3, nous avons vu l'utilité des scénarios prototypiques pour faciliter la transmission de savoir-faire entre experts de l'extraction de connaissances et les utilisateurs finaux et la mise en évidence de problèmes algorithmiques pour lesquels des solutions efficaces restent à concevoir. Nous avions ainsi vu sur un exemple jouet, des conjonctions de contraintes intéressantes sur deux types de motifs séquentiels. Plus précisément, elles portaient sur la

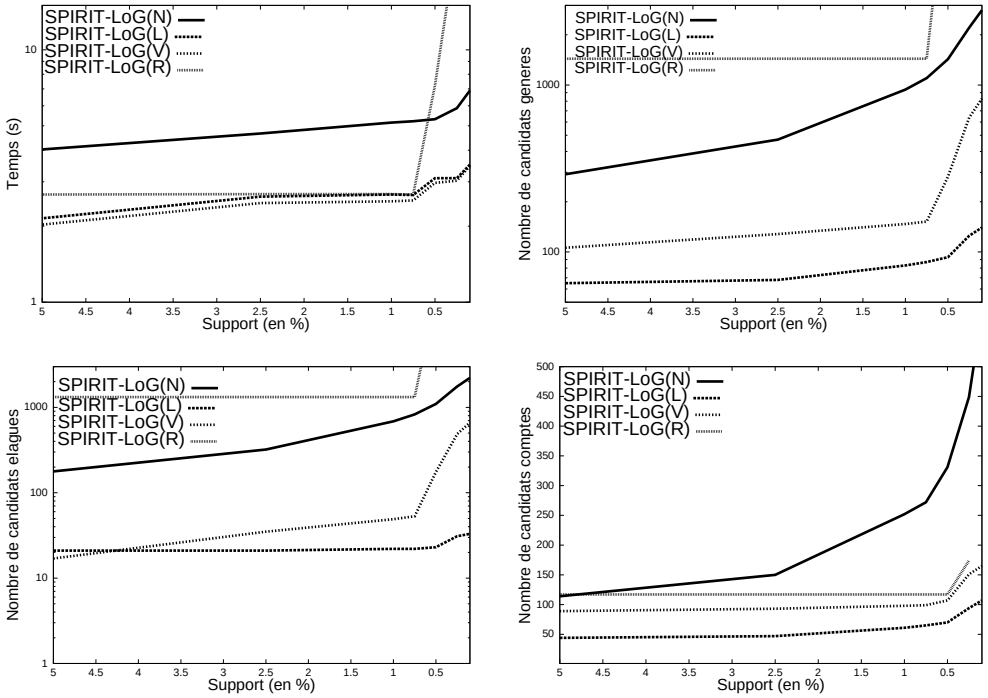

FIG. 4.5 – Résultats avec des données synthétiques

conjonction d'une contrainte de fréquence minimale avec une contrainte syntaxique exprimée par l'utilisateur. Dans les deux cas, cette dernière contrainte ne présentait pas une propriété classique d'anti-monotonie, de monotonie ou de convertibilité, habituellement utile lors d'une extraction par niveaux. C'est pourquoi nous avons choisi de relaxer ces contraintes syntaxiques en des contraintes moins fortes mais pour lesquelles nous avons été capables de mettre au point des techniques de génération et d'élagage des candidats intéressantes. Nous avons montré l'intérêt de l'approche par relaxation de contraintes sur des jeux de données synthétiques.

Du point de vue de l'efficacité, il est ainsi généralement préférable de travailler avec une contrainte même relaxée que sans contraintes du tout, car cela permet de réduire même de façon modeste l'espace de recherche à explorer. Si on ne peut relaxer une contrainte, on est alors obligé d'utiliser une stratégie de post-traitement, qui consiste à ne prendre en compte pour l'extraction que les autres contraintes apparaissant dans la requête. On obtient alors un sur-ensemble de la collection de motifs recherchée que l'on est obligé de filtrer pour parvenir au résultat désiré. Cependant, une telle technique n'est pas toujours possible, les autres contraintes prenant part à l'extraction n'étant pas toujours suffisamment sélectives pour permettre une élagage efficace des motifs candidats. Lors de la conception de systèmes de bases de données inductives, ces aspects seront donc à prendre en compte, il faudra dans l'idéal disposer d'un moyen d'évaluer la sélectivité des contraintes et également que le système puisse décider pour l'exécution d'une requête de considérer une relaxation d'une contrainte et de filtrer les résultats obtenus pour aboutir à la solution. Par exemple, dans le cas de GALIBOT, face à une conjonction de contraintes $C_{MinFreq(D,t)} \wedge C_{Sim(ref)}$, nous utiliserons un algorithme portant sur la conjonction $C_{MinFreq(D,t)} \wedge C_{Sim-Pot(ref)}$ qui renvoie un sur-ensemble de ce que nous cherchons.

Dans le chapitre suivant, nous allons nous intéresser à une autre fonctionnalité intéressant des scénarios d'ECD : les scénarios d'évaluation. Le but de ce type de scénario est de mettre en place des outils d'évaluation quantitative et qualitative des solutions d'extraction de connaissances dans les données. L'objectif à terme est d'arriver à la mise au point de benchmarks pour les bases de données inductives, dans la lignée de ceux existants pour les bases de données "classiques".

Chapitre 5

Application : scénarios d'évaluation

Dans le chapitre 3, nous avons présenté la notion de scénarios d'ECD. Nous avons présenté deux déclinaisons de ceux-ci :

- les scénarios prototypiques, essentiellement tournés vers le transfert de savoir-faire entre utilisateurs de systèmes d'ECD. Par leur formalisation, ils permettent également de mettre en évidence des conjonctions ou disjonction particulières de contraintes pour lesquels nous ne disposons pas d'algorithmes adaptés ou efficaces. Nous avons vu dans le chapitre précédent deux algorithmes destinés à résoudre les problèmes soulevés par les scénarios prototypiques présentés au Chapitre 3.
- les scénarios d'évaluation, tourné vers le "benchmarking" aussi bien qualitatif que quantitatif des solutions d'ECD. Le but n'est pas seulement d'évaluer les performances d'un système d'ECD à travers des mesures d'efficacité objectives, mais aussi d'analyser les stratégies mises en oeuvre pour résoudre le problème posé par le scénario.

Dans la première partie de ce chapitre, nous présentons un premier exemple de scénario d'évaluation pour l'extraction de connaissance dans les données. Il est adapté du scénario prototypique présenté dans le Chapitre 3, mais nous nous intéressons plus particulièrement ici à des difficultés algorithmiques que nous souhaitons mettre en évidence. Plus précisément, nous nous intéresserons à l'extraction de motifs séquentiels vérifiant une contrainte syntaxique, à l'extraction de représentations condensées et au besoin de méthodes efficaces pour le traitement des requêtes ensemblistes dans le cadre des bases de données inductives. Nous montrerons une exécution possible du scénario avec des outils développés dans notre équipe. Pour certaines de ces étapes, nous verrons qu'il est possible d'utiliser des solutions alternatives, ce qui permet de mettre en évidence le fait qu'il peut exister plusieurs solutions possibles à un problème donné. La dernière étape du scénario pointe la nécessité de disposer de méthodes efficaces pour les recherches ensemblistes dans les ensembles d'items extraits et stockés sous forme relationnelle. C'est pourquoi nous présenterons une technique d'indexation destinée à optimiser ce genre de traitements.

5.1 Un scénario d'évaluation en bioinformatique

5.1.1 Présentation du jeu de données

Nous utilisons un jeu de données réel issu de notre pratique dans le champ de la bioinformatique [BBJ+02]. Il consiste en l'analyse de données d'expression de gènes obtenues grâce à

la technique SAGE[1]. Nous considérons une matrice d'expression pour le transcriptome étendu dans un ensemble de tissus humains. Nous avons un total de 90 situations biologiques pour lesquelles nous disposons des mesures du niveau d'expression de 12 636 gènes. De plus, nous disposons des informations suivantes sur les gènes étudiés : le tag (c'est-à-dire une séquence de nucléotides caractérisant le gène), une description du gène (par exemple, son nom scientifique), ainsi que l'identifiant du cluster Unigène depuis lequel il a été obtenu (cela permet de se connecter à d'autres bases de données, comme Gene Ontology (`http ://www.geneontology.org`), afin d'obtenir une information détaillée sur le gène en question. Plus précisément, nous avons deux tables, *expression* et *identification*, qui ont les schémas suivants :

- $expression(situation\_id, gene_1, gene_2, \ldots, gene_{12636})$: elle contient les valeurs d'expression de gènes pour chaque situation biologique.
- $identification(gene\_id, name, tag, cluster\_unigene)$: elle contient les informations sur les différents gènes. Remarquons au passage que l'information sur les tags est de type ordonné (étant donné que l'ordre des nucléotides est important). Ainsi, il est possible, après quelques transformations de la table *identification* d'obtenir un contexte d'extraction pour les motifs séquentiels comme celui décrit dans le chapitre 3.

La Figure 5.1 nous donne un aperçu de ces tables.

gene_id	*name*	*tag*	*cluster_unigene*
1	ESTs	AAAAAAAAAT	Hs.114428
2	chemokine-like factor 1	AAAAAAAAGA	Hs.15159
3	suppressor of G2 allele ...	AAAAAAACTA	Hs.5169
4	Homo sapiens, clone ...	AAAAAAAGCA	Hs.147025
...	...	...	...
7427	Homo sapiens cDNA : FLJ21312 ...	GGAACCTATC	Hs.351781
...	...	...	...

Extrait de la table *identification*

situation_id	$gene_1$	$gene_2$	$gene_3$	$gene_4$	...	$gene_{12635}$	$gene_{12636}$
1	14	0	4	4	...	0	0
2	4	0	9	0	...	0	0
3	0	10	0	0	...	0	0
4	8	8	8	8	...	4	0
...	...	...	...	...	...	...	...

Extrait de la table *expression*

FIG. 5.1 – Exemple de données SAGE

5.1.2 Présentation du scénario d'évaluation

Le scénario que nous allons considérer est constitué de six étapes principales.

A. Génération des contextes d'extraction. Nous devons tout d'abord obtenir des jeux de données vérifiant notre définition de contexte d'extraction. La table *expression* a déjà presque

[1] `http ://www.ncbi.nlm.nih.gov/SAGE/index.cgi`

le bon format, étant donné que nous pouvons considérer les équivalences suivantes : $\mathcal{A}_i = gene_i, \forall i \in \{1, \ldots, 12636\}$ et $tuple\_id = situation\_id$. En fait, nous avons juste besoin de coder des propriétés booléennes afin de distinguer les gènes surexprimés des gènes sous-exprimés. Pour cela, nous proposons d'utiliser une méthode basée sur le niveau d'expression moyen de chaque gène. Supposons que le niveau d'expression moyen d'un gène g soit M_g, alors nous remplacerons les niveaux d'expression de g supérieurs à M_g par 1 (sur-expression) et nous mettrons 0 sinon.
(1) **create dataset** M **as** $Binarize\_average(expression)$

De plus, nous avons besoin de transformer la table $identification$ afin d'obtenir un nouveau contexte d'extraction compatible avec l'extraction de motifs séquentiels sur les tags EST. Pour rester cohérent avec les définitions données dans le Chapitre 3, nous avons besoin d'un champ pour mettre les nucléotides du tag EST et d'un autre pour stocker leur position dans le tag. Après cette transformation, nous obtiendrons une table I_1 doté d'un schéma (*gene_id*, *position*, *nucleotid*), qui correspond à notre définition de contexte d'extraction. En effet, par rapport à la définition donnée au Chapitre 3, nous avons $ind\_attr = gene\_id$, $ordering\_attr = position$, $event\_attr = nucleotid$.
(2) **create dataset** I_1 **as** $Transform(identification)$

Dans la suite de cette partie, nous considérerons les deux ensembles d'items suivants : $\mathtt{T}_1 = \{(gene_i = 1), 1 \leq i \leq 12636\}$ et $\mathtt{T}_2$ l'ensemble des descripteurs basés sur l'attribut *nucleotid* dans la table I_1, i.e. $\mathtt{T}_2 = \{(nucleotid = p), p \in \{A, C, G, T\}\}$. La Figure 5.2 donne un aperçu des contextes d'extraction obtenus à cette première étape.

gene_id	*position*	*nucleotid*
...	...	...
7427	1	G
7427	2	G
7427	3	A
7427	4	A
7427	5	C
7427	6	C
7427	7	T
7427	8	A
7427	9	T
7427	10	C
7428	1	G
...	...	...

situation_id	$gene_1$	$gene_2$	$gene_3$	$gene_4$	...	$gene_{12636}$
1	1	0	0	0	...	0
2	0	0	1	0	...	0
3	0	0	0	0	...	0
4	0	0	0	1	...	0
...	...	...	...	...	...	...

FIG. 5.2 – Contextes d'extraction I_1 (à gauche) et M (à droite)

B. Extraction de motifs séquentiels avec une contrainte syntaxique. Nous réalisons l'extraction des motifs séquentiels de nucléotides fréquents dans I_1 et dont la séquence des items constitue un mot décrit par l'expression régulière $\mathcal{R} : T(C|G)^*A(C|G)^*T$. Le seuil de fréquence minimale que nous utiliserons est 2%. Nous obtenons ainsi un premier ensemble de motifs séquentiels P_1.
(3) **create pattern set** P_1 **as** $s \in P_1 \wedge s \in \mathcal{S}(\mathtt{T}_2) \wedge C_{MinFreq(I_1,0.02)}(s) \wedge C_{ER(\mathcal{R})}(s)$

C. Croisement entre données et motifs Nous réalisons maintenant une opération de croisement entre l'ensemble de motifs séquentiels P_1 et le contexte d'extraction I_1. Plus précisément, nous souhaitons séparer I_1 en deux nouveaux contextes d'extraction : I_{pos} qui sera uniquement

situation_id	$gene_1$	$gene_4$	...	$gene_{12635}$
1	1	0	...	0
2	0	0	...	0
3	0	0	...	0
4	0	1	...	0
...	...	...	...	...

situation_id	$gene_2$	$gene_3$	...	$gene_{12636}$
1	0	0	...	0
2	0	1	...	0
3	0	0	...	0
4	0	0	...	0
...	...	...	...	...

FIG. 5.3 – Exemple de contextes M_{pos} (à droite) et M_{neg} (à gauche)

constitué des séquences d'entrée contenant au moins un motif de P_1, et I_{neg} qui contiendra les autres séquences d'entrée.
(4) **create data set** I_{pos} **as** $\alpha(I_1 P_1)$
(5) **create data set** I_{neg} **as** $\neg\alpha(I_1 P_1)$

D. Création de nouveaux contextes d'extraction Nous voulons diviser le contexte d'extraction M en deux contextes d'extraction plus petits pour l'extraction d'ensembles d'items fréquents : l'un contiendra l'information sur l'expression des gènes apparaissant dans I_{pos} et l'autre sur ceux de I_{neg}. Chacun d'entre eux sera donc une projection de M sur les attributs apparaissant comme identifiants de tuples dans les contextes I_{pos} ou I_{neg}.
(6) **create data set** M_{pos} **as** $\Pi_{\sigma_{gene\_id}(I_{pos})}(M)$
(7) **create data set** M_{neg} **as** $\Pi_{\sigma_{gene\_id}(I_{neg})}(M)$

Exemple. Supposons que I_{pos} contienne entre autres les tags g_2, g_3, et g_{12636}, et que I_{neg} contienne entre autres les tags g_1 et g_4, alors la Figure 5.3 nous donne un exemple du contenu de M_{pos} et M_{neg}.

E. Extraction d'ensembles d'items clos Par la suite, nous voulons extraire les ensembles clos et fréquents de situations biologiques parmi les gènes surexprimés. Comme les matrices d'expression dont nous disposons contiennent les situations biologiques en ligne et les gènes en colonne, nous exprimerons donc les seuils de fréquence minimale dans les matrices transposées de M_{pos} et M_{neg}, que nous noterons respectivement M^T_{pos} et M^T_{neg}. Nous voulons plus précisément extraire des ensembles clos fréquents de situations à deux seuils de fréquence minimale : 1% et 0.5%. De plus, pour chaque seuil, nous avons ajouté une contrainte de taille minimale étant donné que nous sommes uniquement intéressés par les ensembles d'items contenant plus de 5 éléments.
(8) **create data set** $P_{Pos\_0.5}$ **as** $r \in P_{Pos\_0.5} \wedge r \in \mathcal{I}(\mathtt{T}_3) \wedge C_{MinFreq(M^T_{pos},0.005)}(r) \wedge C_{MinSize(5)}(r) \wedge C_{Close(M^T_{Pos})}(r)$
(9) **create data set** P_{Pos_1} **as** $r \in P_{Pos\_1} \wedge r \in \mathcal{I}(\mathtt{T}_3) \wedge C_{MinFreq(M^T_{pos},0.01)}(r) \wedge C_{MinSize(5)}(r) \wedge C_{Close(M^T_{Pos})}(r)$
(10) **create data set** $P_{Neg\_0.5}$ **as** $r \in P_{Neg\_0.5} \wedge r \in \mathcal{I}(\mathtt{T}_3) \wedge C_{MinFreq(M^T_{neg},0.005)} \wedge C_{MinSize(5)}(r) \wedge C_{Close(M^T_{Neg})}(r)$
(11) **create data set** P_{Neg_1} **as** $r \in P_{Neg\_1} \wedge r \in \mathcal{I}(\mathtt{T}_3) \wedge C_{MinFreq(M^T_{neg},0.01)} \wedge C_{MinSize(5)}(r) \wedge C_{Close(M^T_{Neg})}(r)$

F. Post-traitement Finalement, nous souhaitons post-traiter les quatre ensembles de résultats que nous avons obtenus à l'étape précédente. Plus précisément, nous voulons trouver des ensembles clos de situations biologiques qui peuvent être utilisés pour discriminer les deux matrices.

Pour cela, nous recherchons les ensembles d'items dans P_{Pos_1} qui ne sont sous-ensembles d'aucun ensemble d'items de $P_{Neg\_0.5}$. Nous avons choisi de considérer ce différentiel en fréquence (i.e. en utilisant $P_{Neg\_0.5}$ plutôt que P_{Neg_1} pour la recherche des sur-ensembles) afin d'augmenter le caractère discriminant des ensembles d'items trouvés. De plus, nous réalisons le même type de test dans l'autre sens en recherchant des ensembles d'items de P_{Neg_1} qui n'ont pas de sur-ensembles dans $P_{Pos\_0.5}$.
(14) **create pattern set** $Relevant\_Set$ **as** $\{p \in P_{Pos\_1} \mid \nexists q \in P_{Neg\_0.5}, p \subseteq q\} \cup \{p \in P_{Neg\_1} \mid \nexists q \in P_{Pos\_0.5}, p \subseteq q\}$

5.1.3 Une exécution possible de ce scénario d'évaluation

Dans cette partie, nous montrons une exécution possible du scénario d'évaluation présenté précédemment en utilisant les outils et algorithmes que nous avons développés et/ou sur lesquels nous avons travaillé.

Étape A.

Pour réaliser la transformation du schéma de la table $identification$ et la création du contexte booléen, nous pouvons utiliser un script écrit à cette fin.

Étape B.

Pour extraire les motifs séquentiels fréquents qui vérifient la contrainte syntaxique exprimée sous forme d'une expression rationnelle $\mathcal{R}$, nous pouvons utiliser un extracteur classique de motifs séquentiels fréquents et post-traiter les résultats afin de ne récupérer que les motifs pertinents. C'est ce que nous avons fait ici : nous avons d'abord extrait uniquement les motifs séquentiels fréquents dans le contexte d'extraction I_1 avec un seuil de fréquence minimale fixé à 2%, nous avons trouvé un total de 2499 motifs séquentiels. Ensuite, nous appliquons un filtre à ce premier ensemble de résultats afin de conserver seulement les motifs vérifiant la contrainte syntaxique dérivée de l'expression rationnelle $\mathcal{R}$. Nous pouvons alors utiliser un script pour cette étape et nous récupérons finalement 211 motifs séquentiels dans P_1.

Solution alternative : une autre solution est d'utiliser directement un algorithme conçu pour réaliser une exploitation active d'une contrainte syntaxique exprimée sous forme d'expression rationnelle. Comme cela a déjà été évoqué au chapitre précédent, la famille d'algorithmes SPIRIT [GRS02] peut réaliser une telle extraction. Ils utilisent à la fois la contrainte de fréquence minimale et celle à base d'expression rationnelle pour réduire l'espace de recherche des motifs candidats pendant l'extraction. Comme la contrainte syntaxique basée sur l'expression rationnelle n'est généralement ni monotone, ni anti-monotone, ils utilisent différentes relaxations de cette contrainte afin d'obtenir des propriétés plus intéressantes pour l'optimisation d'une extraction de motifs par niveaux. Remarquons, que dans le cas particulier des motifs séquentiels constitués d'items consécutifs (c'est-à-dire des chaînes), des améliorations intéressantes ont été réalisées depuis [ALB03].

Étape C.

Cette étape de croisement entre les données et les motifs peut facilement être réalisée à l'aide de quelques scripts *ad hoc*. Nous obtenons deux nouveaux contextes d'extraction :

– I_{pos} qui correspond à 5535 gènes (et nous avons donc 55350 lignes).
– I_{neg} qui correspond à 7101 gènes (et nous avons alors 71010 lignes).

Étape D.

Là encore, nous utilisons un script *ad hoc* pour réaliser la séparation de la matrice d'expression originale en deux nouveaux contextes d'extraction : M_{pos} et M_{neg}.

Étape E.

Comme nous souhaitons extraire des ensembles clos fréquents de situations biologiques dans les matrices d'expression M_{pos} and M_{neg}, nous devons d'abord modifier la disposition de la matrice afin d'avoir les gènes en ligne et les situations biologiques en colonne. Une telle transformation peut être réalisée à l'aide de scripts, étant donné que le format que nous utilisons est relativement simple (pour chaque transaction, nous avons la liste de tous les identifiants de colonne pour lesquelles on a un '1'). Ensuite, nous utilisons le prototype ACMiner prototype pour extraire les ensembles libres fréquents [BBR03]. Heureusement, il est relativement facile d'obtenir les ensembles clos fréquents à partir des ensembles libres fréquents, via une étape de post-traitement. De plus, nous n'avons en fait besoin de faire que deux extractions étant donné que nous avons la propriété suivante sur les ensembles de résultats :

– $r \in P_{Pos\_1} \leftarrow C_{MinFreq(M^T_{pos},0.01)}(r) \wedge C_{MinSize(5)}(r) \wedge C_{Close(M^T_{Pos})}(r)$
  Comme $C_{MinFreq(M^T_{pos},0.01)}(r) \Rightarrow C_{MinFreq(M^T_{pos},0.005)}$, nous avons :
  $r \in P_{Pos\_1} \leftarrow r \in P_{Pos\_0.5} \wedge C_{MinFreq(M^T_{pos},0.01)}(r)$
  i.e. nous avons juste à filtrer les résultats de $P_{Pos\_0.5}$ pour obtenir P_{Pos_1}.
– Nous avons une propriété duale pour P_{Neg_1} :
  $r \in P_{Neg\_1} \leftarrow r \in P_{Neg\_0.5} \wedge C_{MinFreq(M^T_{Neg},0.01)}(r)$
  i.e. nous pouvons filtrer $P_{Neg\_0.5}$ pour obtenir P_{Neg_1}.

Nous avons donc utilisé AC Miner [BBR03] sur nos 2 matrices pour obtenir les ensembles libres fréquents et leur fermeture avec un seuil minimal de fréquence fixé à 0.5 %. Puis nous avons filtré les résultats pour obtenir les ensembles libres fréquents à un seuil de 1 % (cf. les résultats présentés dans la Table 5.1). La génération des ensembles clos et la sélection des ceux ayant une taille supérieure à 5 peut être réalisée dans la même étape de post-traitement. Nous avons donc écrit un script pour le post-traitement des ensembles de résultats (P_{pos_1} , P_{neg_1}, $P_{pos\_0.5}$ et $P_{neg\_0.5}$) afin de régénérer les ensembles clos contenant plus de 5 items. Nous avons obtenu les résultats présentés dans la Table 5.1.

Solution alternative : Nous pouvons utiliser le prototype `DMiner` décrit dans [BRBR05] pour extraire directement des concepts à partir des matrices originales M_{Pos} et M_{neg}. Plus

Seuil	M_{Pos}	M_{neg}
0.5 %	201 871	251 548
1 %	26 723	31 054

Nombre d'ensembles libres fréquents

Seuil	M_{pos}	M_{neg}
0.5 %	28 663	71 797
1 %	154	661

Nombre d'ensembles clos vérifiant la contrainte de taille

Tab. 5.1 – Résultats pour l'extraction d'ensembles d'items

précisément, les concepts sont des tuples constitués d'ensembles clos à la fois sur les lignes et les colonnes (i.e. les gènes et les situations biologiques). Cet algorithme exploite le fait que les ensembles clos sur les lignes et les colonnes sont fortement liés dans un contexte d'extraction. Ainsi, nous n'avons plus besoin de transposer la matrice pour obtenir les ensembles clos de situations. De plus, ce prototype peut gérer des contraintes de taille minimale sur les ensembles d'items, ce qui nous épargne une étape de post-traitement par rapport à ce que nous avons fait avec ACMiner. Plus formellement, `DMiner` gère directement la conjonction de contraintes $C_{MinFreq(M,t)} \wedge C_{Close(M)} \wedge C_{Minsize(m)}$. Cependant, il n'est pas possible d'extraire des ensembles clos qui soient fréquents dans un contexte d'extraction et non fréquents dans un autre (à l'instar du prototype `Molfea` [dRK01] [2]). Une telle fonctionnalité serait extrêmement utile étant donné que cela nous permettrait de réaliser la dernière étape de notre scénario dans la même exécution de l'algorithme.

Étape F.

La méthode choisie pour discriminer les deux matrices est basée sur leur caractérisation à l'aide d'ensembles fréquents de situations biologiques : nous voulons trouver des ensembles de situations qui sont présents dans P_{Pos_1} (resp. P_{Neg_1}) et qui ne sont sous-ensembles d'aucun ensemble de situations dans $P_{Neg\_0.5}$ (resp. $P_{Pos\_0.5}$). Ainsi, nous devons réaliser de nombreux tests d'inclusion entre les différents ensembles d'items stockés dans les 4 ensembles de résultats différents que nous avons obtenus à l'étape précédente. Si nous considérons le cadre des bases de données inductives, les motifs et les données doivent être stockés dans un cadre de travail commun. Si nous regardons quelle est la pratique actuelle dans ce domaine, il est clair que souvent, les données sont stockées sous forme de tables relationnelles dans un système de gestion de base de données (comme dans le système `MINE RULE` [MPC98], par exemple). Ainsi, sous ces restrictions, il est assez logique de stocker aussi les motifs extraits dans des bases de données relationnelles. Cependant, il est connu que les requêtes ensemblistes et notamment celles chargées de rechercher un sur-ensemble d'un certain ensemble d'enregistrements sont difficiles à réaliser dans le cadre de l'algèbre relationnelle (opérateur de division). C'est pourquoi nous avons besoin de techniques de stockage et d'indexation efficaces sur les ensembles d'items afin d'optimiser le traitement de ce genre d'opérations.

Une première technique pour l'optimisation des requêtes de recherche de sur-ensembles parmi des ensembles d'items dans les implémentations de bases de données inductives au-dessus des SGBDR a été proposée dans [MZ98] : il repose sur l'utilisation de clés bitmap pour résumer le contenu des ensembles d'items et accélérer ainsi les tests d'inclusion. Dans [MRB04], nous avons proposé une amélioration de cette technique d'indexation en utilisant une structure d'arbres préfixes (qui est communément utilisée dans de nombreux algorithmes d'ensembles d'extraction) comme moyen d'indexation des clés bitmap et des ensembles d'items. C'est cette technique que nous allons présenter dans la deuxième partie de ce Chapitre. Les résultats expérimentaux ont montré l'efficacité de notre méthode, en particulier lorsque l'on travaille avec des ensembles d'items qui peuvent apparaître plusieurs fois à des endroits différents dans les ensembles résultats. Comme la régénération des ensembles clos à partir des ensembles libres peut conduire à de telles répétitions dans les résultats, nous avons utilisé cette technique pour la dernière étape de notre scénario d'évaluation.

[2] http ://utah.informatik.uni-freiburg.de/cgi-bin/molfea.cgi

Comme résultat final, nous avons trouvé que 83 ensembles clos dans P_{pos_1} n'étaient pas sous-ensembles d'éléments de $P_{Neg\_0.5}$. De manière duale, 484 ensembles d'items de P_{neg_1} n'étaient pas sous-ensembles de motifs de $P_{Pos\_0.5}$.

5.2 Recherche ensembliste sur les ensembles d'items

5.2.1 Les limites de l'algèbre relationnelle

Dans le cadre des bases de données relationnelles, les opérations ensemblistes sont généralement des tâches difficiles à réaliser. Nous nous intéressons ici plus particulièrement aux requêtes de sur-ensembles, c'est-à-dire la découverte des ensembles d'items dans lesquels un ensemble d'items connu est inclus. Les requêtes de recherche de sur-ensembles peuvent être exprimées via l'utilisation de l'opérateur de division en algèbre relationnelle. Cependant, SQL ne l'implémente pas et il est nécessaire d'utiliser des requêtes faisant appel à plusieurs jointures pour obtenir un résultat équivalent. Les résultats expérimentaux figurant dans la Section 5.4 confirment que de telles requêtes impliquant de nombreuses jointures sont peu efficaces pour la recherche de sur-ensembles. Précisons que nous ne nous intéressons pas ici au problème de l'utilisation seule de SQL pour la recherche d'ensembles d'items fréquents, qui fait elle aussi massivement appel à l'évaluation de requêtes du type qui nous intéresse [STA00, Ran04]. La Figure 5.4 montre un exemple d'une requête SQL qui récupère les sur-ensembles de l'ensemble d'items $\{5, 8\}$.

Table des ensembles d'items

set_id	item_id
1	2
1	5
1	8
2	10
2	8
2	7
3	3
3	4

Requête SQL

```
SELECT a.itemset_id
FROM Itemset a, Itemset b
WHERE a.set_id = b.set_id
AND a.item_id = 5
AND b.item_id = 8
```

Résultat SQL

itemset_id
1

FIG. 5.4 – Une table contenant des ensembles d'items (à gauche) et une requête SQL retournant les sur-ensembles de $\{5, 8\}$ (à droite).

Morzy et al. [MZ98] ont proposé une méthode efficace pour le stockage et la manipulation des ensembles d'items dans les bases de données relationnelles. Celle-ci repose sur l'utilisation de clés bitmap qui synthétisent dans une chaîne de caractères le contenu d'un ensemble d'items. Nous proposons ici une nouvelle méthode pour le stockage et l'indexation des ensembles d'items dans les bases de données relationnelles tournée vers l'optimisation des requêtes de sur-ensembles. Elle est basée sur l'utilisation d'une structure de données qui encode partiellement la relation d'inclusion entre les ensembles d'items. Nos expériences sur des données réelles et synthétiques montrent une amélioration sensible des performances sur les requêtes de sur-ensembles. Nous rappelons les principes de méthode des clés bitmaps hachées ("hash group bitmap keys") de [MZ98] dans la Section 5.2.2, et notre technique dans la Section 5.2.3. Les expériences et leurs résultats sont présentés dans la Section 5.4.

5.2.2 Clés Bitmap Hachées

Afin d'optimiser les requêtes de sur-ensembles, [MZ98] a introduit l'idée d'une clé bitmap hachée associée à chaque ensemble d'items de la base de données et qui résume le contenu de l'ensemble d'items. Nous rappelons d'abord la notion de clé bitmap classique, puis les principes du hachage sur ces clés bitmap. Dans cette section, nous supposons que nous disposons de N items différents.

L'index bitmap d'un ensemble d'items S est un nombre binaire de longueur N dans lequel un bit à la position k a la valeur 1 si et seulement si $k \in S$. Toutes les différentes clés bitmap d'une collection d'un ensemble d'items peuvent ainsi être stockées dans une table d'index avec un identifiant de l'ensemble d'items qu'elles encodent. Par exemple, les clés bitmap associées aux ensembles d'items $\{0,3,5,9\}, \{2,5\}, \{1,4,6\}$ sont respectivement 1000101001, 100100, 1010010. La Figure 5.5 donne un exemple du stockage des clés bitmap pour la collection d'ensembles d'items $\{\{0,3,5,9\}, \{2,5\}, \{1,4,6\}\}$. Quand on procède à l'évaluation d'une requête de sur-ensemble, une clé bitmap B est calculée pour l'ensemble d'items recherché, puis un test d'inclusion est effectué avec l'opérateur logique AND entre B et l'ensemble des clés déjà présentes dans la base. Ainsi, pour chaque clé bitmap dans la base de données, nous avons juste à vérifier s'il y a un 1 à chaque position où il y a un 1 dans B. Par exemple, si nous recherchons les sur-ensembles de $\{4,6\}$ dans les ensembles d'items de l'exemple précédent, nous calculerons d'abord la clé bitmap associée à $\{4,6\}$, i.e. 1010000, puis nous la comparerons avec un opérateur AND avec chaque ensemble d'items. Nous trouverons ainsi que $\{1,4,6\}$ est un sur-ensemble de $\{4,6\}$. La figure 5.6 décrit la recheche des sur-ensembles de $\{4,6\}$.

Table relationnelle

itemset_id	**Item**
1	0
1	3
1	5
1	9
2	2
2	5
3	1
3	4
3	6

→ 1000101001

→ 0000100100

→ 0001010010

Index bitmap

itemset_id	Clé bitmap
1	1000101001
2	0000100100
3	0001010010

FIG. 5.5 – Exemple de clés bitmap

Cette définition des clés bitmap a essentiellement un intérêt théorique. En effet, dès que N prend des valeurs importantes (et c'est très rapidement la cas quand on considère des processus d'extraction tirés de la pratique), stocker toutes les clés bitmap consomme énormément de place, car chaque clé est de longueur N bits. De plus, cette technique n'est pas adaptée à un contexte dynamique, où de nouveaux items peuvent apparaître à travers différentes expériences En effet, supposons qu'un nouvel item soit ajouté, dans ce cas là, nous devons mettre à jour la longueur de toutes les clés binaires stockées dans la table. Ainsi, la maintenance de ce type d'index est relativement coûteuse et difficile. C'est pourquoi, dans [MZ98], Morzy and Zakrzewicz ont proposé le concept de clé bitmap hachée. L'idée est de considérer des clés bitmap de longueur

Subset
4
6

$\} \rightarrow 0001010000$ AND

itemset_id	Bitmap Key
1	1000101001
2	0000100100
3	0001010010

$\Rightarrow \{3\}$ est un sur-ensemble de $\{4, 6\}$

FIG. 5.6 – Recherche d'ensembles d'items en utilisant les clés bitmap

fixe n avec $n << N$. La clé bitmap hachée d'un ensemble d'items est créée avec toutes les clés hachées des items qu'il contient. La clé hachée d'un item X est une chaîne binaire de n bits calculée comme suit : $hash\_key(X) = 2^{X \ mod \ n}$. La Figure 5.7 illustre ce calcul de clé bitmap hachée pour la collection d'ensembles d'items $\{\{2, 5, 8\}, \{7, 8, 10\}, \{3\}\}$ avec $n = 5$.

ITEMSET table

itemset_id	item_id				bitmap_key
1	2	$\rightarrow$	00100		
1	5	$\rightarrow$	00001	$\rightarrow$	01101
1	8	$\rightarrow$	01000		
2	7	$\rightarrow$	00100		
2	8	$\rightarrow$	01000	$\rightarrow$	01101
2	10	$\rightarrow$	00001		
3	4	$\rightarrow$	01000	$\rightarrow$	01000

FIG. 5.7 – Exemple de calcul pour $n = 5$.

Pour la recherche des sur-ensembles d'un ensemble d'items X, une requête de sur-ensemble est évaluée en deux étapes :

1. On calcule la clé bitmap hachée de X et on la compare avec les clés bitmap hachées de chaque ensemble d'items de la base de données, au moyen d'une opération logique bit à bit de type AND. Puis, on retourne les identifiants des ensembles d'items qui satisfont ce test.
2. Ensuite, il faut vérifier que les identifiants d'ensembles d'items retournés par la première étape corresondent bien à des sur-ensembles de X. En effet, comme une clé bitmap hachée peut encoder différents ensembles d'items (à cause de la technique de hachage), il est possible d'obtenir de faux positifs. C'est pourquoi nous devons faire attention à ce point, et cela est réalisé par un test d'inclusion classique.

La Figure 5.8 montre l'évaluation d'une requête de recherche de sur-ensemble pour l'ensemble d'items $\{5, 8\}$.

L'idée consistant à utiliser une signature d'un ensemble d'items sous le forme d'une clé bitmap est utile, mais l'un des principaux inconvénients de cette méthode est que la recherche de sur-ensemble est conduite de manière "aveugle". En effet, à chaque fois que nous recherchons les sur-ensembles d'un ensemble d'items donné, nous devons vérifier toutes les clés bitmap pour les trouver. En fait, nous aurions besoin d'une méthode d'indexation sur les ensembles d'items et les

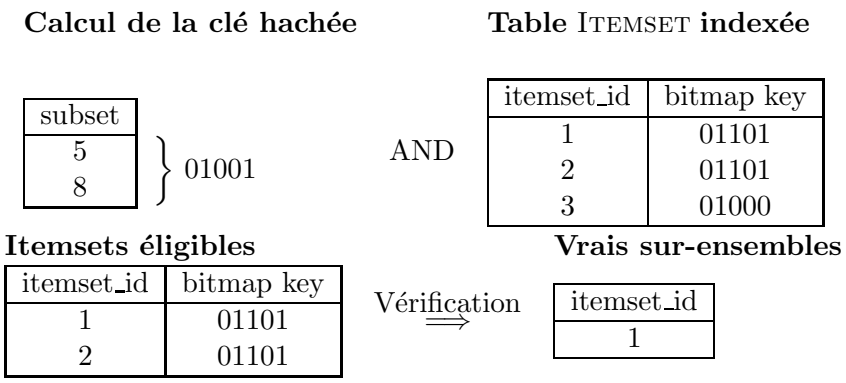

FIG. 5.8 – Recherche avec des clés bitmap hachées

clés bitmap qui puisse partiellement encoder la relation d'inclusion entre les ensembles d'items, ce qui permettrait d'accélerer l'examen des clés. C'est pourquoi dans la Section suivante, nous proposons une nouvelle méthode pour le stockage des ensembles d'items dans les bases de données relationnelles. Elle encode partiellement la relation d'inclusion entre les ensembles d'items en stockant une structure d'arbre d'ensembles d'items ("Itemset Tree" [HDR99]) dans une table relationnelle. Afin de rendre plus efficace la comparaison entre les ensembles d'items, les clés bitmap hachées sont également stockées dans la table.

5.2.3 Les arbres d'ensembles d'items

Les arbres d'ensembles d'items ont été proposés par Alladin Hafez et al. afin de permettre le stockage incrémental d'ensembles d'items lors de la recherche de règles d'association fréquentes et valides [HDR99]. Soit $I = \{i_1, i_2, \ldots, i_n\}$ un ensemble ordonné d'items. Chaque nœud s de l'arbre T représente soit un ensemble d'items rencontré, soit un de ses sous-ensembles. Dans les deux cas, ces ensembles sont ordonnés selon l'ordre défini sur I. Considérons alors deux ensembles d'items $s_i = \{a_1, \ldots, a_k\}$ et $s_j = \{b_1, \ldots, b_l\}$. On écrira $s_i \leq_o s_j$ si et seulement si $\forall p \in \{1, \ldots, \min\{k, l\}\}$, $a_p \leq b_p$. s_i est un sous-ensemble ordonné de s_j, dénoté par $s_i \subset_o s_j$, si et seulement si $s_i = \{a_1, \ldots a_k\}$, $s_j = \{a_1, \ldots, a_k, b_{k+1}, \cdots, b_l\}$ et $k < l$. L'insertion d'un nouvel item s dans T se faire de manière incrémentale et récursive. Le nœud racine r représente l'ensemble vide $\{\}$. Un ensemble d'item est inséré en examinant les fils du nœud racine r, ordonné par la relation $\leq_o$. La procédure d'insertion récursive se poursuit en considérant les différents cas suivant :

1. Si aucun des fils s_j de r n'a le même premier élément que s, alors s est inséré comme un fils de r et la procédure d'insertion s'arrête.
2. S'il existe un fils s_j de r tel que $s_j = s$, alors la procédure s'arrête.
3. S'il existe un fils s_j de r tel que $s \subset_o s_j$, alors s est inséré comme fils de r et comme parent de s_j. Puis la procédure s'arrête.
4. S'il existe un fils s_j de r tel que $s_j \subset_o s$, alors la procédure d'insertion est appelée récursivement avec s_j comme racine de l'arbre.
5. S'il existe un fils s_j de r tel que s et s_j partagent certains de leurs premiers éléments

($s \cap_o s_j \neq \emptyset$[3]), alors deux nouveaux nœuds sont insérés : un nœud $s_i = s \cap_o s_j$ comme fils de r et parent de s_j et un nœud s comme fils de s_i. Puis la procédure s'arrête.

La Figure 5.9 montre un exemple de la construction d'un arbre des ensembles d'items pour la collection $\{\{1,2\},\{4,6\},\{1,3,5\}\}$.

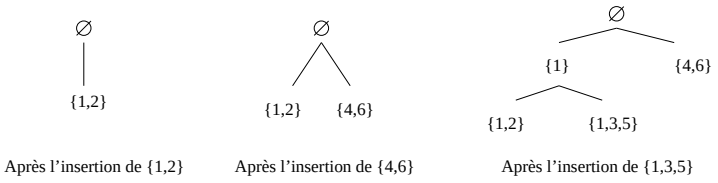

FIG. 5.9 – Exemple d'arbre d'ensembles d'items

5.3 Contribution : arbres d'ensembles d'items et SGBDR

En partant des travaux de Hafez et al., nous souhaitons stocker des ensembles d'items dans des bases de données relationnelles de telle sorte que les relations d'inclusions ordonnées qui existent entre eux soient préservées. Comme nous sommes intéressés par l'efficacité du scan des clés bitmap proposé par Morzy et al. lors de la recherche de sur-ensembles, nous voulons aussi stocker dans l'arbre des ensembles d'items la clé bitmap hachée associée à chacun des ensembles d'items. Pour cela, nous proposons d'utiliser le schéma relationnel suivant : (*Itemset_Id int*, *Nb_Items int*, *HBitmap_Key string*, *Ancestor int*, *First_Child int*, *First_Sibling int*, *Pattern bool*).

ITEMSET_ID	NB_ITEMS	HASH_BITMAP_KEY	ANCESTOR	FIRST_CHILD	FIRST_SIBLING	PATTERN
(num)	(num)	(String)	(num)	(num)	(num)	(yes/no)

Chaque ligne correspond à un ensemble d'items ou à un sous-ensemble d'un ensemble d'items. Dans le premier cas, le champ PATTERN prend la valeur "yes", sinon il prend la valeur "no". Chaque ligne est identifiée par une unique valeur ITEMSET_ID qui est aussi utilisée comme identifiant dans le champ ANCESTOR qui pointe sur le nœud parent, dans le champ FIRST_CHILD qui pointe sur le premier fils du nœud courant, et dans le champ FIRST_SIBLING qui pointe sur le premier frère du nœud courant. Le champ NB_ ITEMS contient le nombre d'items de l'ensemble d'items. La clé bitmap hachée de l'ensemble d'items est stockée dans le champ HBITMAP_KEY et l'ensemble d'items est encodé dans une table à part, qui possède, elle, un schéma classique (*Itemset_Id int*, *item int*). La Table 5.2 montre l'encodage de l'arbre des ensembles d'items pour la collection d'ensembles d'items $\{\{1,2\},\{4,6\},\{1,3,5\}\}$.

Nous utilisons la structure d'arbre des ensembles d'items pour résoudre les requêtes de sur-ensemble sans avoir à scanner toutes les lignes de la table des ensembles d'items. L' algorithme 8 présente le principe de la résolution d'une requête de sur-ensemble : *node* est un nœud de l'arbre des ensembles d'items et correspond à un enregistrement de la table encodant l'arbre, le champ *itemset* dénote l'ensemble d'items associé. La fonction est récursive : pour un nœud, elle scanne tous ses frères. Quand un frère est trouvé comme étant un sur-ensemble d'une partie de l'ensemble d'items, un appel récursif est réalisé sur le premier fils de ce frère. C'est ici que

[3] $s_i \cap_o s_j = \{a_1, \cdots, a_k\}$ iff $s_i = \{a_1, \cdots, a_k, a_{k+1}, \cdots, a_p\}$ and $s_j = \{a_1, \cdots, a_k, b_{k+1}, \cdots, b_q\}$ with $a_{k+1} \neq b_{k+1}$

Item--set_Id	Nb_items	HBitmap Key	Ancestor	First_ Child	First_ Sibling	Pattern
1	0	000000	NULL	4	NULL	No
2	2	000110	4	NULL	5	Yes
3	2	010001	1	NULL	NULL	Yes
4	1	000010	1	2	3	No
5	3	101010	4	NULL	NULL	Yes

Itemset_Id	Item
2	1
2	2
3	4
3	6
4	1
5	1
5	3
5	5

TAB. 5.2 – Codage d'un arbre d'ensembles d'items dans une table

nous utiliserons une technique à base de clés bitmap hachées pour vérifier que l'ensemble d'items en cours d'analyse est bien un sur-ensemble de l'ensemble d'items recherché. La recherche en profondeur est arrêtée quand nous trouvons dans l'ensemble d'items initial un item qui n'apparaît pas dans le sur-ensemble candidat, alors qu'un item plus grand y apparaît. Le fait que les items soient ordonnés est utile ici afin de simplifier la vérification. Quand on trouve un nœud dans lequel l'ensemble d'items recherché est inclus, on ajoute automatiquement au résultat tous les fils de ce nœud qui sont marqués comme étant des ensembles d'items. Cette tâche est réalisée par la fonction *Itemsets_in_Subtree*.

Algorithme 8: Algorithme de recherche dans un arbre préfixe d'ensembles d'items

Données : *Subset* est un ensemble d'items, *node* est un nœud de l'arbre
Résultat : results, l'ensemble des sur-ensembles de Subset
current = node ; // e.g. node={1,3,5}
répéter
  brother = current.first_sibling;
  si *Subset* $\subseteq$ *current.itemset* ***et*** *current.pattern=yes* **alors**
    // nous pouvons ajouter tous les ensembles d'items du sous-arbre
    results = results $\cup$ *Itemsets_in_Subtree*(current) ;
  sinon
    si $\forall a \in Subset$ *t.q.* $a \notin current.itemset$, $\nexists\ b \in current.itemset, a < b$ **alors**
      //Appel récursif sur le sous-arbre ; results=results$\cup$Search_Tree(current.first_child) ;
    fin
  fin
  current = brother ;
jusqu'à *current* $\neq$ *null*;
retourner *results*

5.4 Expériences

Nous avons réalisé des expériences à la fois sur des données réelles et des données synthétiques. Nos expériences ont été conduites avec le SGBDR MySQL[4]. Nous avons utilisé trois méthodes différentes pour rechercher des sur-ensembles d'ensembles d'items dans la base de données : (1) la méthode utilisant l'arbre des ensembles d'items (cf. Section 5.2.3), (2) la méthode utilisant des clés bitmap hachées [MZ98], (3) une requête SQL classique. Nous avons utilisé le générateur de données du projet Quest d'IBM [5] pour générer des données synthétiques avec les paramètres suivants : n_trans (le nombre d'ensembles d'items) était égal à 20000, n_items (le nombre d'items différents) à 1000, n_pats (le nombre de motifs différents) à 10000, $patlen$ (la longueur moyenne des motifs maximaux) à 4 et $corr$ (le corélation entre les motifs) à 0.25.

D'abord, nous avons analysé le comportement des différentes méthodes par rapport à la taille de l'ensemble d'items recherché. Nous avons utilisé un jeu de données synthétique avec t_{len} (le nombre moyen d'items par ensemble d'items) égal à 15. Puis, nous avons recherché 5 ensembles d'items de tailles différentes avec les techniques (la taille de la clé bitmap hachée était 29). Remarquons que les ensembles d'items recherchés sont présents dans la base de données, de telle sorte que la méthode à base de clés bitmap hachées ne soit pas automatiquement désavantagée. En effet, cette dernière nécessite une analyse systématique de toutes les clés stockées, alors que les deux autres s'arrêteraient très rapidement si aucun des ensembles recherchés n'apparaissait dans les données. La taille de la table encodant l'arbre des ensembles d'items est de 3.5 Mb et celle de la table encodant les clés bitmap hachées est d'environ 1Mb. Ce surcoût reste cependant loin de l'explosion combinatoire qui peut se produire dans le pire des cas lors de la construction d'un arbre complet. La Figure 5.10 montre les résultats obtenus.

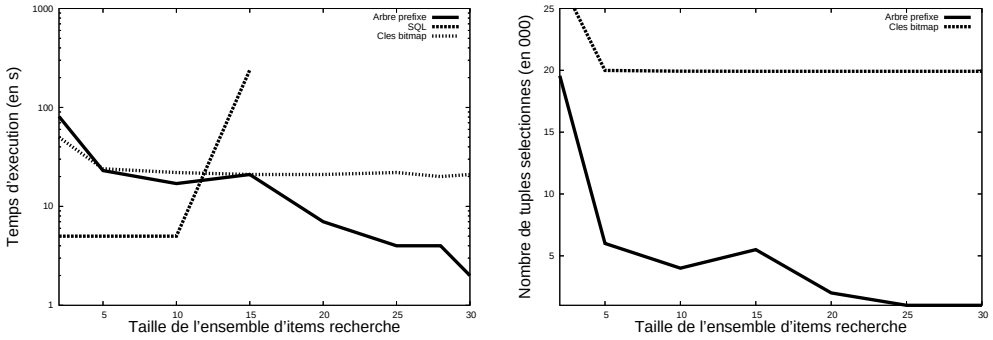

FIG. 5.10 – Expériences avec des jeux de données synthétiques

Une première remarque que l'on peut faire est que SQL est assez rapide lors de la recherche d'ensembles d'items de petite taille, notamment à cause des optimisations des serveurs SQL qui poussent les contraintes sur la présence de certains items, supprimant ainsi de nombreuses jointures inutiles. La technique des clés bitmap hachées est efficace pour les petits ensembles d'items, parce que dans ce cas, les clés bitmap sont suffisamment discrimantes pour filtrer efficacement les faux positifs. Cependant, avec des ensembles recherchés plus grands, différents items risquent plus facilement de correspondre au même bit, ce qui réduit sensiblement l'effica-

[4] `http ://www.mysql.com`
[5] `http ://www.almaden.ibm.com/cs/quest/index.html`

cité de la méthode. Notre méthode à base d'arbres d'ensembles d'items apparaît comme étant très efficace. Les temps d'exécution sont généralement meilleurs que ceux obtenus avec les deux autres techniques et celui-ci continue de baisser régulièrement au fur et à mesure que la taille des ensembles d'items recherchés s'accroît. Au contraire, les temps d'exécution de la méthode à base de clés bitmap hachées restent globalement stables après un certain seuil, parce que cette méthode implique une borne inférieure sur le nombre de scans à réaliser. Ainsi, si nous stockons N ensembles d'items, nous aurons toujours besoin d'analyser au moins N clés bitmap. Avec la méthode à base d'arbre d'ensembles d'items, si l'arbre est encodé en utilisant M tuples, nous savons que nous analyserons au plus M tuples ($M \geq N$) dans le pire des cas, et dans le meilleur des cas 1 tuple (la racine). Le temps d'exécution se réduit aussi car plus grand est l'ensemble recherché, moins nombreux sont les ensembles d'items dans lequel il peut être inclus. La Figure 5.10 montre aussi le nombre de tuples sélectionnés par chaque méthode et cela confirme l'intuition théorique : la quantité de tuples analysés par notre méthode diminue lorque la taille de l'ensemble d'items recherché augmente, alors que la méthode à base clés bitmap hachées atteint un seuil en dessous duquel il n'est plus possible de descendre.

Dans une deuxième expérience, nous avons analysé comment les temps d'exécution évoluaient par rapport à la taille moyenne des ensembles d'items stockés. Nous avons généré différents jeux de données avec les mêmes paramètres que dans la première expérience, sauf pour t_{len} dont la valeur variait entre 5 et 30. La taille de la clé bitmap hachée était de 97. Nous avons cherché un ensemble de 90 ensemble d'items de taille 12. Les résultats sont donnés sur la Figure 5.11 (la courbe correspondant à la technique SQL n'est pas tracée car les temps d'exécution étaient trop élevés). Elle montre que notre méthode passe à l'échelle lorsque la taille des ensembles d'items dans lesquels la recherche est effectuée s'accroît, ce qui est un point très important si l'on s'intéresse à l'applicabilité d'une technique à des processus d'ECD réels, où il est difficile de connaître *a priori* la taille des ensembles d'items. Concernant la taille de la table encodant l'arbre des ensembles d'items, les valeurs s'étalaient de 2,23 Mb (pour une taille moyenne de l'ensemble d'items de 2) à 3,4 Mb (pour une taille moyenne de l'ensemble d'items de 30) avec une augmentation régulière pour les valeurs intermédiaires. Pour une table encodant les clés bitmap hachées, la taille variait entre 1,5 Mb et 2,3 Mb. Cela montre que notre méthode passe aussi à l'échelle par rapport à la taille moyenne des ensembles d'items stockés, ce qui est important lorsqu'on considère des cas pratiques. Par exemple, dans le domaine de la bioinformatique, différentes techniques de discrétisation appliquées sur des données d'expression de gènes peuvent conduire à des tailles complètement différentes pour les collections d'ensembles d'items fréquents découverts [PLBB04]. Le coût de la méthode à base de clés bitmap hachées devient prohibitif lorsqu'on travaille avec des ensembles d'items de grandes tailles, parce que la taille de la clé bitmap est fixée alors que le nombre d'items à encoder augmente. Ainsi, dans ces cas, deux items différents ont plus de chance de correspondre au même bit de la clé bitmap, ce qui conduit à un plus grand nombre de vérifications inutiles sur des ensembles réels.

Ensuite, nous avons réalisé une expérience avec le jeu de données Census disponible sur le site Web de l'UCI [6]. Pour montrer la pertinence de notre approche dans un processus d'ECD, nous avons d'abord extrait tous les ensembles d'items fréquents (avec un seuil de fréquence minimale de 30 %) et nous avons obtenu environ 17 000 ensembles d'items d'une taille maximale de 13. Nous avons choisi une taille de clé bitmap de 57 (un dixième du nombre possible d'items). Nous les avons ensuite stockés dans une base de données avec les deux méthodes décrites dans les

[6] http ://www.ics.uci.edu/ mlearn/MLSummary.html

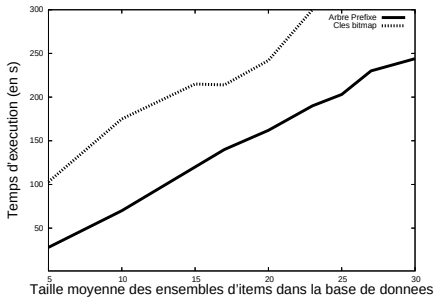

FIG. 5.11 – Temps d'exécution vs. taille moyenne des ensembles d'items

Sections 5.2.2 et 5.2.3. La taille de la table encodant l'arbre des ensembles d'items était de 1,9 Mb et celle de la table encodant les clés bitmap de 1,3 Mb. Nous avons cherché 10 ensembles d'items de tailles variables, tous étaient présents dans le jeu de données. Les résultats sont sur la Figure 5.12. Nous pouvons remarquer que les performances de la méthode à base de clés bitmap hachées sont moins bonnes que celles de la requête SQL classique. Comme il a déjà été dit pour les données synthétiques, la méthode SQL basique est efficace lorsque la taille de l'ensemble recherché est assez faible. Néanmoins, quand la taille de l'ensemble d'items recherché augmente, la technique utilisant l'arbre des ensembles d'items redevient la plus efficace.

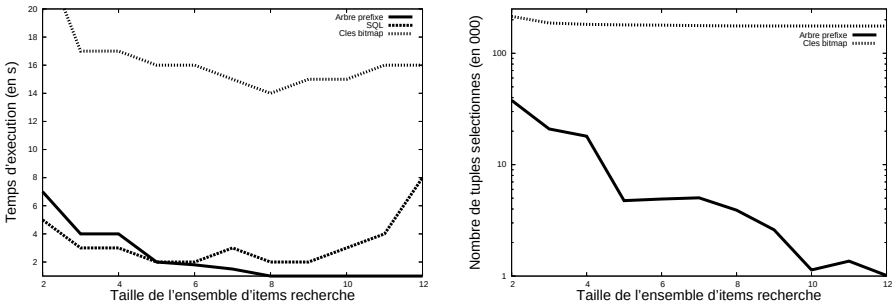

FIG. 5.12 – Résultats avec le jeu de données Census

Enfin, nous avons réalisé les mêmes expériences sur un jeu de données de type Microarray discrétisé. Nous avons stocké dans une base de données relationnelles tous les ensembles clos de ce jeu de données (10 098 ensembles d'items). La taille de la table encodant les clés bitmap hachées était de 486 Kb et celle de la table encodant l'arbre des ensembles d'items était de 696 Kb. Il y avait un nombre total d'items de 162 et la taille moyenne des ensembles d'items était de 5,25. Là encore, nous avons appliqué les 3 techniques pour trouver 5 ensembles d'items de tailles différentes. La taille de la clé bitmap hachée était de 29 bits afin de respecter les proportions données dans [MZ98]. Nous avons observé les temps d'exécution et le nombre de tuples sélectionnés (cf. Figure 5.13). Cela confirme les résultats précédemment obtenus. La méthode à base de clés bitmap hachées atteint un seuil en dessous duquel il n'est pas possible de descendre, alors que la méthode à base d'arbre des ensemble d'items devient plus rapide quand la taille de l'ensemble recherché augmente.

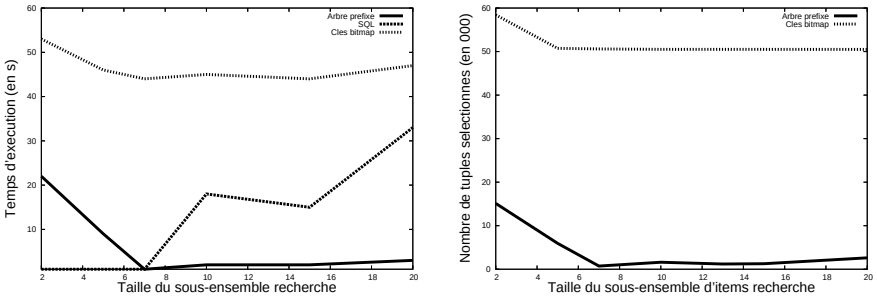

FIG. 5.13 – Expériences sur des jeux de données d'expression de gènes réels

5.5 Travaux analogues récents

Dans [SSMM04], les auteurs proposent une nouvelle technique pour les tests d'inclusion entre des ensembles d'items. Au lieu d'utiliser des clés bitmap, ils suggèrent d'affecter un nombre premier à chaque item et de calculer, pour chaque ensemble d'items X, un nombre $N(X)$ correspondant au produit des nombres premiers associés à chacun des items qui le compose. Lorsque l'on désire savoir si un ensemble d'items s est inclus dans un autre ensemble d'items t, il suffit de calculer $N(s)$ et $N(t)$ ainsi que la valeur de $N(t)$ modulo $N(s)$. Si cette dernière valeur est nulle, cela signifie que s est inclus dans t.

Exemple. Considérons un alphabet d'items `Items` $= \{A, B, C, D, E, F, G\}$, et associons à chaque item de `Items` un nombre premier de la manière suivante : $A : 2$, $B : 3$, $C : 5$, $D : 7$, $E : 11$, $F : 13$, $G : 17$. Supposons que nous disposions des ensembles d'items $\{A, C, D\}$, $\{A, D, F\}$ et $\{B, D, F, G\}\}$. Le calcul de N nous donne alors $N(\{A, C, D\}) = 70$, $N(\{A, D, F\}) = 182$ et $N(\{B, D, F, G\}) = 4641$. Nous voulons trouver les sur-ensembles de $\{D, F\}$. Pour cela, nous calculons $N(\{D, F\}) = 91$. Il suffit alors de calculer les modulos :

- $N(\{A, C, D\})$ *mod* $N(\{D, F\}) = 70$ donc $\{D, F\} \not\subseteq \{A, C, D\}$
- $N(\{A, D, F\})$ *mod* $N(\{D, F\}) = 0$ donc $\{D, F\} \subseteq \{A, D, F\}$
- $N(\{B, D, F, G\})$ *mod* $N(\{D, F\}) = 0$ donc $\{D, F\} \subseteq \{B, D, F, G\}$

La méthode de tests d'inclusion à base de nombres premiers présente néanmoins quelques inconvénients :

- Le cadre général de l'ECD recommande de considérer en entrée des bases de données pouvant être d'une taille très importante. Cela signifie entre autres que le nombre d'items utilisés peut être grand. C'est le cas par exemple lorsque l'on s'intéresse à l'analyse des valeurs d'expression de plusieurs milliers de gènes. Or, les nombres premiers ont tendance à rapidement atteindre de grandes valeurs. Les multiplier entre eux conduit à l'obtention d'une signature d'une taille très importante, parfois plus grande que l'ensemble d'items que l'on cherche à représenter. En ce sens, [SSMM04] s'inscrit dans la logique inverse de celle de [MZ98] où le but était explicitement obtenir une signature de taille inférieure à l'ensemble d'items à représenter.
- Il est indispensable de disposer d'outils capables de gérer des très grands nombres, ce qui n'est pas forcément le cas si on se place dans le cas d'une implémentation par dessus un SGBD existant qui n'est pas forcément optimisé pour ce genre de calcul. De plus, il n'est

pas certain que le calcul du modulo, qui repose sur le calcul d'une division entière et dont la rapidité est liée à la nature du diviseur, soit plus rapide que le test de type XOR bit à bit utilisé dans le cas des clés bitmap hachées et qui correspond à une opération atomique du processeur.
- Enfin, tout comme pour la méthode à base de clé bitmap, une analyse systématique de tous les ensembles d'items stockés lors de la recherche des sur-ensembles d'un ensemble d'items donné n'est pas envisageable. Le besoin d'une méthode d'indexation des ensembles d'items demeure et cela est d'autant plus sensible dans le cas d'une recherche négative, où il n'existe pas de sur-ensembles à l'ensemble d'items étudié. Le phénomène de palier constaté avec les clés bitmap se reproduirait ici.

Ce dernier point n'est néanmoins pas limitant et pose au contraire la question du couplage de cette méthode à base de nombres premiers avec une technique d'indexation par arbre d'ensembles d'items. Dans notre cas, aussi bien pour la construction de l'arbre que pour la recherche de sur-ensembles, il est possible de remplacer le test d'inclusion à base de clés bitmap par ce test à base de nombres premiers. Néanmoins, cela ne dispense pas de devoir conserver la représentation originale complète de l'ensemble d'items, aussi bien pour l'insertion d'un nouveau noeud dans l'arbre (en particulier lorsque celui-ci n'est ni parent ni enfant d'un noeud déjà existant) que pour la restitution des résultats lors de la recherche des sur-ensembles. En effet, concernant ce dernier point, retrouver la décomposition en facteurs premiers d'un nombre de très grande taille n'est pas forcément une tâche aisée.

5.6 Discussion et synthèse

Dans ce chapitre, nous avons présenté un premier scénario d'évaluation dans le cadre de la bioinformatique. Nous avons vu qu'avec les algorithmes dont nous disposions, nous pouvions mettre au point deux techniques d'exécution différentes pour ce scénario. Nous n'avons donc pas conçu de système de bases de données inductives qui auraient intégré ces différents outils dans une application commune. Cela est perceptible par le fait, que lors du déroulement de l'exécution de notre scénario, nous avons du avoir recours à différents scripts intermédiaires, pour deux grands types d'opérations :
- Les manipulations sur la représentation des données, comme la binarisation d'un contexte d'extraction (étape A), le croisement entre les données et les motifs (étape C), et la séparation d'un contexte en deux sous-contextes par sélection de transactions (étape D).
- Le post-traitement de motifs lors de l'utilisation d'extracteurs de motifs qui ne prenaient pas en compte toutes les contraintes présentes dans la requête inductive (étape B) où lors de la réutilisation de résultats déjà obtenus (étape E).

Lors de la conception d'un vrai système de gestion de bases de données inductives, mettre au point des outils permettant de réaliser automatiquement les manipulations sur la représentation des données ne devrait pas poser problème. En revanche, automatiser les post-traitements de motifs peut parfois être plus délicat, car celà suppose de disposer d'un véritable moteur de compilation des requêtes inductives capable de savoir comment traiter dans une requête une contrainte qui ne sera pas exploitée de manière active pendant l'extraction. Il faudra donc construire un filtre à partir de cette contrainte en sachant que l'on dispose d'un sur-ensemble de ce qu'on recherche. Prenons l'exemple de la requête inductive suivante :
create pattern set P **as** $s \in P \wedge\ s \in \mathcal{S}(\texttt{Items}) \wedge\ C_{MinFreq(I,t)}(s) \wedge\ C_{ER(R)}(s)$

Nous savons que l'algorithme SPIRIT permettent de gérer ce type de conjonction de contraintes, moyennant une relaxation de la contrainte sous forme d'expression rationnelle. Supposons que le système choisisse la relaxation V, alors il résoudra d'abord la requête inductive suivante :
create pattern set P' **as** $s \in P \wedge\ s \in \mathcal{S}(\texttt{Items}) \wedge\ C_{MinFreq(I,t)}(s) \wedge\ C_{Valid(\mathcal{R})}(s)$
puis il filtrera les résultats afin d'obtenir l'ensemble P recherché :
create pattern set P **as** $s \in P \wedge\ s \in P' \wedge\ C_{ER(\mathcal{R})}(s)$
Il apparait donc qu'un système de gestion de bases de données inductives devrait inclure des scripts permettant de réaliser des opérations de filtrage sur les ensembles de motifs extraits.

Pour finir, la dernière étape du scénario d'évaluation en bioinformatique que nous avons proposé faisait appel à des recherches ensemblistes dans des bases d'ensembles d'items stockés sous forme relationnelle. Nous ne disposions d'aucune méthode efficace pour résoudre ce problème, c'est pourquoi nous avons proposé une nouvelle méthode d'indexation basée sur des arbres préfixes pour optimiser ces problèmes de recherches de sur-ensembles. Nous avons montré l'efficacité de notre approche par rapport à d'autres techniques sur différents jeux de données aussi bien synthétiques que réels. Ce chapitre aura donc permis de montrer la pertinence de l'approche "scénarios d'évaluation" sur un exemple déroulé de bout en bout dans le domaine de la bioinformatique.

Chapitre 6

Conclusions

6.1 Synthèse

Ce travail de thèse s'est déroulé dans le cadre du projet européen cInQ (IST-FET 2000-26469) qui avait pour but le développement du cadre des bases de données inductives et qui s'est déroulé du 1er mai 2001 au 1er mai 2004. Mes travaux s'intègrent pleinement dans l'agenda proposé par ce projet. Dans les grandes lignes, ce dernier se composait des étapes suivantes. Dans un premier temps, l'analyse critique des langages de requêtes déjà existants pour la fouille de données, puis l'utilisation d'une approche par domaine de motifs pour la présentation des fonctions d'évaluation, des contraintes primitives et des algorithmes d'extraction, l'implantation des extracteurs et enfin l'application des algorithmes sur des données issues de différents domaines. De manière orthogonale, une importante partie du projet était également dédiée à la mise au point du cadre théorique des BDI, à la formalisation des processus d'extraction de connaissances sous forme de scénarios, ainsi qu'à l'analyse de l'apport des bases de données inductives dans différents domaines d'applications.

Les contributions de mon travail concernent donc les différents points de cet agenda. Dans un premier temps, j'ai réalisé un état de l'art sur les langages de requêtes pour la fouille de données et plus particulièrement pour l'extraction de règles d'association comme MINE RULE [MPC98], MSQL [IV99] et DMQL [HFW+96]. Une analyse de l'apport de l'API OLE DB for DM [NCFB01] a également été proposée. Cela a permis de mettre en évidence quelques contributions intéressantes comme les primitives de croisement entre les données et motifs, mais surtout de pointer le manque de support des processus d'extraction de données dans leur globalité. C'est à partir de ce constat et après une présentation du cadre formel des bases de données inductives que la notion de scénario d'ECD a été proposée. Le but d'un scénario d'ECD est de formaliser sous la forme d'une suite de requêtes inductives de l'ensemble des opérations qui constituent un processus d'ECD. Nous avons alors proposé deux déclinaisons de ce concept :

- les scénarios prototypiques : ils sont essentiellement tournés vers le transfert d'expertise entre utilisateurs et spécialistes de l'ECD. Ils permettent de formaliser le problème sur une base théorique solide afin de décrire les opérations à effectuer. Il faut cependant noter qu'un scénario prototypique n'est pas un log des actions de l'utilisateur. En effet, les opérations non pertinentes (comme la modification d'un paramètre pour l'exécution d'un algorithme afin d'obtenir un ensemble de résultats d'une taille suffisante) ne sont pas conservées. Le but est donc de décrire de manière abstraite et non ambigüe ce que l'utilisateur fait ou est

susceptible de faire.
- les scénarios d'évalution : ils sont essentiellement destinés au benchmarking des solutions d'ECD existantes. Le but ici est de décrire un problème d'ECD pertinent du point de vue par exemple de ses difficultés algorithmiques, où des stratégies d'exécution qu'il nécessite. L'idée n'est pas seulement de comparer les schémas d'exécution proposés par les différents outils sur des aspects quantitatifs (usage mémoire, temps d'exécution), mais également sur des aspects qualitatifs (stratégies mises en œuvre, contraintes exploitées dans les extractions, contraintes post-traitées, etc).

Outre le transfert d'expertise, la formalisation mise en place par les scénarios prototypiques permet de repérer et mettre en évidence des contraintes algorithmiques intéressantes à étudier pour l'extraction de certains types de motifs donnés et pour lesquelles nous ne disposons pas de techniques adaptées. C'est ainsi que dans le Chapitre 3, nous avons présenté des exemples de scénarios prototypiques faisant intervenir des motifs de nature séquentielle (motifs séquentiels d'items et motifs séquentiels logiques). Pour ces cas-là, nous avons été amené à nous intéresser à des conjonctions de contraintes particulières faisant intervenir à la fois une contrainte de fréquence minimale et une contrainte syntaxique. Cette contrainte syntaxique pouvant être une contrainte de similarité (dans le cas de Galibot) ou une contrainte sur la forme des motifs à découvrir (dans le cas de SPIRIT-LºG). Cependant, un problème s'est rapidement posé : ces contraintes ne présentaient en effet aucun caractère anti-monotone pour en faire une exploitation active dans un algorithme d'extraction par niveaux. Il a donc fallu mettre en place une stratégie de relaxation de contraintes pour lesquels on disposait de stratégies de génération et d'élagage de candidats efficaces. Puis nous avons montré l'intérêt de ces techniques sur des jeux de données synthétiques.

Enfin, nous avons présenté un premier scénario d'évaluation dans le cadre de la bioinformatique. Plus précisément, nous avons choisi un jeu de données d'expression de gènes réel et nous avons décrit un ensemble de manipulations qu'un utilisateur pouvait réaliser sur celui-ci. Ce scénario mettait en évidence des problèmes algorithmiques liés à l'extraction de motifs séquentiels, à l'extraction de représentations condensées d'ensembles fréquents et à la recherche ensemblistes parmi des ensembles d'items stockés sous forme relationnelle. Pour ce dernier point, comme nous ne disposions pas d'outils satisfaisants, nous avons proposé une nouvelle méthode d'indexation basée sur l'utilisation d'un arbre préfixe et de clés bitmap décrivant le contenu des ensembles d'items. Là encore, nous avons montré l'intérêt de cette approche en la comparant à d'autres techniques sur des jeux de données synthétiques et réels.

En résumé, on s'aperçoit donc que la notion de scénario d'ECD apparaît comme un fil directeur de mon travail de thèse. À partir de la formalisation des requêtes inductives proposée dans le Chapitre 3 , il a été possible d'écrire deux types différents de scénarios d'ECD. D'une part, les scénarios prototypiques nous ont permis de mettre en évidence des problèmes d'extraction sous contraintes intéressants qui ont été analysés au Chapitre 4. D'autre part, nous avons montré l'intérêt des scénarios d'évaluation sur un exemple tiré du domaine de la bioinformatique. Pour ce dernier, nous avons montré qu'il était possible de mettre en place différentes stratégies pour le résoudre et nous avons proposé une nouvelle méthode originale pour l'indexation des ensemble d'items dans les bases de données relationnelles afin d'optimiser le traitement de la dernière étape de ce scénario d'évaluation.

6.2 Perspectives

Compte-tenu des différents problèmes traités dans ce mémoire, les perspectives de travail ultérieures sont nombreuses. Je les classerai donc en deux catégories : les perspectives méthodologiques sur le cadre des bases de données inductives et les perspectives algorithmiques s'inscrivant dans la lignée du travail présenté au Chapitre 4.

6.2.1 Sur les aspects méthodologiques

Sur les langages

Dans le Chapitre 2, nous avons présenté différents langages pour l'extraction de règles d'associations. Différentes propositions concrètes ont été faites, mais elles ont pour particularité d'être toutes construites autour de certains algorithmes *ad hoc*. Cette méthode, si elle a le mérite d'être pragmatique afin de fournir aux utilisateurs un système opérationnel laisse apparaître certaines failles. La première est son extensibilité en termes d'opérations de post-traitement et de pré-traitement, car on est limité aux primitives fournies par le langage et qui concernent essentiellement la phase d'extraction (et encore plus précisément les contraintes et caractéristiques des algorithmes sous-jacents au système). Sur ce point, un rapprochement avec le travail effectué sur la phase de prétraitement dans le cadre du projet Mining Mart semble nécessaire. La deuxième est que la sémantique des primitives n'est pas toujours clairement définie. Ainsi, la définition du support d'une règle d'association peut ainsi varier d'un système à l'autre.

C'est en partie pour répondre à ce deuxième problème que des propositions de langages plus abstraits ont été réalisées ([cc04a] présentée et étendue dans le Chapitre 3). Mais là encore, un travail reste à faire au niveau des primitives et de la spécification de nouvelles contraintes par l'utilisateur à l'aide de ce langage abstrait. En effet, dans le cadre du projet cInQ, nous nous sommes délibérément limité à l'analyse de motifs locaux, comme les ensembles d'items ou les motifs séquentiels, qui correspondaient aux compétences principales des équipes y ayant pris part. Néanmoins, si l'on souhaite assurer la généricité de notre approche, il est souhaitable d'étendre notre langage abstrait à d'autres types de motifs, comme par exemple les arbres de décisions ou les clusters. Concernant le clustering, nous aurions dans ce cas un domaine de motifs assez proche de celui des concepts. Étant donnés un alphabet de symboles `Items` et un ensemble de transactions $\mathcal{T}$, nous notons $\mathcal{B}(\texttt{Items}, \mathcal{T})$ l'ensemble des clusters que l'on peut construire sur `Items` et $\mathcal{T}$. Les paramètres des algorithmes de clustering pourraient alors assez naturellement devenir des contraintes de notre langage. On pourrait ainsi disposer d'une contrainte de maximisation de la variance intra-classe, d'une contrainte de minimisation de la variance inter-classe, ainsi que de contraintes sur la taille des clusters ou leur composition (inclusion ou exclusion de certains exemple). Une requête inductive pour l'extraction de 2 clusters contenant au moins deux éléments et avec des contraintes sur les 2 variances pourrait s'écrire :
create pattern set P_1 **as** $C_1 \in \mathcal{B}(\texttt{Items}, \mathcal{T}) \wedge\ C_2 \in \mathcal{B}(\texttt{Items}, \mathcal{T}) \wedge\ C_1 \in P_1 \wedge\ C_2 \in P_1 \wedge\ C_{MaxIntra(10)}(C_1) \wedge C_{MaxIntra(10)}(C_2) \wedge C_{MinInter(1000)}(C_1, C_2) \wedge C_{MinSize(2)}(C_1) \wedge C_{MinSize(2)}(C_2)$

Pour les arbres de décision, on pourrait de la même manière noter $\mathcal{A}(\texttt{Items})$ l'ensemble des arbres que l'on peut construire sur l'alphabet de symboles `Items`. Des contraintes typiques sur les arbres pourraient consister en une limitation de la profondeur de l'arbre, une maximisation de l'erreur de classification à chaque noeud, etc.

Concernant les langages, les deux voies de développement présentées précédemment (solutions opérationnelles et langages abstraits) doivent être poursuivies si l'on souhaite mettre au

point des systèmes de gestion de bases de données inductives. La mise au point de langages pour l'extraction de motifs (et pas seulement des règles d'association) permettra de convaincre les utilisateurs de la pertinence de l'approche "Bases de Données Inductives". D'autre part, le développement de langages abstraits permettra de mettre au point des outils d'optimisation et de compilation des requêtes, à la manière dont l'algèbre relationnelle est la base de SQL.

Sur les scénarios

Dans ce mémoire, nous avons présenté un nouveau concept qui découle de l'utilisation de langages abstraits pour l'extraction de connaissances dans les données : le concept de scénarios. Nous en avons distingué deux types : les scénarios prototypiques et les scénarios d'évaluation. Un scénario prototypique permet de décrire sous forme d'une séquence de requêtes inductives un ensemble d'opérations qu'un utilisateur peut réaliser. Cela est notamment utile pour le transfert de savoir-faire entre utilisateurs et experts ou entre experts eux-mêmes. Cette notion de scénario prototypique doit être développée à l'avenir. Pour cela, un travail de diffusion de cette notion auprès des utilisateurs semble nécessaire. Il pourrait ainsi être utile de représenter plusieurs processus d'ECD réels connus sous la forme de scénarios prototypiques pour en montrer la pertinence. De plus, dans cette thèse, nous nous sommes plus particulièrement intéressés à l'extraction de motifs sous contrainte de fréquence minimale, une contrainte classique en ECD. Néanmoins, même dans le cas des ensembles d'items, celle-ci n'est pas toujours pertinente pour certains problèmes et des travaux commencent à s'intéresser à l'extraction sous des contraintes d'utilité autres que la fréquence minimale [CYS03]. De plus, si l'on considère des types de motifs comme les arbres de décision, la notion de fréquence n'a plus vraiment de sens. Il pourrait donc être intéressant de décrire d'autres scénarios prototypiques qui mettent l'accent sur d'autres motifs et contraintes. Enfin, il serait intéressant d'étudier comment apporter une dimension temporelle à la description des processus contenue dans les scénarios, un peu à la manière des diagramme de séquences en UML qui permettent de représenter les interactions temporelles entre les objets d'une application.

Dans le Chapitre 5, nous avons présenté un premier scénario d'évaluation. De nombreuses pistes de travail restent à approfondir sur ce point. Il faut d'abord développer cette notion en cherchant à fournir et à mettre à la disposition de la communauté, des benchmarks basés sur des scénarios d'évaluation. Pour chacun d'entre eux, on pourrait proposer un jeu de données plus un scénario correspondant au problème que l'on cherche à résoudre. Des méthodes pour la comparaison qualitative des plans d'exécution des scénarios restent encore à mettre en place. De plus, le fait de pouvoir exécuter des séquences de requêtes pose le problème de disposer d'outils capables capables non seulement d'exécuter les requêtes de manière consécutive mais aussi de tirer partie des informations présentes dans différentes requêtes inductives pour optimiser le traitement global du scénario. Certains travaux ont commencé à s'intéresser aux problèmes du traitement et de l'optimisation de séquences de requêtes inductives [Meo03, DGLS02, JB02a]. Ce problème conduit *in fine* a la question de l'élaboration de véritables moteurs de compilation pour les bases de données inductives qui seraient capables à partir des propriétés des requêtes et des contraintes de déterminer automatiquement une technique d'exécution. Sur ce point, une piste de recherche intéressante est l'analyse des analogies avec le domaine de la collaboration de solveurs. En effet, le langage que nous avons utilisé pour décrire les scénarios est un langage à base de contraintes et il est courant que dans des processus d'ECD les extractions portent sur différents types de motifs et fassent intervenir plusieurs contraintes. C'est pourquoi, afin

de résoudre le problème de la "compilation" efficace des scénarios, il pourrait être pertinent d'analyser les méthodes de collaboration de solveurs dans le domaine de la programmation par contraintes, où des résultats efficaces ont déjà été obtenus.

6.2.2 Sur les aspects algorithmiques

- L'algorithme Galibot a permis de montrer la possibilité de prendre en compte une contrainte de similarité par rapport à un motif de référence donné par l'utilisateur en plus de la contrainte de fréquence minimale. Nous avons vu qu'il était possible de relaxer la contrainte de similarité, qui n'était ni-monotone, ni anti-monotone, en une contrainte certes moins sélective, mais pour laquelle nous disposions d'algorithme capables d'en faire une exploitation active. Un axe de recherche prometteur pour la suite de Galibot est l'analyse de nouvelles mesures de similarité, comme par exemple l'algorithme Viterbi ou celle du calcul de l'Espérance Maximale utilisée notamment pour les modèles de Markov [BB98, Kro98].
- Concernant l'algorithme SPIRIT-L$^{\circ}$G pour l'extraction de motifs séquentiels logiques, nous pouvons envisager différentes améliorations et extensions. La première consiste à extraire des motifs séquentiels logiques constitués d'atomes clos (i.e. construits sur des variables et des constantes) plutôt que d'atomes libres construits uniquement sur des variables. Cela permettrait de réduire l'espace de recherche lors de la génération de candidats en limitant le nombre de renommages à considérer. Une autre idée intéressante serait de prendre en compte dans l'extraction des biais de langage similaires à ceux utilisés dans `WarMR` [DT99] pour l'extraction de requêtes Datalog fréquentes. Là encore, cela nous permettra de réduire le nombre de candidats générés en guidant les renommages de variables.
- Enfin, en ce qui concerne la technique d'indexation des ensembles d'items stockés dans les bases de données relationnelles présentée au Chapitre 5, différentes pistes peuvent être explorées. La première consiste à implémenter notre méthode à l'intérieur d'un SGBD plutôt que d'utiliser une implémentation "on-top" comme nous le faisons actuellement. Cela permettrait de réduire encore sensiblement les temps d'exécution avec les méthodes à base de clés bitmap. Ensuite, il faudrait exploiter d'autres types d'implémentation pour la représentation des ensembles d'items. Nous avons proposé une méthode à base d'arbres préfixes, mais d'autres structure de données pourraient être imaginées. Pour cela, il semble nécessaire de voir les points de similitude entre notre approche guidée par la problématique ECD et les travaux déjà réalisés au sein de la communauté "Base de Données" sur les recherches ensemblistes, même si ces travaux ne sont pas étiquetés "Fouille de Données". Enfin, nous avons vu que selon la taille de l'ensemble d'items recherché ou la taille de l'ensemble de résultats à fouiller, les performances peuvent être meilleures avec des techniques simples comme la recherche SQL brute ou les clés bitmap classiques. Il faut donc mettre en place un système qui permettent de choisir dynamiquement la meilleure technique en fonction de ces différents paramètres. Là encore, un tel composant aurait pleinement sa place dans un compilateur de requêtes inductives.

Bibliographie

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the ACM International Conference on Management of Data (SIGMOD'93)*, pages 207–216, Washington, D.C., June 1993. ACM Press.

[ALB03] Hunor Albert-Lorincz and Jean-François Boulicaut. Mining frequent sequential patterns under regular expressions : a highly adaptative strategy for pushing constraints. In Daniel Barbará and Chandrika Kamath, editors, *Proceedings of the 3rd SIAM International Conference on Data Mining (SDM'03)*, pages 316–320, San Francisco, USA, May 2003. SIAM.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)*, pages 487–499, Santiago de Chile, Chile, September 1994. Morgan Kaufmann.

[AS95] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In Philip S. Yu and Arbee L. P. Chen, editors, *Proceedings of the 11th International Conference on Data Engineering, (ICDE'95)*, pages 3–14, Taipei, Taiwan, March 1995. IEEE Press.

[AS96] Rakesh Agrawal and Kyuseok Shim. Developing tightly-coupled data mining applications on a relational database system. In Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad, editors, *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pages 287–290, Portland, Oregon, August 1996. AAAI Press.

[BB98] Pierre Baldi and Soren Brunak. *Bioinformatics : the Machine Learning Approach.* Cambridge : MIT Press, 1998. 351 pages.

[BBJ+02] Céline Becquet, Sylvain Blachon, Baptiste Jeudy, Jean-François Boulicaut, and Olivier Gandrillon. Strong association rule mining for large gene expression data analysis : a case study on human SAGE data. *Genome Biology*, 3(12) :1–16, 2002.

[BBMM04] Marco Botta, Jean-François Boulicaut, Cyrille Masson, and Rosa Meo. Query languages supporting descriptive rule mining : a comparative study. In Meo et al. [MLK04], pages 27–54.

[BBR03] Jean-François Boulicaut, Artur Bykowski, and Christophe Rigotti. Free-sets : a condensed representation of boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery journal*, 7(1) :5–22, 2003.

[BBR04] Jérémy Besson, Jean-François Boulicaut, and Céline Robardet. Constraint-based mining of formal concepts in transactional data. In *Proceedings of the Eighth Pacific-Asia Conference on Knowledge Discov ery and Data Mining (PaKDD'04)*, volume 3056 of *Lecture Notes in Computer Science*, pages 615–624, Sydney, Australia, May 2004. Springer-Verlag.

[BD97] Véronique Benzaken and Anne Doucet. *Bases de Données Orientées Objet*. Paris : Dunod, 1997.

[BGKW03] Cristian Bucila, Johannes Gehrke, Daniel Kifer, and Walker M. White. Dualminer : A dual-pruning algorithm for itemsets with constraints. *Data Mining and Knowledge Discovery*, 7(3) :241–272, 2003.

[BGMP03a] Francesco Bonchi, Fosca Giannotti, Alessio Mazzanti, and Dino Pedreschi. Examiner : Optimized level-wise frequent pattern mining with monotone constraint. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM'03)*, pages 11–18, Melbourne, Florida, USA, December 2003. IEEE Computer Society.

[BGMP03b] Francesco Bonchi, Fosca Giannotti, Alessio Mazzanti, and Dino Pedreschi. Exante : Anticipated data reduction in constrained pattern mining. In Nada Lavrac, Dragan Gamberger, Hendrik Blockeel, and Ljupco Todorovski, editors, *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03)*, volume 2838 of *Lecture Notes in Computer Science*, pages 59–70, Cavtat-Dubrovnik, Croatia, September 2003. Springer-Verlag.

[BJ01] Jean-François Boulicaut and Baptiste Jeudy. Mining free-sets under constraints. In Michel E. Adiba, Christine Collet, and Bipin C. Desai, editors, *Proceedings of the International Database Engineering & Applications Symposium IDEAS'01*, pages 322 – 329, Grenoble, France, July 2001. IEEE Computer Society.

[BJ05] Jean-François Boulicaut and Baptiste Jeudy. Constraint-based data mining. In Oded Maimon and Lior Rokach, editors, *Data Mining and Knowledge Discovery Handbook : A Complete Guide for Practitioners and Researchers*. Kluwer Academic Publishers, 2005. To appear, 18 pages.

[BKM99] Jean-François Boulicaut, Mika Klemettinen, and Heikki Mannila. Modeling KDD processes within the inductive database framework. In Mukesh K. Mohania and A. Min Tjoa, editors, *Proceedings of the First International Conference of Data Warehousing and Knowledge Discovery (DaWaK'99)*, volume 1676 of *Lecture Notes in Computer Science*, pages 293 – 302, Florence, Italy, September 1999. Springer-Verlag.

[BMUT97] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In Joan Peckham, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'97)*, SIGMOD Record, pages 255–264, Tucson, Arizona, USA, May 1997. ACM Press.

[Bon03] Francesco Bonchi. *Frequent Pattern Queries : Language and Optimizations*. PhD thesis, Dipartimento di Informatica Università di Pisa, December 2003.

[Bou04] Jean-François Boulicaut. Inductive databases and multiple uses of frequent itemsets : The cInQ approach. In Meo et al. [MLK04], pages 1–23.

[BR98] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2) :285–297, June 1998.

[BR03] Artur Bykowski and Christophe Rigotti. DBC : A condensed representation of frequent patterns for efficient mining. *Information Systems*, 28(8) :949–977, 2003.

[BRBR05] Jérémy Besson, Céline Robardet, Jean-François Boulicaut, and Sophie Rome. Constraint-based concept mining and its application to microarray data analysis. *Intelligent Data Analysis journal*, 9(1) :59–82, 2005.

[CBM02] Matthieu Capelle, Jean-François Boulicaut, and Cyrille Masson. Extraction de motifs séquentiels sous contrainte de similarité. In *Actes des 2èmes Journées francophones d'Extraction et de Gestion des Connaissances (EGC'02)*, pages 65–76, Montpellier, France, January 2002. Hermes.

[cc02] cInQ consortium. D4a : Pattern Domain DD (data dependencies). Deliverable Project cInQ IST-2000-26469. Technical report, Coordination INSA Lyon, Bâtiment Blaise Pascal, F-69621 Villeurbanne, France, February 2002. Restricted access.

[cc04a] cInQ consortium. D1c : Theory for Inductive Databases. Deliverable Project cInQ IST-2000-26469. Technical report, Coordination INSA Lyon, Bâtiment Blaise Pascal, F-69621 Villeurbanne, France, May 2004. Public report.

[cc04b] cInQ consortium. D2c : Pattern Domain ITEM (itemsets, episodes and rules). Deliverable Project cInQ IST-2000-26469. Technical report, Coordination INSA Lyon, Bâtiment Blaise Pascal, F-69621 Villeurbanne, France, May 2004. Restricted access.

[cc04c] cInQ consortium. D3c & D7c : Solvers for the DLOG Pattern Domain. Deliverable Project cInQ IST-2000-26469. Technical report, Coordination INSA Lyon, Bâtiment Blaise Pascal, F-69621 Villeurbanne, France, May 2004. Restricted access.

[cc04d] cInQ consortium. DI25c : Pattern domain EQN. Deliverable Project cInQ IST-2000-26469. Technical report, Coordination INSA Lyon, Bâtiment Blaise Pascal, F-69621 Villeurbanne, France, May 2004. Restricted access.

[cc04e] cInQ consortium. U1c : XML patterns. Deliverable Project cInQ IST-2000-26469. Technical report, Coordination INSA Lyon, Bâtiment Blaise Pascal, F-69621 Villeurbanne, France, May 2004. Restricted access.

[CG02] Toon Calders and Bart Goethals. Mining all non-derivable frequent itemsets. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD'02)*, volume 2431 of *Lecture Notes in Computer Science*, pages 74–85, Helsinki, Finland, August 2002. Springer-Verlag.

[Cha98] Surajit Chaudhuri. Data mining and database systels : Where is the intersection ? *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 1(21) :4–8, 1998.

[CMB02] Matthieu Capelle, Cyrille Masson, and Jean-François Boulicaut. Mining frequent sequential patterns under a similarity constraint. In Hujun Yin, Nigel M. Allinson, Richard Freeman, John A. Keane, and Simon J. Hubbard, editors, *Proceedings of the 3rd International Conference on Intelligent Data Engineering and Automated*

Learning (IDEAL'02), volume 2412 of *Lecture Notes in Computer Science*, pages 1–6, Manchester, United Kingdom, August 2002. Springer-Verlag.

[CYS03] Raymond Chan, Qiang Yang, and Yi-Dong Shen. Mining high utility itemsets. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003)*, pages 19–26, Melbourne, Florida, USA, December 2003. IEEE Computer Society.

[DGLS02] Cheikh Talibouya Diop, Arnaud Giacometti, Dominique Laurent, and Nicolas Spyratos. Composition of mining contexts for efficient extraction of association rules. In Christian S. Jensen, Keith G. Jeffery, Jaroslav Pokorný, Simonas Saltenis, Elisa Bertino, Klemens Böhm, and Matthias Jarke, editors, *Proceedings of the 8th International Conference on Extending Database Technology (EDBT'02)*, volume 2287 of *Lecture Notes in Computer Science*, pages 106–123, Prague, Czech Republic, March 2002. Springer-Verlag.

[DJHS01] Heikki Mannila David J. Hand and Padhraic Smyth. *Principles of Data Mining*. Cambridge : MIT Press, 2001. 546 pages.

[dRK01] Luc de Raedt and Stefan Kramer. The levelwise version space algorithm and its application to molecular fragment finding. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 853 – 862, Seattle, Washington, USA, August 2001. Morgan Kaufmann.

[DT99] Luc Dehaspe and Hannu Toivonen. Discovery of frequent Datalog patterns. *Data Mining and Knowledge Discovery*, 3(1) :7–36, 1999.

[FR04] Johannes Fischer and Luc De Raedt. Towards optimizing conjunctive inductive queries. In Honghua Dai, Ramakrishnan Srikant, and Chengqi Zhang, editors, *Proceedings of the 8th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, (PAKDD'04)*, volume 3056 of *Lecture Notes in Computer Science*, pages 625–637, Sydney, Australia, May 2004. Springer-Verlag.

[GLD04] Arnaud Giacometti, Dominique Laurent, and Cheikh Talibouya Diop. Condensed representations for sets of mining queries. In Meo et al. [MLK04], pages 254–273.

[GRS99] Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. SPIRIT : Sequential pattern mining with regular expression constraints. In *Proceedings of 25th International Conference on Very Large Data Bases (VLDB'99)*, pages 223–234, Edinburgh, Scotland, UK, September 1999. Morgan Kaufmann.

[GRS02] Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Mining sequential patterns with regular expression constraints. *IEEE Transactions on Knowledge and Data Engineering*, 14(3) :530–552, 2002.

[Han02] David J. Hand. Pattern detection and discovery. In David J. Hand, Niall M. Adams, and Richard J. Bolton, editors, *Proceedings of the ESF Exploratory Workshop*, volume 2447 of *Lecture Notes in Computer Science*, pages 1–12, London, United Kingdom, September 2002. Springer-Verlag.

[Hay98] Simon Haykin. *Neural Networks : A Comprehensive Foundation*. Upper Saddle River : Pearson Education, 1998.

[HBK+03] Kimmo Hätönen, Jean-François Boulicaut, Mika Klemettinen, Markus Miettinen, and Cyrille Masson. Comprehensive log compression with frequent patterns. In Yahiko Kambayashi, Mukesh K. Mohania, and Wolfram Wöß, editors, *Proceedings of the 5th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'03)*, volume 2737 of *Lecture Notes in Computer Science*, pages 360–370, Praha, Czech Republic, September 2003. Springer-Verlag.

[HDR99] Alladin Hafez, Jitender S. Deogun, and Vijay V. Raghavan. The item-set tree : a data structure for data mining. In Mukesh K. Mohania and A. Min Tjoa, editors, *Proceedings of the 1st International Conference on Data Warehousing and Knowledge Discovery (DaWaK '99)*, volume 1676 of *Lecture Notes in Computer Science*, pages 183–192, Florence, Italy, August 1999. Springer-Verlag.

[HFW+96] Jiawei Han, Yongjian Fu, Wei Wang, Krzysztof Koperski, and Osmar Zaiane. DMQL : a data mining query language for relational databases. In Raymond Ng, editor, *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'96)*, Montreal, Canada, June 1996.

[HK00] Jiawei Han and Micheline Kamber. *Data Mining : Concepts and techniques*. Morgan Kaufmann Publishers, San Francisco, California, USA, 2000. 533 pages.

[HKMT95] Marcel Holsheimer, Martin L. Kersten, Heikki Mannila, and Hannu Toivonen. A perspective on databases and data mining. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, pages 150–155, Montreal, Canada, August 1995. AAAI Press.

[IM96] Tomasz Imielinski and Heikki Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11) :58–64, November 1996.

[IV99] Tomasz Imielinski and Aashu Virmani. MSQL : A query langugage for database mining. *Data Mining and Knowledge Discovery*, 3(4) :373–408, December 1999.

[IVA96] Tomasz Imielinski, Aashu Virmani, and Amin Abdulghani. Datamine : Application programming interface and query language for database mining. In Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 256–262, Portland, Oregon, USA, August 1996. AAAI Press.

[JB01] Nico Jacobs and Hendrik Blockeel. From shell logs to shell scripts. In Céline Rouveirol and Michèle Sebag, editors, *Proceedings of the 11th International Conference Inductive Logic Programming (ILP'01)*, volume 2157 of *Lecture Notes in Artificial Intelligence*, pages 80–90, Strasbourg, France, September 2001.

[JB02a] Baptiste Jeudy and Jean-François Boulicaut. Optimization of association rule mining queries. *Intelligent Data Analysis*, 6(4) :341–357, 2002.

[JB02b] Baptiste Jeudy and Jean-François Boulicaut. Using condensed representations for interactive association rule mining. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, *Proceedings of the 6th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'02)*, volume 2431 of *Lecture Notes in Artifical Intelligence*, pages 225–236, Helsinki, Finland, August 2002. Springer-Verlag.

[Jeu02] Baptiste Jeudy. *Extraction de motifs sous contraintes : application à l'évaluation de requêtes inductives*. PhD thesis, Institut National des Sciences Appliquées de Lyon, LIRIS, F-69621 Villeurbanne cedex, France, December 2002.

[Kro98] Anders Krogh. An introduction to Hidden Markov Models for biological sequences. In Steven L. Salzberg, David B. Searls, and Simon Kasif, editors, *Computational Methods in Molecular Biology*, pages 45–63, 1998.

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8) :707–710, 1966.

[LR05] Sau Dan Lee and Luc De Raedt. An efficient algorithm for mining string databases under constraints. In Bart Goethals and Arno Siebes, editors, *Proceedings of the Third International Workshop on Knowledge Discovery in Inductive Databases (KDID'04)*, volume 3377 of *Lecture Notes in Computer Science*, pages 108–129, Pisa, Italy, September 2005. Springer-Verlag.

[LRBE03] Marion Leleu, Christophe Rigotti, Jean-François Boulicaut, and Guillaume Euvrard. Constraint-based mining of sequential patterns over datasets with consecutive repetitions. In Nada Lavrac, Dragan Gamberger, Hendrik Blockeel, and Ljupco Todorovski, editors, *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03)*, volume 2838 of *Lecture Notes in Computer Science*, pages 291–302. Springer Verlag, September 2003.

[Man97] Heikki Mannila. Inductive databases and condensed representations for data mining. In Jan Maluszynski, editor, *Proceedings of the International Logic Programming Symposium (ILPS'97)*, pages 21 – 30, Port Jefferson, USA, October 1997. MIT Press.

[Meo03] Rosa Meo. Optimization of a language for data mining. In *Proceedings of the 18th ACM Symposium on Applied Computing (SAC'03)*, pages 437–444, Melbourne, Florida, USA, March 2003. ACM Press.

[Mit82] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2) :203–226, 1982.

[Mit97] Tom M. Mitchell. *Machine Learning*. Boston : McGraw Hill, 1997. 414 pages.

[MJ02a] Cyrille Masson and François Jacquenet. Découverte de séquences logiques fréquentes sous contraintes. In *Actes du 13ème Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificielle (RFIA'02)*, pages 673–684, Angers, France, January 2002.

[MJ02b] Cyrille Masson and François Jacquenet. Mining frequent logical sequences with SPIRIT-LoG. In Stan Matwin and Claude Sammut, editors, *Proceedings of the 12th International Conference on Inductive Logic Programming (ILP'02)*, volume 2583 of *Lecture Notes in Artificial Intelligence*, pages 166–182, Sydney, Australia, July 2002. Springer-Verlag.

[MLK04] Rosa Meo, Pier Luca Lanzi, and Mika Klemettinen, editors. *Database support for Data Mining Applications : Discovering Knowledge with Inductive Queries*, volume 2682 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[Moe00] Pirjo Moen. *Attribute, Event Sequence, and Event Type Simarity Notions for Data Minin g*. PhD thesis, Departement of Computer Science, University of Helsinki, Finland, February 2000.

[MPC96] Rosa Meo, Giuseppe Psaila, and Stefano Ceri. A new SQL-like operator for mining association rules. In *22nd International Conference on Very Large Data Bases (VLDB'96)*, pages 122–133, Bombay, India, September 1996.

[MPC98] Rosa Meo, Giuseppe Psaila, and Stefano Ceri. An extension to SQL for mining association rules. *Data Mining and Knowledge Discovery*, 2(2) :195–224, June 1998.

[MR94] Stephen Muggleton and Luc De Raedt. Inductive logic programming : theory and methods. *Journal of Logic Programming*, 19-20 :629–679, 1994.

[MRB04] Cyrille Masson, Céline Robardet, and Jean-François Boulicaut. Optimizing subset queries : a step towards SQL-based inductive databases for itemsets. In Hisham Haddad, Andrea Omicini, Roger L. Wainwright, and Lorie M. Liebrock, editors, *Proceedings of the 19th ACM Symposium on Applied Computing (SAC'04)*, pages 535–539, Nicosia, Cyprus, March 2004. ACM Press.

[MZ98] Tadeusz Morzy and Maciej Zakrzewicz. Group bitmap index : a structure for association rules retrieval. In Rakesh Agrawal, Paul E. Stolorz, and Gregory Piatetsky-Shapiro, editors, *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pages 284–288, New York City, New York, USA, August 1998. AAAI Press.

[NCBF00] Amir Netz, Surajit Chaudhuri, Jeff Bernhardt, and Usama Fayyad. Integration of data mining and relational databases. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *Proceedings of the 26th International Conference on Very Large Databases (VLDB'00)*, pages 719–722, Cairo, Egypt, September 2000. Morgan Kaufmann.

[NCFB01] Amir Netz, Surajit Chaudhuri, Usama M. Fayyad, and Jeff Bernhardt. Integrating data mining with SQL databases : OLE DB for data mining. In *Proceedings of the 17th International Conference on Data Engineering (ICDE'01)*, pages 379–387, Heidelberg, Germany, April 2001. IEEE Computer Society.

[NLHP98] Raymond T. Ng, Laks V. S. Lakshmanan, Jiawei Han, and Alex Pang. Exploratory mining and pruning optimizations of constrained associations rules. In Laura M. Haas and Ashutosh Tiwary, editors, *Proceedings of the ACM International Conference on Management of Data (SIGMOD'98)*, pages 13–24, Seattle, Washington, USA, June 1998. ACM Press.

[ole00] Microsoft Corporation. *OLE DB for Data Mining Specification*, July 2000. Available at http ://www.microsoft.com/data/oledb/dm.

[PHL01] Jian Pei, Jiawei Han, and Laks V. S. Lakshmanan. Mining frequent item sets with convertible constraints. In *Proceedings of the 17th International Conference on Data Engineering (ICDE'01)*, pages 433–442, Heidelberg, Germany, April 2001. IEEE Computer Society.

[PHMA+04] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Mining sequential patterns by pattern-growth : The prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11) :1424–1440, 2004.

[PLBB04] Ruggero G. Pensa, Claire Leschi, Jérémy Besson, and Jean-François Boulicaut. Assessment of discretization techniques for relevant pattern discovery from gene

expression data. In *Proceedings of the 4th ACM Workshop on Data Mining in Bio-Informatics (BIOKDD'04), co-located with ACM-SIGKDD'04*, pages 24–30, Seattle, Washington, August 2004. ACM Press.

[pmm] *Predictive Model Markup Language*. Available online at http ://www.dmg.org/, consulté le 01/06/2005.

[PSF91] Gregory Piatetsky-Shapiro and William J. Frawley. *Knowledge Discovery in Databases*. Menlo Park : AAAI Press, 1991. 525 pages.

[Qui86] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1) :81–106, 1986.

[Rae03] Luc De Raedt. A perspective on inductive databases. *SIGKDD Explorations : Newsletter of the Special Interest Group on Knowledge Discovery and Data Mining, ACM*, 4(2) :69–77, January 2003.

[Ran03] Ralf Rantzau. Processing frequent itemset discovery queries by division and set containment join operators. In Mohammed J. Zaki and Charu C. Aggarwal, editors, *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery (DMKD'03)*, pages 20–27, San Diego, California, June 2003. ACM Press.

[Ran04] Ralf Rantzau. Frequent itemset discovery with SQL using a vertical layout and universal quantification. In Meo et al. [MLK04], pages 198–217.

[RBCB03] François Rioult, Jean-François Boulicaut, Bruno Crémilleux, and Jérémy Besson. Using transposition for pattern discovery from microarray data. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 73–79, June 2003.

[SA95] Ramakrishnan Srikant and Rakesh Agrawal. Mining generalized association rules. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *Proceedings of 21th International Conference on Very Large Data Bases (VLDB'95)*, pages 407–419, Zurich, Switzerland, September 1995. Morgan Kaufmann.

[SK97] Arno Siebes and Martin L. Kersten. KESO : Minimizing database interaction. In David Heckerman, Heikki Mannila, and Daryl Pregibon, editors, *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (KDD'97)*, pages 247–250, Newport Beach, USA, August 1997. AAAI Press.

[SSMM04] S.N.Sivanandam, S.Sumathi, Ms.T.Hamsapriya, and Mr.K.Babu. Parallel buddy prima - a hybrid parallel frequent itemset mining algorithm for very large databases. *Academic Open Internet Journal*, 13, 2004. Available online at http ://www.acadjournal.com.

[STA98] Sunita Sarawagi, Shiby Thomas, and Rakesh Agrawal. Integrating association rule mining with relational database systems : Alternatives and implications. In Laura Haas and Ashutosh Tiwary, editors, *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD'98)*, pages 343–354. AAAI Press, June 1998.

[STA00] Sunita Sarawagi, Shiby Thomas, and Rakesh Agrawal. Integrating association rule mining with relational database systems : Alternatives and implications. *Data Mining and Knowledge Discovery*, 4(2–3) :89–125, 2000.

[TL02] Alexander Tuzhilin and Bing Liu. Querying multiple sets of discovered rules. In *Proceedings of the Eighth ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'02)*, pages 52–60, Edmonton, Alberta, Canada, July 2002. ACM Press.

[Vir98] Aashu Virmani. *Second Generation Data Mining*. PhD thesis, Rutgers University, New Jersey, USA, 1998.

[Wil82] Rudolf Wille. Restructuring lattice theory : An approach based on hierarchies of concepts. In Ivan Rival, editor, *Ordered sets*, pages 445–470, 1982.

[Zak98] Mohammed Javeed Zaki. Efficient enumeration of frequent sequences. In Georges Gardarin, James C. French, Niki Pissinou, Kia Makki, and Luc Bouganim, editors, *Proceedings of the 7th ACM International Conference on Information and Knowledge Management (CIKM'98)*, pages 68–75, Bethesda, Maryland, USA, November 1998. ACM Press.

[Zak00] Mohammed Javeed Zaki. Sequence mining in categorical domains : Incorporating constraints. In *Proceedings of the 9th Conference on Information and Knowledge Management (CIKM'00)*, pages 422–429, Washington DC, November 2000. ACM Press.

[Zak01] Mohammed Javeed Zaki. SPADE : an efficient algorithm for mining frequent sequences. *Machine Learning Journal, special issue on Unsupervised Learning*, 42(1/2) :31–60, 2001.