N° d'Ordre : D.U. 1461
EDSPIC : 288

Université Blaise Pascal - Clermont II

École doctorale
sciences pour l'ingénieur de Clermont-Ferrand

THÈSE

présentée par

Fabien De Marchi

Formation doctorale :
Informatique, productique, imagerie médicale

pour obtenir le grade de

Docteur d'université
Spécialité : Informatique

Découverte et visualisation par l'exemple des dépendances fonctionnelles et d'inclusion dans les bases de données relationnelles

Soutenue publiquement le 9 décembre 2003 devant le jury :

Mme Nicole Bidoit	Rapporteur
M. Dominique Laurent	Rapporteur
M. Jean-Marc Petit	Co-directeur de Thèse
M. Alain Quilliot	Président
M. Michel Schneider	Directeur de Thèse
M. Farouk Toumani	Examinateur
M. Djamel Abdelkader Zighed	Examinateur

Cette thèse est le résultat de la coopération de nombreuses personnes, collaborant dans l'intérêt de la recherche et des étudiants. Je remercie ainsi l'ensemble des travailleurs de l'université Blaise Pascal qui m'ont apporté leur aide, qu'elle soit administrative, pédagogique ou technique.

Cette thèse est aussi le résultat d'un travail d'équipe, c'est pourquoi je remercie Michel Schneider de m'avoir permis d'intégrer l'équipe base de données du LIMOS. Je remercie les membres (et anciens membres...) de cette équipe parmi lesquels j'ai trouvé beaucoup d'aide et d'amitié, et notamment Stéphane Lopes pour son rôle central dans le projet DBA Companion.

Mais cette thèse est avant tout le fruit de nombreuses heures de travail, de discussions, de doutes et de succès que j'ai eu la chance de partager avec Jean-Marc Petit.

"Aujourd'hui, les techniques de l'information à l'ère du numérique qui bouleversent aussi bien la production que la consommation, devraient permettre de passer de la violence hiérarchique et de l'entropie des sociétés énergétiques à une société de l'information en réseaux plus écologique et durable, prenant en charge la responsabilité de notre avenir."
Issu du Groupe de Recherche Inter et Transdisciplinaire (GRIT), 15/01/2003

Résumé

Les bases de données relationnelles sont couramment utilisées, en pratique, pour assurer la gestion des données. La notion de contrainte d'intégrité est un atout fondamental du modèle relationnel ; les contraintes les plus importantes sont les *dépendances fonctionnelles* (DF) et les *dépendances d'inclusion* (DI). De nombreux travaux exploitent ces dépendances lors de la conception ou du réglage des bases de données, pour l'optimisation de requêtes ou pour l'intégration de données. Toutefois dans de nombreuses bases existantes, la mise en application de ces résultats est rendue impossible par l'ignorance des contraintes satisfaites par les données. Si plusieurs propositions ont été faites pour la découverte des DF satisfaites, peu de travaux ont considéré le cas des DI.

Ce mémoire aborde tout d'abord la découverte exhaustive des DI satisfaites dans une base de données relationnelle. Une méthode originale est donnée pour la découverte des DI unaires, qui améliore significativement les performances des algorithmes existants. Nous proposons ensuite un nouvel algorithme dédié à la découverte des DI n-aires. Son principe est de parcourir l'espace de recherche "par niveaux", tout en effectuant d'éventuels "sauts optimistes" pour atteindre des DI satisfaites de grande taille. Si celles-ci ne sont pas satisfaites, une mesure d'erreur est alors calculée, permettant d'orienter la suite du parcours. Afin de prendre en compte d'éventuelles incohérences dans les données, les algorithmes seront étendus à la découverte des *DI approximatives*, qui sont des DI presque satisfaites dans la base. Toutes les propositions ont été implantées et testées sur une base réelle issue du web ainsi que sur des bases synthétiques.

Une fois que les DF et les DI satisfaites dans une base de données sont découvertes, se pose alors le problème de la présentation des résultats à l'utilisateur. La simple liste des dépendances est une représentation spécialisée, permettant peu d'interaction avec un spécialiste du domaine d'application. Nous proposons une représentation alternative des DF et DI satisfaites, les *bases de données d'Armstrong informatives* (BDAI). Il s'agit de petits échantillons de la base initiale, satisfaisant *exactement* les mêmes DF et DI. Une telle représentation fournit un ensemble d'exemples et de contre-exemples (pour les dépendances non satisfaites) familiers à un expert du domaine, qui pourra alors valider les résultats, ou encore détecter des erreurs dans les données.

Les travaux présentés sont intégrés dans un outil, "DBA Companion", dédié à l'analyse et au réglage des bases de données relationnelles.

Mots-clés : Bases de données relationnelles, dépendances fonctionnelles, dépendances d'inclusion, extraction de connaissances dans les données, visualisation des connaissances, échantillons, bases de données d'Armstrong.

Table des matières

Liste des figures

Liste des algorithmes

Première partie

Introduction

Problème et motivations

Depuis son introduction dans [Cod70], le modèle relationnel s'est progressivement imposé comme un standard pour la gestion des données. Les systèmes de gestion de bases de données relationnelles (SGBDR) sont utilisés par une grande majorité d'entreprises ou administrations, pour manipuler et interroger des volumes parfois gigantesques de données. D'autres modèles de données ont récemment vu le jour pour répondre à des besoins spécifiques, comme le modèle objet pour manipuler des données complexes, ou les modèles semi-structurés acceptant des données hétérogènes notamment issues du Web. Mais les implantations directes de ces modèles se heurtent à des problèmes de performances lorsque de gros volumes de données sont considérés, et leur gestion est souvent confiée à des SGBDR.

Les fondements théoriques solides sur lesquels repose le modèle relationnel expliquent en partie son succès. Ils sont issus principalement de la théorie des ensembles, de la logique mathématique et de la théorie des modèles finis. Ils ont permis le développement de nombreux travaux théoriques et pratiques qui n'ont cessé d'enrichir et de solidifier le modèle relationnel et ses implantations jusqu'à la fin des années 90, où l'on trouve alors plusieurs ouvrages de synthèse comme [MR94b, AHV95, LL99, GMUW00].

Une caractéristique centrale du modèle relationnel est la notion de contraintes d'intégrités. D'une façon générale, il s'agit de conditions que doivent satisfaire des données pour être acceptées dans une base, afin d'assurer sa cohérence avec le monde réel. Les contraintes souvent citées comme les plus importantes sont les dépendances fonctionnelles (DF) et les dépendances d'inclusion (DI), qui véhiculent à elles seules une grande part de la sémantique des données. Elles généralisent respectivement les concepts fondamentaux de clés et de clés étrangères, et constituent la clé de voûte des méthodes de conception de bases de données relationnelles [MR94b, CTF88, LV00]. Elles permettent en effet de traduire dans le modèle relationnel des informations sémantiques issues des modèles conceptuels tels que les schémas Entité-Association [Che76]. En outre, elles servent à définir des formes normales pour des bases de données, telles que la forme normale de Boyce-Codd [Cod74] en ce qui concerne les DF seules, et la forme normale des DI [MR86a, LV00] prenant en compte les DF et les DI.

Dans le cas de bases de données existantes, la sémantique véhiculée par les DF et les DI est à l'origine de nombreux travaux. Parmi eux, des travaux de rétro-conceptions [CdS83, MM90, PTBK96, CWA99], dont le but est de retrouver le schéma conceptuel des données à partir de leur forme relationnelle, des travaux sur l'intégration de données issues de sources différentes [MHH$^+$01], ou sur la maintenance des vues dans les entrepôts de données [QGMW96, LLSV99]. Au-delà de la sémantique des bases de données, les DF et les DI apportent également une connaissance forte sur la structure des données à un instant précis, pouvant être exploitée pour la ré-écriture et l'optimisation des requêtes [Gry98, CGK$^+$99] ou encore la définition automatique d'index [CN98].

Toutefois dans de nombreuses bases existantes, la mise en application de ces résultats est rendue impossible par l'ignorance des DF et DI satisfaites. C'est le cas par exemple des nombreuses bases de données conçues rapidement, pour lesquelles aucune étude conceptuelle n'a été faite. Les informations sur la sémantique des données peuvent aussi avoir été perdues lors des changements d'administrateurs, ou encore être rendues obsolètes par une évolution non maîtrisée de la base ou des applications. Pour surmonter cette lacune, se pose alors le problème de la découverte des DF et DI satisfaites dans une base de données.

Un problème d'ECD

Ce problème peut s'inscrire dans la problématique plus large de l'extraction de connaissances dans les bases de données (ECD) [FPSSU96]. L'ECD peut se définir comme un processus de découverte de connaissances non triviales et implicites dans des données volumineuses, se décomposant en trois étapes principales (figure 1) : la préparation des données, la fouille de données proprement dite (data mining), et le post-traitement des résultats. La dernière étape peut être automatisée, ou laissée à l'appréciation des experts du domaine, auquel cas se pose le problème de la visualisation des connaissances extraites. L'étape de fouille de données est la plus traitée dans les travaux existants, notamment pour la découverte des motifs fréquents dans une base de données de transactions (dont par exemple [AS94, Bay98, PBTL99, BTP+00]).

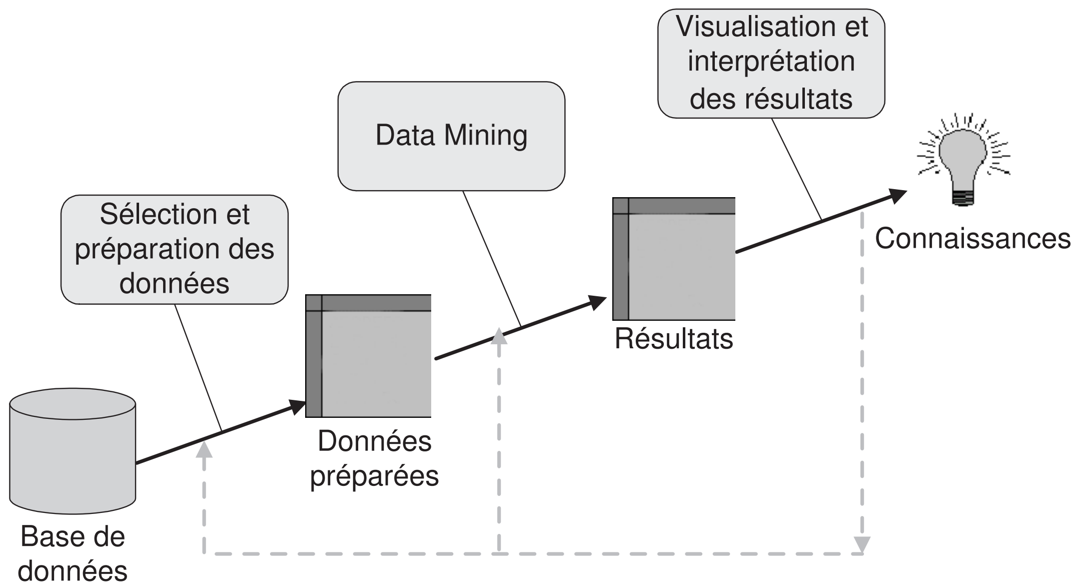

Fig. 1 – Étapes d'un processus d'extraction de connaissances dans les données

En ce qui concerne la découverte des DF, l'étape de fouille de données à donné lieu à plusieurs propositions dont [HKPT99, FS99, NC01, LPL02]. En revanche, peu de travaux ont abordé la découverte des DI comme un problème de fouille de données. Deux raisons peuvent expliquer cette absence de résultats : la complexité du problème, avec un nombre plus que factoriel de DI potentielles [KMRS92], et le fait que les études sur les DF ont historiquement précédé celles sur les DI. Le problème de la découverte des DI succède donc logiquement à celui de la découverte des DF dans ce domaine ; cette tendance est confirmée par une proposition récente parue dans [KR03].

L'étape de post-traitement des DF et DI découvertes a été peu étudiée. Une *couverture* est en général fournie à l'utilisateur ; il s'agit d'une liste condensée, mais équivalente, des règles découvertes. La qualité d'une couverture dépend surtout de sa taille et de son caractère informatif. Mais quelle que soit cette couverture, il s'agit toujours d'une *liste de règles*, parfois conséquente. Pour les experts du domaine, une telle liste est parfois difficilement lisible et exploitable, car les règles sont sorties de leur contexte. Il est alors mal aisé pour eux de comprendre *pourquoi* telle ou telle règle est satisfaite, ou n'est pas satisfaite dans les données.
Dans ce contexte, le concept de *base de données d'Armstrong* [Arm74, FV83] a été utilisé comme une représentation alternative à une simple liste de règles. Il s'agit d'une base de données vérifiant exactement un ensemble de DF et de DI pris en entrée, dont le principal intérêt reconnu est l'aide à la conception de bases de données par l'exemple [SM79, MR86b, DT95]. Nous verrons toutefois que cet intérêt est limité lorsque le problème est de comprendre les dépendances dans une base de données existante.

Contribution du mémoire

Notre contribution porte sur la découverte exhaustive des DI satisfaites dans une base de données relationnelles, et la visualisation des DF et DI découvertes dans un processus d'ECD.

La découverte exhaustive des DI est un problème dont la complexité prend sa source à deux niveaux différents : le nombre important d'expressions de DI potentiellement satisfaites, et le coût élevé d'un test de satisfaction d'une DI sur la base. En outre, alors qu'il est possible dans d'autres problèmes de fouille de données (comme la découverte de motifs fréquents dans des bases de transactions) de tester un ensemble de candidats en un seul parcours de la base, chaque candidat conduit généralement à un test sur la base dans le cas de la découverte de dépendances [MT97]. Nous proposons de séparer le problème en deux sous-problèmes distincts : la découverte des *DI unaires* (définies entre singletons d'attributs), et la découverte des *DI n-aires* (définies entre séquences d'attributs), connaissant les DI unaires. Plusieurs justifications motivent cette démarche :

- Les DI unaires constituent un "point d'entrée" privilégié dans l'espace de recherche constitué des DI potentielles. Leur découverte, quadratique dans le nombre d'attributs, permet généralement d'élaguer une grande partie de l'espace de recherche des DI d'arité supérieure.
- Contrairement à ce que nous verrons pour les DI n-aires, aucun élagage efficace ne peut être effectué pour les DI unaires. Quelque soit l'ensemble des DI satisfaites dans la base, le nombre de candidats est quadratique dans le nombre d'attributs.

En pratique, étant donné le coût élevé d'un test de DI, la découverte des DI unaires par une approche naïve peut s'avérer tellement lourde qu'elle compromet tout espoir de parcourir le reste de l'espace de recherche.

Pour palier ces limites, une méthode spécifique pour la découverte des DI unaires est présentée. Elle est basée sur une réorganisation des données conduisant à une nouvelle caractérisation des DI unaires satisfaites, puis à un algorithme efficace pour leur découverte [DLP02a].

À partir de la connaissance des DI unaires, nous traitons le problème de la découverte des DI n-aires ($n > 1$). Un espace de recherche sera défini, composé d'un sous-ensemble partiellement ordonné des DI potentiellement satisfaites. Nous montrerons que les DI satisfaites les plus grandes dans cet espace constituent une solution au problème de la découverte des DI. Étant donnée une expression de DI, cette solution permet en effet de décider efficacement si elle est satisfaite ou non. Nous proposons alors deux algorithmes, ayant pour objectif d'atteindre cette solution.

Le premier est l'algorithme *Mind*, effectuant un parcours *par niveaux* de l'espace de recherche. Nous le qualifions de *pessimiste*, car la plupart des DI candidates sont supposées être non satisfaites ; un niveau n'est exploré que lorsque tous les niveaux qui le précèdent ont été explorés. Toute DI non satisfaite découverte, permet d'élaguer une partie de l'espace de recherche restant, par une propriété d'anti-monotonie. La découverte des DI par niveaux fait l'objet d'un exemple de quelques lignes dans [MT97] ; beaucoup de points que nous traitons ici n'y sont toutefois pas abordés, comme l'étape clé de génération des candidats, la réduction de l'espace de recherche en fixant un ordre sur les attributs, ou encore la prise en compte des doublons [DRLP01, DLP02a].

L'algorithme *Zigzag* utilise un mode de parcours différent pour découvrir la solution. Dans un premier temps, un parcours par niveaux permet de considérer les DI de faible arité, et d'éliminer une partie de l'espace de recherche par anti-monotonie. À partir d'un certain niveau, nous formulons l'hypothèse optimiste que toutes les DI non encore éliminées sont satisfaites, et proposons de considérer directement les plus grandes. Cette hypothèse est confortée par une propriété du modèle relationnel. Ces "sauts optimistes" sont réalisés grâce à une représentation ensembliste des DI, ramenant le problème à la génération des transversaux minimaux d'un hypergraphe. Lorsque les grandes DI ainsi générées s'avèrent fausses, une mesure d'erreur donne une estimation de la distance à parcourir pour la correction, permettant d'orienter la suite de la recherche [DP03c, DP03b]. Ce mode de parcours est original, et offre des perspectives d'application à d'autres problèmes de fouille de données comme la découverte des motifs fréquents.

Afin de prendre en compte d'éventuelles erreurs de saisie ou des incohérences dans les données, pouvant contredire "accidentellement" une DI potentiellement intéressante, nous utilisons la notion de *DI approximatives*, pour lesquelles la condition de satisfaction accepte un certain nombre de contre-exemples. Nous montrons comment les algorithmes décrits pour les DI exactes peuvent s'adapter à la découverte des DI approximatives [DLP02a].

Pour la visualisation des DF et des DI satisfaites dans une base de données, nous définissons les *bases de données d'Armstrong informatives* (BDAI), qui sont des petits échantillons de la base de départ vérifiant *exactement* les mêmes DF et DI. Elles regroupent donc un ensemble exhaustif d'exemples, issus du monde réel, parfaitement exploitables par un expert du domaine pour l'aider dans l'analyse de la sémantique de la base.
L'étude des BDAI est séparée en deux parties. La première ne considère que les DF, avec la notion de *relation d'Armstrong informative* (RAI) [DLP01, DLP02b]. Une propriété assure l'existence de RAI de petite taille (avec peu de lignes) pour toute relation prise en entrée, et conduit à la mise en place d'algorithmes pour leur construction. Nous soulevons également le problème de la construction de RAI de taille minimale ; à partir d'une représentation graphique du problème, des pistes seront ouvertes pour en évaluer la complexité.

Nous étendons ensuite l'étude à la prise en compte à la fois des DF et des DI [DP03a], pour obtenir des BDAI. Un algorithme sera donné pour construire des petites BDAI ; une borne pour la taille des échantillons ainsi construits découlera de cet algorithme.

Ce travail s'inscrit dans un projet appelé "DBA Companion", dont l'objectif est l'aide à l'administration et au réglage logique des bases de données relationnelles [DLPT03]. Un prototype a été réalisé [LDP02, LDP04], dans lequel les propositions présentées ici sont intégrées, ou en cours d'intégration. L'utilisateur se connecte à une base de données via une interface graphique ; il peut alors rechercher les clés, les DF, les DI, ou construire des bases de données d'Armstrong informatives.

Le comportement des algorithmes de fouille de données, pour la découverte des DI, seront évalués sur des bases de données synthétiques. Mais l'ensemble de nos propositions portent sur la compréhension de bases existantes, et il nous a semblé intéressant de les confronter à des situations réelles. L'obtention de bases réellement en activité est malheureusement difficile, et souvent dépendante de partenariats avec des entreprises externes.

C'est pourquoi nous avons utilisé une base de données universitaire, mais peuplée de données issues de situations réelles. Il s'agit de la base de données **movies**, issue de l'*UCI KDD Archive* [Bay99], et maintenue à l'université de Stanford. Cette base contient une liste de 10000 films, avec des informations sur les acteurs, le casting, les réalisateurs, les producteurs, les studios, etc. Elle a été importée sous Oracle 9i avec SQL*Loader. Les caractéristiques des relations sont reportées dans la figure 2. Les notions de bases de données, relations, attributs et tuples seront rappelées dans les préliminaires.

Nous utiliserons également une version exemple de la base de données **movies** pour illustrer les notions et les méthodes présentées. Il s'agit de deux relations faisant partie du schéma réel de **movies** ; seuls quelques changements mineurs ont été apportés dans les tuples et dans les attributs pour clarifier au mieux les différents exemples. La figure 3, qui représente cet exemple courant, sera abondamment référée dans la suite. Pour des raisons de simplicité, chaque attribut pourra être nommé par la lettre unique donnée entre parenthèses.

FIG. 2 – Caractéristiques de la base de données réelle **movies**.

Nom de la relation	nb d'attributs	nb de tuples
actors	12	6728
casts	7	45442
movies	12	11405
people	11	3304
remakes	7	1188
studios	9	195

remakes

ID (A)	*TITLE* (B)	*YEAR* (C)	*FRAC* (D)	*PRID* (E)	*PRTITLE* (F)
16	Le voyage	1959	0.9	11	Mademoiselle Fifi
29	Cléopâtre	1962	0.3	23	Cléopâtre
28	Des soeurs courageuses	1938	0.9	17	Quatre soeurs
15	Les amours de Carmen	1948	0.2	22	Gilda
21	Une heure avec toi	1932	0.9	18	Le cercle du mariage
16	Le voyage	1959	0.3	17	Quatres soeurs
40	L'Oeuvre au noir	1990	0.8	42	Le nom de la rose
12	Young at heart	1938	0.9	30	Casablanca
40	L'Oeuvre au noir	1990	0.1	14	Thérèse

movies

FILMID (G)	*TITLE* (H)	*YEAR* (I)
11	Mademoiselle Fifi	1944
12	Young at heart	1938
13	L'exorciste	1971
43	Thérèse	1980
15	Les amours de Carmen	1948
16	Le voyage	1959
17	Quatre soeurs	1938
18	Le cercle du mariage	1924
21	Une heure avec toi	1932
22	Gilda	1946
23	Cléopâtre	1934
28	Des soeurs courageuses	1938
29	Cléopâtre	1962
30	Casablanca	1944
40	L'Oeuvre au noir	1990
41	Le manitou	1978
42	Le nom de la rose	1985

FIG. 3 – Version exemple de la base de données **movies**.

Organisation du mémoire

La suite de ce mémoire est structurée de la façon suivante.

Un ensemble de définitions et de résultats préliminaires, concernant le modèle relationnel, sont rappelés dans la deuxième partie. Ils concernent principalement la structure du modèle relationnel et la notion de contraintes d'intégrités, dont les DF et les DI sont des cas particuliers.

La troisième partie traite du problème de la découverte exhaustive des DI dans une base de données relationnelles. Le chapitre 2 dresse un état de l'art des travaux relatifs à ce problème. Une méthode spécifique à la découverte des DI unaires est donnée dans le chapitre 3. Puis le chapitre 4 considère la découverte des DI n-aires. Les algorithmes proposés sont étendus au cas des DI approximatives dans le chapitre 5. Enfin, le chapitre 6 présente les expérimentations réalisées sur les implantations de nos algorithmes.

La visualisation par l'exemple des DF et DI satisfaites dans une base de données est abordée dans la quatrième partie. La présentation des bases de données d'Armstrong constitue l'état de l'art, dans le chapitre 7. Le chapitre 8 introduit, puis étudie, les relations d'Armstrong informatives qui ne concernent que les DF. Puis le chapitre 9 étend cette notion aux bases de données d'Armstrong informatives, qui prennent en compte les DF et les DI. Des expérimentations sur la base de données **movies** seront finalement présentées dans le chapitre 10.

Enfin, nous conclurons et donnerons quelques perspectives dans la cinquième et dernière partie de ce mémoire.

Deuxième partie

Préliminaires

Chapitre 1

Rappels sur le modèle relationnel

Sommaire

Le but de ces préliminaires n'est pas de définir en détail le modèle relationnel, mais de préciser quelques uns de ses aspect fondamentaux pour aider à la compréhension du mémoire. Le modèle relationnel est décrit dans notamment [MR94b, AHV95, LL99], avec des notations parfois différentes. Les notations choisies dans la suite sont celles utilisées dans [MR94b].

1.1 Structure des données

Dans le modèle relationnel, les données sont stockées dans des tables. Les colonnes des tables ont des noms qu'on appelle *attributs*. Les attributs sont nommés par des lettres capitales du début de l'alphabet (telles que A, B ou C).

Les ensembles d'attributs sont nommés par des lettres capitales de la fin de l'alphabet (telles que X, Y ou Z). Un ensemble constitué d'un seul attribut est nommé par l'attribut lui-même, et la concaténation dénote l'union ensembliste. Ainsi ABC dénote l'ensemble $\{A, B, C\}$, et XY signifie $X \cup Y$.

Une *séquence d'attributs* est une liste ordonnée d'attributs (non forcément distincts), noté $< A_1 ... A_n >$. Toutefois, lorsque le contexte sera clair, les mêmes notations seront employées pour une séquence ou un ensemble d'attributs. Soit A un attribut, le domaine de A, noté $Dom(A)$, est l'ensemble de toutes les valeurs possibles pour l'attribut A. Si X est un ensemble d'attributs, on définit $Dom(X) = \cup_{A \in X} Dom(A)$.

Définition 1 (Schéma de relations et schéma de bases de données) Un schéma de relation $R = \{A_1, ..., A_n\}$ est un ensemble fini d'attributs distincts. Un schéma de base de données $\mathbf{R} = \{R_1, ..., R_n\}$ est un ensemble fini de schémas de relations.

Définition 2 (Tuple) Un tuple t sur un schéma de relations R est une application $t : R \rightarrow \cup_{A \in R} Dom(A)$, où $t(A) \in Dom(A) \forall A \in R$.

Définition 3 (Relation et base de données) Une *relation* r sur un schéma de relation R est une ensemble de tuples sur R. Une *base de données* $\mathbf{d} = \{r_1, ..., r_n\}$ sur un schéma de base de données $\mathbf{R} = \{R_1, ..., R_n\}$ est un ensemble de relations définies sur les éléments de $\mathbf{R}$.

Si r est une relation sur un schéma R, on dit que R est le schéma de r, et que r est une *instance* de R.

Soit r une relation, le nombre de tuples dans r, noté $|r|$, est appelé *cardinalité de* r. Par extension, si $\mathbf{d} = \{r_1, ..., r_n\}$ est une base de données, le nombre de tuples dans $\mathbf{d}$, noté $|\mathbf{d}|$, est appelé *cardinalité de* $\boldsymbol{d}$ et est égal à $\sum_{i=1}^{n} |r_i|$.

Exemple 1 Sur l'exemple de la figure 3 page 8, la base de données **movies** est un ensemble de deux relations : $\{remakes, movies\}$. Le schéma de $movies$ est un ensemble de 3 attributs : $\{FILMID, TITLE, YEAR\}$. Chacune des lignes des relations représente un tuple de la relation.

Langages de requêtes et de mises à jour Dans la suite, nous utilisons l'algèbre relationnelle pour exprimer les interrogations des données sur un plan théorique. Les implantations des requêtes exprimées sont réalisées en SQL (Structured Query Langage). Le détail de ces langages dépassant de loin la portée de ce mémoire, le lecteur est renvoyé à [AHV95, LL99] pour des présentations approfondies.

Nous nous contenterons de définir la *projection* d'un tuple ou d'un ensemble de tuples sur un ensemble d'attributs, car cet opérateur sera fréquemment utilisé dans la suite.

Définition 4 (Projection) Soit t un tuple sur R. La *projection* de t sur un ensemble d'attributs $X \subseteq R$, notée $t[X]$, est la restriction de t sur X. Soit r une relation sur R, la projection de r sur X, notée $\pi_X(r)$ est définie par $\pi_X(r) = \{t[X] \mid t \in r\}$.

A l'instar de [MR94b, AHV95, LL99], on étend de façon naturelle l'opérateur de projection sur une *séquence d'attributs*, de la façon suivante : Soit $X =< A_1, ..., A_k >$ une séquence d'attributs d'un schéma de relation R, et t un tuple sur R. La *projection sur* X de t est la séquence $< t[X] >=< t[A_1], ..., t[A_k] >$. Si r est une relation sur R, la projection de r sur la séquence X, notée $\pi_X(r)$ est définie par $\pi_X(r) = \{< t[X] >\mid t \in r\}$.

Si A est un attribut sur un schéma R et r une relation sur $\mathbf{R}$, alors l'ensemble $\pi_A(r)$ est appelé le *domaine actif* de A.

1.2 Contraintes d'intégrité

1.2.1 Généralités

Les contraintes d'intégrité sont des conditions que doivent remplir les bases de données pour être cohérentes. Parmi les plus importantes du modèle relationnel, on trouve les dépendances fonctionnelles et les dépendances d'inclusion, que nous définirons plus en détail dans la suite. Auparavant, nous revenons sur quelques concepts communs aux contraintes d'intégrité dans le modèle relationnel, empruntés à [LL99].

Une *classe de contraintes d'intégrité* correspond à un type particulier de contraintes d'intégrité sur un schéma de relation ou de base de données. Par exemple, l'ensemble de toutes les dépendances fonctionnelles sur un schéma de relation R est une classe de contrainte d'intégrité. Un *ensemble de contraintes d'intégrité*, noté Σ, est un ensemble de contraintes appartenant à une classe de contraintes donnée.

Pour plus de clarté, nous considérons dans la suite un ensemble de contraintes d'intégrité sur un schéma de relation ; les concepts peuvent s'adapter à un ensemble de contraintes sur un schéma de base de données par simple transposition.

On note le fait qu'une relation r sur un schéma R *satisfait* une contrainte d'intégrité α par $r \models \alpha$. Dans le cas contraire, on dit que r *viole* la contrainte α, noté $r \not\models \alpha$. On dit que r *satisfait* un ensemble Σ de contraintes d'intégrité sur R, noté $r \models \Sigma$ si pour toute contrainte $\alpha \in \Sigma, r \models \alpha$.

Un ensemble Σ de contraintes d'intégrité sur R *implique logiquement* une contrainte α, noté $\Sigma \models \alpha$, si pour toute relation r sur R, on a :

$$r \models \Sigma \Longrightarrow r \models \alpha$$

Un ensemble de contraintes Σ sur R *implique logiquement* un autre ensemble de contraintes Γ sur R, noté $\Sigma \models \Gamma$, si pour tout $\alpha \in \Gamma, \Sigma \models \alpha$. On appelle *problème d'inférence* (ou *problème d'implication*) d'une classe de contraintes le problème de décider si, étant donné un ensemble de contraintes Σ et une contrainte α, on a $\Sigma \models \alpha$.

Une *règle d'inférence* est une règle permettant de déduire des contraintes valides à partir de contraintes connues (pour une classe particulière de contraintes). Un *système d'inférence* est un ensemble de règles d'inférence.

Soit S un système d'inférence et Σ un ensemble de contraintes. On note $\Sigma \vdash_S \alpha$ le fait de pouvoir *inférer* la contrainte α à partir de Σ en appliquant les règles de S. Les propriétés fondamentales d'un système d'inférence sont la *justesse* et la *complétude*.

Définition 5 (Système d'inférence juste et complet) Soit Σ un ensemble de contraintes d'intégrité et α une contrainte de la même classe.

Un système d'inférence S est *juste* si $\Sigma \vdash_S \alpha \Longrightarrow \Sigma \models \alpha$.

Un système d'inférence S est *complet* si $\Sigma \models \alpha \Longrightarrow \Sigma \vdash_S \alpha$.

La notion d'inférence de contraintes à partir d'autres contraintes conduit à définir la *fermeture* d'un ensemble de contraintes. Pour aider à la compréhension, nous rappelons tout d'abord ce qu'est un opérateur de fermeture.

Définition 6 (Opérateur de fermeture) Soit R un ensemble fini et $\mathcal{P}(R)$ l'ensemble des parties de R. Un *opérateur de fermeture* $.^+$ est une application de $\mathcal{P}(R)$ dans $\mathcal{P}(R)$ remplissant les trois conditions suivantes $(\forall X, Y \in \mathcal{P}(R))$:
- $X \subseteq X^+$ (extensivité),
- si $X \subseteq Y$ alors $X^+ \subseteq Y^+$ (isotonie), et
- $(X^+)^+ = X^+$ (idempotence).

Nous pouvons alors définir la fermeture d'un ensemble de contraintes Σ, comme étant l'ensemble de toutes les contraintes qu'il est possible d'inférer à partir de Σ avec un système d'inférence donné.

Définition 7 (Fermeture d'un ensemble de contraintes) La *fermeture* d'un ensemble de contraintes Σ sur R (par rapport à un système d'inférence donné), notée Σ^+, est défini par $\Sigma^+ = \{\alpha \mid \Sigma \vdash \alpha\}$.

L'opérateur ainsi défini est un opérateur de fermeture. On dit qu'un ensemble Σ de contraintes est fermé si $\Sigma^+ = \Sigma$. On dit également qu'un ensemble de contraintes Σ est une *couverture* d'un autre ensemble de contraintes Γ si $\Sigma^+ = \Gamma^+$; la relation de couverture est donc une relation d'équivalence entre les ensembles de contrainte.

1.2.2 Dépendances fonctionnelles et clés

Les *dépendances fonctionnelles* forment la classe de contraintes la plus étudiée. Elles généralisent la notion de clé d'une relation, et permettent de caractériser les problèmes de mises à jour dans les données.

Définition 8 (Dépendance fonctionnelle (DF)) Une *dépendance fonctionnelle (DF)* sur un schéma de relation R est une expression de la forme $R : X \rightarrow Y$ (ou simplement $X \rightarrow Y$ lorsque R est implicite), où $X, Y \subseteq R$. Une DF $X \rightarrow Y$ est dite *triviale* si $Y \subseteq X$ et *standard* si $X \neq \emptyset$.

Une DF est donc une contrainte définie sur un schéma, devant être satisfaite dans toute relation définie sur ce schéma, selon la définition suivante :

Définition 9 (Satisfaction d'une DF) Soit r une relation sur R. Une DF $R : X \rightarrow Y$ est *satisfaite* dans r, noté $r \models R : X \rightarrow Y$, si $\forall t_1, t_2 \in r$ on a $t_1[X] = t_2[X] \Longrightarrow t_1[Y] = t_2[Y]$.

Si $X \rightarrow Y$ est satisfaite dans une relation r, on dit aussi que X *détermine* Y dans r. On note F_r l'ensemble de toutes les DF satisfaites dans une relation r. Par extension, on note $F_{\mathbf{d}}$ l'ensemble des DF définies sur les éléments de $\mathbf{d}$, soit $F_{\mathbf{d}} = \cup_{r \in \mathbf{d}} F_r$, à un renommage près assurant l'unicité des attributs.

Le système d'inférence des DF de référence est le *système d'Armstrong* [Arm74].

Définition 10 (Système d'inférence d'Armstrong) Soit F un ensemble de DF sur un schéma de relation R. Les règles d'inférence de DF suivantes sont appelées *système d'inférence d'Armstrong* :
– Réflexivité : si $Y \subseteq X \subseteq R$ alors $F \vdash X \rightarrow Y$.
– Augmentation : si $F \vdash X \rightarrow Y$ et $F \vdash W \subseteq R$ alors $F \vdash XW \rightarrow YW$.
– Transitivité : si $F \vdash X \rightarrow Y$ et $F \vdash Y \rightarrow Z$ alors $F \vdash X \rightarrow Z$

Théorème 1 *[Arm74] Le système d'inférence d'Armstrong est juste et complet pour les DF.*

Une conséquence de ce résultat est que $F \models \alpha \Leftrightarrow F \vdash \alpha$; nous utiliserons dans la suite la notation $F \models \alpha$. Ajoutons que le problème d'inférence des DF est linéaire dans le nombre de DF en entrée.

Nous avons rappelé plus haut la définition d'un opérateur de fermeture, et défini la fermeture d'un ensemble de contraintes. Cette définition s'applique naturellement aux DF, et on note donc F^+ l'ensemble de toutes les contraintes pouvant être inférées à partir d'un ensemble F de DF.

Mais les DF, de part leur définition, peuvent être vues elles-mêmes comme *des implications entre des ensembles d'attributs*, et permettent ainsi de définir un opérateur de fermeture pour les ensembles d'attributs.

Définition 11 (Fermeture d'un ensemble d'attributs) Soit R un schéma de relation et F un ensemble de DF sur R. La *fermeture d'un ensemble d'attributs* $X \subseteq R$ relativement à F, notée X_F^+ (X^+ lorsque F est implicite dans le contexte), est définie par $X^+ = \bigcup\{Y \mid F \models X \longrightarrow Y\}$ (ou encore $X^+ = \bigcup\{Y \mid X \rightarrow Y \in F^+\}$

Cet opérateur de fermeture permet de définir simplement la notion de clé qui est primordiale dans le modèle relationnel. Une clé d'une relation est un ensemble minimal d'attributs qui permet de déterminer l'ensemble des attributs de la relation.

Définition 12 (Clé d'une relation) Soit R un schéma de relation et F un ensemble de DF sur R. Un ensemble d'attributs $K \subseteq R$ est une *clé minimale* de R relativement à F (ou simplement clé de R) si les deux conditions suivantes sont remplies :
- *unicité* : $K^+ = R$ et
- *minimalité* : $\forall Y \subset K, Y^+ \neq R$.

Une *clé primaire* de R est un ensemble d'attributs choisi parmi les clés de R.

Exemple 2 Considérons la relation *movies* de la base de données **movies** (figure 3 page 8). On a $movies \models G \longrightarrow HI$. Ainsi, $G^+ = GHI$. G est donc une clé de *movies*.

A partir de la définition de la fermeture d'un ensemble d'attributs, on peut associer à chaque ensemble F de DF un ensemble de fermés ; un fermé pour F est un ensemble d'attributs qui est égal à sa fermeture relativement à F.

Définition 13 (Fermés d'un ensemble de DF) Soit F un ensemble de DF sur un schéma R. L'ensemble des *fermés de F*, noté $CL(F)$ est défini par $CL(F) = \{X \subseteq R \mid X^+ = X\}$.

On appelle générateurs d'un ensemble de DF F, noté $GEN(F)$, la famille unique minimale d'éléments de $CL(F)$ qui permet de générer $CL(F)$ par intersections. Notons que si G est une couverture de F on a : $CL(F) = CL(G)$ et donc $GEN(F) = GEN(G)$.

Exemple 3 Considérons la base de données **movies** (figure 3 page 8). Une couverture des DF satisfaites dans la relation *movies* est $F = \{G \longrightarrow HI, HI \longrightarrow G\}$. Parmi les 8 sous-ensembles du schéma de *movies*, les fermés de F sont : $CL(F) = \{\emptyset, H, I, GHI\}$. Les générateurs de F sont : $GEN(F) = \{H, I, GHI\}$.

1.2.3 Dépendances d'inclusion et clés étrangères

Les dépendances d'inclusion (DI) généralisent les clés étrangères qui, avec les clés, constituent les contraintes d'intégrité fondamentales du modèle relationnel.

Les DI se différencient des DF sur plusieurs points. D'abord, elles peuvent être définies entre attributs de relations différentes, et possèdent ainsi un caractère plus “global” que les DF dont la définition est locale à une relation (bien que l'on parle parfois de *DF globale* [MR94a], que nous n'abordons pas ici). Ensuite, les DI sont définies non pas entre deux ensembles quelconques d'attributs, mais *entre deux séquences d'attributs de même taille.*

Définition 14 (Dépendance d'inclusion) Soit $\mathbf{R}$ un schéma de base de données. Une dépendance d'inclusion sur $\mathbf{R}$ est une expression de la forme $R[X] \subseteq S[Y]$, où $R, S \in \mathbf{R}$, X et Y sont des séquences d'attributs respectivement de R et de S, et $|X| = |Y|$.

Pour simplifier les notations, et lorsque le contexte sera parfaitement clair (unicité des noms d'attributs de la base), il nous arrivera d'omettre de préfixer les séquences par le nom de la relation.

Étant donnée une DI $R[X] \subseteq S[Y]$, X est appelée la *partie gauche* et Y la *partie droite* de la DI. On appelle *taille* (ou *arité*) d'une DI i, et on note $|i|$, le nombre d'attributs dans la partie droite (ou gauche) de i. Une DI de taille 1 est appelée DI *unaire*. Les DI de taille supérieure à 1 sont appelées DI *n-aires*.

Les parties droites et gauches sont des séquences d'attributs, dans lesquelles le même attribut peut apparaître plusieurs fois. Certaines définitions diffèrent légèrement de celle présentée, en se restreignant à des séquences d'attributs distincts. Par souci d'exhaustivité, nous nous placerons dans le cas plus général, autorisant donc la présence de doublons dans la définition des séquences (page 13). Nous distinguerons néanmoins le cas des DI sans doublons lorsque cela sera nécessaire.

Parmi les DI, nous distinguerons les cas particuliers des DI *triviales*, des DI *pseudo-*

triviales et des DI *redondantes* :

Définition 15 (DI triviales, pseudo-triviales et redondantes) Soit $i = R[A_1 ... A_n] \subseteq S[B_1 ... B_n]$ une DI.
- Si $R = S$ et $\forall k = 1, .., n$ on a $A_k = B_k$, alors i est dite *triviale* ;
- Si $R = S$ et $\exists \{k_1, ..., k_l\} \subseteq \{1, ..., n\}$ un ensemble d'indices pour lesquels $A_{k_1} = B_{k_1}, ..., A_{k_l} = B_{k_l}$, alors i est *pseudo-triviale* ;
- Si $\exists \{k_1, ..., k_l\} \subseteq \{1, ..., n\}$, $l \geq 2$, un ensemble d'indices pour lesquels $A_{k_1} = ... = A_{k_l}$ et $B_{k_1} = ... = B_{k_l}$, alors i est dite *redondante* [Mit83].

Exemple 4 Soit $R = ABC$ et $S = DEF$ deux schémas de relations. Alors la DI $AB \subseteq AB$ est une DI triviale ; la DI $AB \subseteq AC$ est une DI pseudo-triviale et la DI $ABB \subseteq DEE$ est une DI redondante.

Une *contrainte d'intégrité référentielle* est une DI dont la partie droite est une clé ; la partie gauche d'une contrainte d'intégrité référentielle est appelée *clé étrangère*.

Intuitivement, une DI est satisfaite dans une base de données si toutes les valeurs prises par la partie gauche apparaissent dans la partie droite.

Définition 16 (Satisfaction d'une DI) Soit $\mathbf{d} = \{r_1, ..., r_n\}$ une base de données sur un schéma $\mathbf{R} = \{R_1, ..., R_n\}$. Une dépendance d'inclusion $R_i[X] \subseteq R_j[Y]$ sur $\mathbf{R}$ est *satisfaite* dans $\mathbf{d}$, noté $\mathbf{d} \models R_i[X] \subseteq R_j[Y]$, si $\forall t_i \in r_i, \exists t_j \in r_j \mid t_i[X] = t_j[Y]$ (de manière équivalente, $\pi_X(r_i) \subseteq \pi_Y(r_j)$).

Exemple 5 Considérons la base de données **movies** (figure 3 page 8). Les expressions $E \subseteq G$ et $F \subseteq H$ sont des dépendances d'inclusion unaires définies sur le schéma de **movies**. On a **movies** $\not\models F \subseteq H$ puisque la valeur "Quatres soeurs" de l'attribut F n'apparaît pas dans les valeurs de l'attribut H. Cette DI est contredite par une erreur de saisie (et de grammaire), ayant ajouté un "s" au mot "Quatre". En revanche, on peut vérifier que **movies** $\models E \subseteq G$.

Nous aurons parfois à distinguer, dans la suite, le cas de DI circulaires et celui de DI non circulaires.

Définition 17 (Ensemble de DI circulaire) Un ensemble de DI I sur un schéma $\mathbf{R}$ est dit *circulaire* si l'une des deux conditions suivantes est remplie :
- il existe une DI non triviale de la forme $R[X] \subseteq R[Y]$ ou

– il existe m schémas de relation distincts $R_1, ..., R_m \in \mathbf{R}$, avec $m > 1$, tels que I contient les DI : $R_1[X_1] \subseteq R_2[Y_2]$,$R_2[X_2] \subseteq R_3[Y_3]$,...,$R_m[X_m] \subseteq R_1[Y_1]$.

En outre, un ensemble *circulaire propre* de DI est un ensemble de DI qui est soit non circulaire, soit dont tous les cycles $R_1[X_1] \subseteq R_2[Y_2]$,$R_2[X_2] \subseteq R_3[Y_3]$,...,$R_m[X_m] \subseteq R_1[Y_1]$ sont tels que $Y_1 = X_1$.

Les règles d'inférence pour les DI acceptant les doublons sont appelées *système d'inférence de Mitchell* [Mit83].

Définition 18 (Système d'inférence de Mitchell) Soit I un ensemble de DI sur un schéma de base de données $\mathbf{R}$. Les règles d'inférence de DI suivantes sont appelées *système d'inférence de Mitchell* :
– Reflexivité : $I \vdash R[X] \subseteq R[X]$.
– Projection et permutation :
si $I \vdash R[A_1...A_n] \subseteq S[B_1...B_n]$ alors $I \vdash R[A_{\sigma 1}...A_{\sigma m}] \subseteq S[B_{\sigma 1}...B_{\sigma m}]$ pour chaque séquence $\sigma 1...\sigma m$ d'entier distincts dans $\{1...n\}$
– transitivité : si $I \vdash R[X] \subseteq S[Y]$ et $I \vdash S[Y] \subseteq T[Z]$ alors $I \vdash R[X] \subseteq T[Z]$
– Substitution des équivalents : si $I \vdash R[AB] \subseteq S[CC]$ et $I \vdash \alpha$, où α est une DI ayant la séquence $R[X]$ en partie gauche ou droite, alors $I \vdash \beta$, où β est une DI obtenue en substituant une ou plusieurs occurrences de B dans $R[X]$ par A.
– Redondance : Si $I \vdash R[X] \subseteq S[Y]$, alors $I \vdash R[XU] \subseteq S[YV]$ si $R[U] \subseteq S[V]$ peut être obtenue par permutation et/ou projection sur $R[X] \subseteq S[Y]$.

On observe que les DI obtenues par la première règle d'inférence sont des DI triviales, et celles issues de la dernière règle d'inférence des DI redondantes.

Théorème 2 *[Mit83] Le système d'inférence de Mitchell est juste et complet pour les DI.*

Si l'on exclut la présence de doublons dans les parties droites et gauches, on enlève les deux dernières règles du système de Mitchell pour obtenir *le système de Casanova et al.* [CFP84], qui est aussi juste et complet. La justesse et la complétude de ces systèmes impliquent l'équivalence des notations $I \models \alpha$ et $I \vdash \alpha$; nous utiliserons dans la suite la notation $I \models \alpha$. Le problème de l'inférence (implication) des DI est un problème PSPACE-complet dans le cas général [CFP84], NP-complet dans la classe des DI non-circulaires [CKS86] ou circulaires propres [LL99], et polynomial dans la classe des DI unaires [KCV83].

Les notions de fermeture et de couverture d'un ensemble de contraintes s'appliquent aux DI. Ainsi, si I est un ensemble de DI, on note I^+ l'ensemble des DI pouvant être inférées à partir de I. Si J est un autre ensemble de DI, I est une couverture de J si $I^+ = J^+$.

1.2.4 Interactions entre les DF et les DI

Lorsque la classe de contraintes considérée englobe à la fois les DF et les DI, des interactions apparaissent induisant de nouvelles contraintes, non inférées par les DF ou les DI prises séparément.

Exemple 6 Considérons par exemple un schéma de relation $\mathbf{R} = \{R, S\}$ avec $R = ABC$ et $S = DEF$, et l'ensemble de contraintes définies sur $\mathbf{R}$:$\Sigma = \{A \longrightarrow B, DE \subseteq AB\}$. De cet ensemble de contraintes, on peut inférer la DF $D \longrightarrow E$ par la règle *Pullback* [Mit83], alors que cette DF ne peut pas être déduite par le système d'inférence d'Armstrong.

Un système d'inférence juste et complet pour les DF et les DI est donné dans [Mit83]. Le problème est toutefois indécidable dans le cas général [Mit83]. Si l'ensemble des DI est non circulaire ou circulaire propre, le problème reste EXPTIME-Complet [CK85]. Ces résultats négatifs ont amené à identifier les cas de *non-interaction* entre des ensembles de DF et de DI.

Définition 19 (Non-interaction entre DF et DI) Soit F un ensemble de DF et I un ensemble de DI. On dit que F et I *n'ont pas d'interaction* si $\forall\alpha$ (DF ou DI) telle que $F \cup I \models \alpha$, on a $F \models \alpha$ ou $I \models \alpha$.

Par exemple, un cas de non-interaction consiste en un ensemble $F \cup I$ de DF et DI sur un schéma $\mathbf{R}$ tels que $\mathbf{R}$ est en *Forme normale de Boyce-Codd*[Cod74] par rapport à F, et toutes les parties droites des DI dans I sont des clés (on dit alors que I est à *base de clé*).

Pour finir, précisons qu'aucune condition *nécessaire et suffisante* n'a été donnée sur un ensemble $F \cup I$ assurant qu'il n'y ait pas d'interaction entre F et I [LL99].

Troisième partie

Découverte des dépendances d'inclusion

Chapitre 2

Etat de l'art

Sommaire

2.1 Résultats sur la complexité du problème

Le problème de la découverte des DI a été étudié de façon théorique dans [KMRS92]. On y souligne que le nombre de DI potentiellement satisfaites entre deux relations composées chacune de N attributs est supérieur à $N!$.

Les auteurs considèrent également le sous-problème suivant :

Problème $FULLINDEXISTENCE$: *Soient deux relations r et s sur des schémas R et S, avec $|R| = |S| = N$. Soit X, une séquence composée de tous les attributs de R. Existe-t-il une séquence Y dans S telle que $\{r, s\} \models R[X] \subseteq S[Y]$?*

Ce problème est démontré NP-complet, par une réduction du problème d'isomorphisme de sous-graphe.

Malgré ces différents résultats, les auteurs précisent eux-mêmes que les bases de données qu'ils considèrent sont hautement artificielles, et n'excluent pas la possibilité de développer des algorithmes efficaces en pratique.

2.2 Découverte des DI unaires

La découverte des DI unaires est un sous-problème important du problème général qui nous intéresse. Il s'agit d'un problème quadratique dans le nombre d'attributs, puisque chaque couple d'attributs forme une DI potentielle. Ainsi, le nombre total de DI unaires potentielles (en excluant les triviales) est $N(N-1)$, où N est le nombre d'attributs dans la base.

2.2.1 Approche exhaustive par candidats

Dans [BB95], le problème de la découverte exhaustive des DI unaires est traité. Une approche d'exploration des candidats de type "générer/tester" (en générant puis testant des DI potentielles), est utilisée, conduisant basiquement à $N(N-1)$ tests sur la base. Afin de réduire ce nombre, les auteurs utilisent la troisième règle d'inférence des DI, la transitivité. Si cette optimisation peut s'avérer intéressante dans le cas où de nombreuses DI unaires sont satisfaites, en pratique on constate qu'elle s'applique très peu. Le nombre de candidats est également réduit en tenant compte des intervalles de valeurs sur lesquels les attributs sont définis : le problème est que ceux-ci sont rarement disponibles, et qu'il est coûteux de les rechercher pour chaque attribut. Ajoutons qu'aucune implantation de cette méthode n'a été présentée à notre connaissance.

2.2.2 Utilisation de résumés des données

Plus récemment, un travail a été présenté dans [DJMS02] pour la compréhension de la structure des bases. Seule la découverte des DI unaires est abordée ; nous en donnons le principe général sans entrer dans les détails.

La première étape est de calculer certaines statistiques sur le contenu de la projection des attributs de la base de départ. Elles proviennent de plusieurs échantillons sur les données afin de réduire l'erreur commise par l'approximation. Ces statistiques constituent alors des résumés, à partir desquels des taux de ressemblances sont calculés entre les projections des attributs ; chaque DI unaire potentielle est alors estimée satisfaite ou non.

L'objectif visé est ainsi différent du nôtre, puisque le résultat découvert n'est pas exact. Mais cette approximation permet de traiter des bases de très grande taille, composées de plusieurs milliers de tables, d'attributs et de tuples. Des expérimentations très détaillées sont proposées, principalement afin d'estimer le nombre d'échantillons à considérer pour obtenir des taux erreurs intéressants.

2.3 Découverte des DI n-aires

2.3.1 Découverte spécifique des clés étrangères

Une méthode dédiée à la découverte des clés étrangères est proposée dans [KA96]. Après avoir trouvé les clés des relations par une adaptation de [BB95] (qui considère également les DF), les auteurs recherchent les DI pour détecter celles dont la partie droite est une clé, et qui définissent donc des clés étrangères. Ils commencent par découvrir les clés étrangères unaires en appliquant la méthode donnée dans [BB95] et décrite plus haut. Les clés étrangères n-aires sont déduites des unaires comme suit ($\forall k = 1..n, B_k$ est une clé de S) :

$$\mathbf{d} \models R[A_1...A_n] \subseteq S[B_1...B_n] \iff \forall k = 1, \ldots, n \; R[A_k] \subseteq S[B_k]$$

Bien qu'ils n'énoncent pas explicitement cette propriété, ils l'utilisent alors qu'elle s'avère trivialement fausse, comme le montre le contre-exemple suivant.

Exemple 7 Soient r et s respectivement deux relations sur les schémas $R = AB$ et $S = CD$, représentées dans la figure 2.1. Notons que C et D sont bien des clés de la relation s.

r

A	B
1	b
2	b
2	a

s

C	D
1	a
2	b
3	c

FIG. 2.1 – Une base de données $\{r, s\}$

On a bien $\{r, s\} \models \{A \subseteq C, B \subseteq D\}$, mais par contre $\{r, s\} \not\models \{AB \subseteq CD\}$.

On peut tout de même retenir le principe de réduire le nombre potentiel d'expressions de DI à tester si l'on se limite à la recherche des clés étrangères et dans la mesure où l'on connaît les clés.

2.3.2 Utilisation du langage naturel

Dans [ABDT95] une méthode informelle pour la découverte de contraintes dans les bases de données est proposée, incluant la découverte des clés étrangères. En entrée de la méthode, une description en langage naturel des données de la base est fournie par un utilisateur. Une analyse linguistique est réalisée sur cette description, des contraintes candidates sont déduites, et testées sur des échantillons de données fournis par l'utilisateur. D'autres contraintes candidates sont déduites des noms des attributs, ou encore du schéma entité/associations représentant les données (s'il est disponible). Les contraintes finalement vérifiées sur les échantillons sont validées par l'utilisateur au travers d'une interface graphique.

La méthode proposée diffère de la notre dans son objectif, car elle ne découvre pas une couverture exhaustive des dépendances d'inclusion. Les contraintes découvertes ne sont pas déduites des données, il n'est donc pas certain qu'elles soient satisfaites dans la base.

2.3.3 Utilisation de la navigation logique

Afin de découvrir un sous ensemble des DI satisfaites, on trouve dans [LPT99, LPT02] un travail exploitant les charges de travail sur la base pour générer des DI candidates. À partir des jointures effectuées sur la base, les auteurs repèrent les "attributs dupliqués", et construisent ainsi des DI candidates qui sont ensuite testées sur la base.

Cette approche n'est pas exhaustive ; elle constitue toutefois une chance de découvrir des DI intéressantes dans des bases de très grande taille. L'idée de rechercher dans les charges de travail peut être reprise en complément de l'approche exhaustive que nous proposons dans ce rapport, pour aider à sélectionner automatiquement des DI intéressantes parmi toutes celles découvertes.

2.3.4 Bordures d'une théorie

Nous verrons que la découverte des DI peut s'inscrire dans le cadre plus général de la découverte d'une propriété monotone dans un espace partiellement ordonné, étudié notamment dans [MT97, GKMT97, CW01]. Nous ne faisons qu'esquisser ce cadre dans ce paragraphe ; il sera instancié et adapté pour notre problème dans le chapitre suivant.

Cadre général

D'une manière générale le problème est de découvrir *une théorie* parmi un ensemble de motifs exprimés dans un langage $\mathcal{L}$, c'est-à-dire ceux qui sont "intéressants" relativement à une propriété q donnée. Les motifs sont ordonnés par une relation de spécialisation/généralisation, et la propriété q est anti-monotone relativement à cet ordre : si un motif est intéressant (appartient à la théorie recherchée), alors tous ceux qui le généralisent sont intéressants. La découverte des motifs fréquents dans une base de transactions est l'exemple le plus connu de problème pouvant s'inscrire dans ce cadre : les motifs sont les ensembles d'items, la relation d'ordre correspond à la relation d'inclusion ensembliste, et un motif est intéressant s'il apparaît suffisamment fréquemment dans la base.

La relation d'ordre permet de représenter tout ensemble S de motifs par sa *bordure positive* $\mathcal{B}d^+(S)$ et sa *bordure négative* $\mathcal{B}d^-(I)$. Il s'agit respectivement des motifs inclus dans S les plus spécialisés, et des motifs non inclus dans S les plus généraux. Lorsque S est la théorie que l'on cherche à découvrir, les bordures positives et négatives de S fournissent donc des représentations alternatives de la solution au problème de fouille de données.

Des algorithmes ont été proposés pour parcourir de tels espaces de recherche, et découvrir principalement la bordure positive des motifs intéressants.

Les parcours par niveaux

Les algorithmes par niveaux [AS94, MT97] parcourent l'espace des motifs des plus généraux aux plus spécialisés. L'ensemble des motifs de taille k constitue le *niveau* k de l'espace de recherche. À chaque niveau, les motifs non éliminés sont testés et ceux qui ne satisfont pas la propriété permettent d'élaguer tous les candidats qui les spécialisent, par anti-monotonie. Dans [MT97], les auteurs citent la découverte des DI comme un exemple d'application de leurs travaux ; toutefois, la proposition reste théorique, et plusieurs points cruciaux ne sont pas abordés, comme la découverte des DI unaires et la génération des candidats. A fortiori, aucune implantation d'une telle méthode n'est proposée.

Les algorithmes optimistes

Une approche par niveaux nécessite, pour découvrir un motif intéressant, d'avoir découvert tous ceux qui le généralisent, ce qui rend impossible la découverte de motifs intéressants de grande taille. Un algorithme optimiste utilise un mode de parcours lui permettant d'atteindre justement ceux-ci en supposant que tous ceux qui le généralisent sont intéressants, puis en corrigeant éventuellement cette hypothèse par la suite. Par exemple, dans [LK98, GZ01] pour la découverte des motifs fréquents, des motifs candidats de grande taille sont générés à partir d'ensemble de motifs intéressants de petite taille. Dans [GZ01], cette génération est réalisée en utilisant la notion d'hyperclique maximale dans un hypergraphe régulier.

Ce principe a été appliqué dans [KR03] pour la découverte des DI de grande taille. Le principe est de découvrir dans un premier temps toutes les DI unaires, puis les DI de taille 2 qui sont satisfaites. A partir des DI de taille 2, les DI les plus grandes possibles sont générées, sans parcourir l'espace intermédiaire au préalable. Le problème de la génération des grands candidats à partir d'un ensemble de DI satisfaites à un niveau donné est ramené à celui du calcul des hypercliques maximales dans un hypergraphe régulier. Ces grandes DI sont alors testées ; celles qui sont satisfaites sont ajoutées au résultat. Si certaines sont fausses, alors les DI de taille 3 sont à leur tour testées sur la base. A partir de celles qui sont satisfaites, des grandes DI sont à nouveau générées puis testées, et le processus recommence.

Notre proposition diffère de celle-ci sur plusieurs points. D'abord, nous inscrivons clairement la découverte des DI dans le cadre de la découverte de motifs satisfaisant une propriété anti-monotone. Ensuite, nous justifions l'utilisation d'une approche optimiste par une propriété du modèle relationnel. Nous verrons aussi que les grandes DI candidates sont générées dans notre approche à partir d'un ensemble de DI non satisfaites, en utilisant la notion de transversaux minimaux d'un hypergraphe. Enfin, lorsqu'une grande DI ainsi générée se révèle fausse, nous proposons pour orienter la suite du parcours de tenir compte de la mesure d'erreur de cette grande DI, et d'estimer ainsi son éloignement de la bordure à découvrir.

L'algorithme "Dualize and advance"

Dans [GKMT97], l'algorithme "Dualize and Advance" est proposé pour parcourir l'espace de recherche. Dans leur approche, la bordure positive en construction est toujours un sous-ensemble de la bordure positive à découvrir. À chaque itération, la bordure négative qui correspond à ce sous-ensemble est générée. Les éléments de cette bordure négative sont testés sur les données jusqu'à en trouver un qui soit intéressant ; celui-ci est alors le point de départ d'une recherche en profondeur d'un motif qui le spécialise appartenant à la bordure positive. Une fois trouvé, ce nouveau motif est ajouté à la bordure positive en construction, utilisée pour une nouvelle itération.

Notre algorithme diffère de celui-ci dans le mode de parcours. À chaque itération, les corrections sont apportées aux connaissances courantes en testant des éléments de la bordure positive suspectée. Les contre-exemples sont donc des motifs non-intéressants, ce qui nous permet d'exploiter une mesure d'erreur permettant d'estimer la distance à la bordure positive à découvrir. L'orientation de la suite du parcours est décidée à partir de cette estimation. En outre, nous initialisons le parcours par une exploration par niveaux des candidats les plus généraux, contrairement à ce qui est proposé dans [GKMT97], où l'ensemble vide est pris comme point de départ.

Chapitre 3

Découverte des dépendances d'inclusion unaires

Sommaire

3.1 Le problème

Nous traitons dans ce chapitre un sous-problème du problème de la découverte des DI satisfaites dans une base de données, qui est le problème de la découverte des DI unaires satisfaites. Celui-ci se formule comme suit :

"Étant donnée une base de données **d**, trouver une couverture des DI unaires satisfaites dans **d**".

Il s'agit d'un problème quadratique dans le nombre d'attributs puisque chaque DI unaire candidate correspond à un couple d'attributs. Par exemple, l'algorithme 1 découvre toutes les DI unaires satisfaites dans une base de données **d**, en utilisant une démarche naïve de type "générer/tester" ; les DI candidates sont générées, puis testées sur la base.

Un tel algorithme conduit à effectuer $N(N-1)$ tests sur la base, où N est le nombre total d'attributs. Ce nombre peut parfois être réduit à partir de certaines connaissances sur les données, comme le domaine de définition des attributs ou encore le nombre de valeurs distinctes dans les projections. En effet, on a la propriété suivante : si $\{r_1, r_2\} \models R_1[X] \subseteq R_2[Y] \Rightarrow |\pi_X(r_1)| \leq |\pi_Y(r_2)|$. Dans le cas favorable où toutes les cardinalités des

Algorithme 1 Algorithme de type "générer/tester" pour la découverte des DI unaires

Entrée: une base de données $\mathbf{d} = \{r_1, ..., r_n\}$ sur un schéma $\mathbf{R} = \{R_1, ..., R_n\}$;
Sortie: I_1 l'ensemble des DI unaires satisfaites dans $\mathbf{d}$.

Soit $U = \cup_{i=1,\dots,n} R_i$;
for all $A \in U$ **do**
    for all $B \in U \setminus A$ **do**
        if $\mathbf{d} \models A \subseteq B$ **then**
            $I_1 = I_1 \cup \{A \subseteq B\}$;
        end if
    end for
end for
Retourner I_1.

projections sont connues et toutes différentes, le nombre de tests sur la base est alors de $N(N-1)/2$.

Nous allons montrer dans ce chapitre comment ce problème peut être traité grâce à une ré-organisation des données, donnant lieu à une re-formulation du problème et à un algorithme efficace. Nous évaluerons également, à partir du nombre d'entrées/sorties opérées sur les données, le gain théorique d'efficacité de notre algorithme par rapport à l'algorithme de type "générer/tester". Ce gain sera finalement confirmé par les expérimentations réalisées sur des bases de données synthétiques.

3.2 Ré-organisation des données

L'idée est de réorganiser les données sous la forme d'une relation binaire entre toutes les valeurs prises dans une base de données et l'ensemble des attributs du schéma de cette base. Un couple (attribut,valeur) appartiendra à la relation binaire si et seulement si l'attribut prend cette valeur dans un des tuples de la base. Nous définissons dans un premier temps ce que nous appellerons un contexte d'extraction associé à une base de données. Puisque une DI ne peut pas être satisfaite entre deux attributs définis sur deux domaines différents, un contexte d'extraction pour chaque domaine de la base devrait normalement être défini. Toutefois le domaine de définition des attributs n'est souvent pas connu ; l'information généralement disponible concerne *le type* des éléments de chaque attribut (integer, real,...). C'est pourquoi dans la suite nous considérerons les différents types de données présents dans la base, et non pas les domaines de définition des attributs.

Soit une base de données $\mathbf{d}$ sur un schéma $\mathbf{R}$, pour chaque type de données t dans $\mathbf{d}$, on définit un *contexte d'extraction*, noté $\mathbb{D}_t(\mathbf{d}) = (\mathbb{V}, \mathbb{U}, \mathbb{B})$, par :

- $\mathbb{U} = \{R.A \mid A \in R, R \in \mathbf{R}, A \text{ est défini sur le type } t\}$. $\mathbb{U}$ est l'ensemble des attributs de type t ;

– $\mathbb{V} = \{v \in \pi_A(r) \mid R.A \in \mathbb{U}, r \in \mathbf{d}, r$ défini sur $R\}$. $\mathbb{V}$ est l'ensemble des valeurs prises par les attributs dans les relations ;
– $\mathbb{B} \subseteq \mathbb{V} \times \mathbb{U}$ est une relation binaire définie par :
$(v, R.A) \in \mathbb{B} \iff v \in \pi_A(r)$, où $r \in \mathbf{d}$ et r définie sur R.

Le concept de contexte d'extraction défini par un triplet provient du domaine de *l'analyse formelle de concepts* [GW99], cadre pouvant être utilisé dans le domaine de la fouille de données et notamment pour la découverte des motifs fréquents.

Exemple 8 Considérons la base de données **movies** (figure 3 page 8). Les attributs sont de 4 types : integer, real, date, string. La figure 3.1 représente le contexte d'extraction associé au type string.

FIG. 3.1 – Contexte d'extraction associé à la base **movies** et au type string.

$\mathbb{V}$	$\mathbb{U}$
Mademoiselle Fifi	$F\ H$
Young at heart	$B\ H$
L'exorciste	H
Thérèse	$F\ H$
Les amours de Carmen	$B\ H$
Le voyage	$B\ H$
Quatre soeurs	$F\ H$
Quatres soeurs	F
Le cercle du mariage	$F\ H$
Une heure avec toi	$B\ H$
Gilda	$F\ H$
Cléopatre	$B\ F\ H$
Des soeurs courageuses	$B\ H$
Casablanca	$F\ H$
L'Oeuvre au noir	$B\ H$
Le manitou	H
Le nom de la rose	$F\ H$

3.3 Caractérisation et découverte des DI unaires

A partir de la nouvelle représentation des données, une caractérisation intéressante des DI unaires satisfaites peut-être donnée. Supposons que la relation binaire soit structurée, comme dans l'exemple de la figure 3.1, sous la forme d'une base de données de transactions.

De manière informelle, si toutes les valeurs d'un attribut A sont dans un attribut B, alors par construction B apparaîtra dans toutes les lignes contenant A dans la relation binaire. La proposition suivante formalise, de manière constructive, ce principe.

Proposition 1 *Soit une base de données* **d**, *t un type de données dans* **d** *et le contexte d'extraction associé* $\mathbb{D}_t(d) = (\mathbb{V}, \mathbb{U}, \mathbb{B})$,

$$d \models A \subseteq B \Longleftrightarrow B \in \bigcap_{v \in \mathbb{V} | (v,A) \in \mathbb{B}} \{C \in \mathbb{U} \mid (v, C) \in \mathbb{B}\}$$

où $A, B \in \mathbb{U}$.

Preuve *Soit* $A \in R$, $B \in S$ *tels que* $d \models A \subseteq B$.
$\Longleftrightarrow \forall v \in \pi_A(r), \exists u \in s$ *tel que* $u[B] = v$
$\Longleftrightarrow \forall v \in \mathbb{V}$ *tel que* $(v, A) \in \mathbb{B}$, *on a* $(v, B) \in \mathbb{B}$ □

Remarque : Si le contexte d'extraction est vu comme une base de données de transaction, où les attributs correspondent aux *items* et les valeurs aux *transactions*, les DI unaires correspondent alors aux règles d'association exactes (avec une confiance à 100%) entre les singletons d'items, pouvant être déduites des motifs fréquents de taille 1 et 2 apparaissant au moins une fois dans la base.

La découverte des DI unaires peut ainsi être réalisée en seulement un parcours de chaque contexte d'extraction. L'algorithme 2 réalise cette tâche pour un contexte d'extraction associé à une base de données et un type quelconque. Pour un attribut A, on note $rhs(A)$ l'ensemble des B tels que $A \subseteq B$ est satisfaite.

Exemple 9 Considérons le type string (cf figure 3.1) pour la base **movies**. Pour chaque attribut, l'initialisation des parties droites est : $rhs(B) = rhs(F) = rhs(H) = \{B, F, H\}$.

On considère ensuite l'ensemble des attributs dans la première ligne de la relation binaire : $l_1 = \{F, H\}$. Pour chaque attribut A dans l_1, on procède à l'intersection : $rhs(A) = rhs(A) \cap l_1$, et on obtient ainsi : $rhs(B) = \{B, F, H\}$, $rhs(F) = \{F, H\}$, $rhs(H) = \{F, H\}$.

Ces opérations sont répétées pour chaque valeur de la relation binaire. Finalement, après un parcours, le résultat est : $rhs(B) = \{B, H\}$, $rhs(F) = \{F\}$, $rhs(H) = \{H\}$.

A partir de ces ensembles, on déduit l'ensemble des DI unaires entre des attributs de type string : $\{B \subseteq H\}$.

La même opération doit être répétée pour chaque type. On obtient alors l'ensemble des DI unaires satisfaites dans la base de données **d** : $I_1 = \{A \subseteq G, B \subseteq H, C \subseteq I, E \subseteq G\}$.

Algorithme 2 Découverte exhaustive des DI unaires non triviales

Entrée: un contexte d'extraction $(\mathbb{V}, \mathbb{U}, \mathbb{B})$ associé à une base **d** et un type de données t.

Sortie: I_1 l'ensemble des DI unaires satisfaites dans **d** entre des attributs de type t.

for all $A \in \mathbb{U}$ **do**
  $rhs(A) = \mathbb{U}$;
end for
for all $v \in \mathbb{V}$ **do**
  for all $A \mid (v, A) \in \mathbb{B}$ **do**
    $rhs(A) = rhs(A) \cap \{B \mid (v, B) \in \mathbb{B}\}$;
  end for
end for
for all $A \in \mathbb{U}$ **do**
  for all $B \in rhs(A) \setminus \{A\}$ **do**
    $I_1 = I_1 \cup \{A \subseteq B\}$;
  end for
end for
Retourner I_1.

3.4 Nombre de passages sur les données

Nous considérons dans ce paragraphe que le contexte d'extraction peut être maintenu en mémoire principale, ce qui sera justifié dans les descriptions d'implantations du chapitre 6 page 65. Ainsi, seule l'étape de construction des contextes réalise des accès au disque dans la méthode que nous proposons.

Considérons, sans perte de généralité une base de données sur un schéma composé de N attributs du même type. On suppose également que le coût d'un passage sur une relation est le même dans toute la base, et correspond donc au même nombre de blocs chargés en mémoire.

Nous avons vu qu'une approche de type "générer/tester" (algoritme 1 page 32), dans laquelle la cardinalité de la projection des attributs est connue, conduit au mieux à effectuer $N(N-1)/2$ tests sur les données. Chaque test consiste en un passage sur chacune des parties gauches et droites de la DI testée, soit $N(N-1)$ balayages de relations.

Dans notre proposition, afin de construire le contexte d'extraction, les attributs sont d'abord parcourus pour récupérer les valeurs, ce qui conduit à N passages sur les relations. Ensuite, les curseurs construits sont parcourus pour insérer les valeurs dans la relation binaire du contexte d'extraction, donc N parcours supplémentaires, soit $2N$ passages sur les données. Si les relations binaires tiennent en mémoire, notre approche n'induit pas d'accès supplémentaires aux données sur le disque.

En pratique, notre méthode induit donc environ $N/2$ fois moins de passages sur les données que pour une approche de type "générer/tester". Ce gain sera confirmé par les résultats expérimentaux présentés dans le chapitre 6 page 65.

Chapitre 4

Découverte des dépendances d'inclusion n-aires

Sommaire

4.1 Définition et propriétés du cadre de travail

Dans la suite, nous considérons connues les DI unaires satisfaites. Notre problème peut donc être reformulé comme suit :

"Soit une base de données **d** et les DI unaires satisfaites dans **d**, trouver une couverture de toutes les DI satisfaites dans **d**."

4.1.1 Définition de l'espace de recherche

Nous faisons l'hypothèse qu'un ordre strict est disponible sur les attributs de la base ; à un renommage éventuel près, il est toujours possible par exemple d'utiliser l'ordre lexicographique. De plus, dans la suite du mémoire, nous ne considérerons pas la découverte des DI pseudo-triviales : il s'agit d'un cas particulier de DI dont la prise en compte alourdit considérablement les résultats et les algorithmes. Ces DI particulières sont toutefois prises en compte dans l'implantation de l'algorithme.

Nous définissons tout d'abord, pour un schéma de base de données, l'*espace de recherche des DI associé*. Intuitivement, cet espace de recherche exploite la deuxième règle d'inférence des DI (permutation) afin de restreindre l'espace de recherche à "une seule permutation" de chaque expression de DI. La forme retenue est celle qui semble la plus naturelle, la partie gauche est ordonnée. En cas d'égalité à gauche, c'est l'ordre à droite qui est pris en compte.

Définition 20 (Espace de recherche) Soit $\mathbf{R}$ un schéma de base de données. On appelle C l'*espace de recherche des DI associé à $\boldsymbol{R}$* l'ensemble défini par :

$$C = \{i = R[A_1...A_n] \subseteq S[B_1...B_n] \mid R, S \in \mathbf{R} \mid \forall k, l \mid 1 \leq k < l \leq n,$$
$$(A_k < A_l) \vee (A_k = A_l \wedge B_k < B_l) \text{ et } i \text{ non triviale } \}.$$

Notons que les DI triviales, qui sont toujours satisfaites, sont explicitement exclues de cet espace de recherche. Les DI redondantes quant à elles sont exclues, puisque le cas $(A_k = A_l) \wedge (B_k = B_l)$ n'est pas prévu par la définition (on rappelle que les doublons sont autorisés dans les expressions de DI).

Nous verrons dans ce chapitre que les choix faits pour restreindre l'espace de recherche ne remettent pas en cause la qualité de la couverture des DI qui sera découverte, celle-ci permettant d'inférer efficacement *toutes* les DI satisfaites dans une base de données.

4.1.2 Taille de l'espace de recherche

Soit $\mathbf{R} = \{R_1, ..., R_n\}$ un schéma de bases de données, tel que toutes les relations de $\mathbf{R}$ possèdent chacune N attributs. Soit C l'espace de recherche associé à $\mathbf{R}$. Soit dans un premier temps l'ensemble des DI allant de R_i vers R_j, avec $R_i, R_j \in \mathbf{R}$. Considérons une DI $R_i[A_1...A_k] \subseteq R_j[B_1...B_k]$ élément de C ; celle-ci peut-être vue comme la concaténation des DI unaires $R_i[A_1] \subseteq R_j[B_1], ..., R_i[A_k] \subseteq R_j[B_k]$. Chaque ensemble de n DI unaires distinctes peut ainsi former une DI n-aire dans C, unique puisque l'ordre d'apparition des attributs est fixé. Les DI unaires doivent être distinctes, sinon la DI n-aire correspondante sera redondante.

Le nombre de DI unaires potentielles, définies de R_i vers R_j, est N^2. Elles permettent donc de "former" 2^{N^2} DI définies de R_i vers R_j. Le même raisonnement vaut pour les n^2 couples (R_i, R_j) de schémas de relations dans $\mathbf{d}$, et on obtient :

$$|C| = n^2 \times 2^{N^2}$$

L'exploration exhaustive de C est donc en général impossible. La section suivante montre comment il peut être structuré par une relation d'ordre partiel, conférant un caractère anti-monotone à la propriété de satisfaction des DI.

4.1.3 Structuration de l'espace de recherche

Une relation de spécialisation/généralisation peut être définie sur les DI, établissant ainsi un ordre partiel dans l'espace de recherche.

Définition 21 (Ordre partiel sur les DI) Soient $i = R[X] \subseteq S[Y]$ et $j = R[X'] \subseteq S[Y']$ deux expressions de DI définies sur les mêmes schémas de relations R et S . on dit que i *généralise* j (ou que j *spécialise* i), et on note $i \preceq j$, si $X' =< A_1, ..., A_n >$, $Y' =< B_1, ..., B_n >$, et il existe un ensemble d'indices $k_1 < ... < k_l \in \{1, ..., n\}$ avec $l \leq n$ tels que $X =< A_{k_1}, ..., A_{k_l} >$, $Y =< B_{k_1}, ..., B_{k_l} >$

Exemple 10 Par exemple, on a $R[AC] \subseteq S[GI] \preceq R[ABC] \subseteq S[GHI]$. Par contre, $R[AC] \subseteq S[GH] \npreceq R[ABC] \subseteq S[GHI]$.

Par extension, on note $i \prec j$ pour $i \preceq j$ et $i \neq j$.

L'ordre ainsi défini et la propriété de satisfaction des DI sont reliés par une propriété d'anti-monotonie, provenant directement de la deuxième règle d'inférence des DI (projection) :

Propriété 1 *Soit* $\mathbf{d}$ *une base de données,* C *l'espace de recherche associé, et* i *et* j *deux expressions de DI dans* C *telles que* $i \preceq j$*. On a :*
- $\mathbf{d} \not\models i \Rightarrow \mathbf{d} \not\models j$ *ou encore*
- $\mathbf{d} \models j \Rightarrow \mathbf{d} \models i$.

Bien qu'elles soient équivalentes, nous donnons ces deux expressions pour les mettre en évidence : elles seront chacune utilisées par la suite.

Exemple 11 Dans l'exemple de la base **movies** (figure 3 page 8), la DI $R[ABC] \subseteq S[GHI]$ est satisfaite. Puisque $R[AC] \subseteq S[GI] \preceq R[ABC] \subseteq S[GHI]$, on a nécessairement

movies $\models R[AC] \subseteq S[GI]$. En revanche, puisque **movies** $\not\models R[AF] \subseteq S[GH]$, on a forcément **movies** $\not\models R[ACF] \subseteq S[GIH]$.

Cette propriété est fondamentale dans la mise en place de méthodes d'exploration de l'espace de recherche des DI, et permet d'éviter de nombreux tests sur la base : tout ensemble de DI satisfaites permet de déduire a priori (sans les tester) d'autres DI satisfaites et tout ensemble de DI non satisfaites permet de déduire a priori des DI non satisfaites.

À partir de l'ordre '$\preceq$', on définit la *bordure positive* et la *bordure négative* d'un ensemble d'expressions de DI.

La bordure positive d'un ensemble de DI est l'ensemble des éléments les plus spécialisés qu'il contient :

Définition 22 (Bordure positive) Soit **d** une base de données, C l'espace de recherche des DI associé et I un sous-ensemble de C. On appelle bordure positive de I, notée $\mathcal{B}d^+(I)$, l'ensemble défini par : $\mathcal{B}d^+(I) = \{i \in I \mid \nexists j \in I, i \prec j\}$.

La bordure négative d'un ensemble de DI est l'ensemble des éléments les plus généraux qu'il ne contient pas :

Définition 23 (Bordure négative) Soit **d** une base de données, C l'espace de recherche des DI associé et I un sous-ensemble de C. On appelle bordure négative de I, notée $\mathcal{B}d^-(I)$, l'ensemble défini par : $\mathcal{B}d^-(I) = \{i \in C \setminus I \mid \nexists j \in C \setminus I, j \prec i\}$.

Exemple 12 Considérons la base de données **movies** (figure 3 page 8). La figure 4.1 représente le graphe de l'ordre '$\preceq$' pour l'ensemble des DI candidates. Les DI satisfaites et non satisfaites sont reportées, ainsi que les bordures négatives et positives des DI satisfaites. Afin d'alléger les notations, les expressions de DI ne sont pas préfixées par le nom de la relation. Pour plus de clarté, nous limitons la représentation de l'espace de recherche à l'ensemble des DI qui ne sont pas contredites par une DI unaire fausse par anti-monotonie. Par exemple, la DI unaire $A \subseteq C$ n'est pas représentée puisqu'elle n'est pas satisfaite, et ainsi toutes les DI qui la spécialisent n'apparaissent pas. Naturellement, *toutes les DI unaires non satisfaites font partie de la bordure négative* des DI satisfaites.

Soit $\mathbf{d} = \{r_1, ..., r_n\}$ une base de données sur un schéma $\mathbf{R} = \{R_1, ..., R_n\}$. Dans toute la suite, soit C l'espace de recherche associé à **d**, et $I \subseteq C$ l'ensemble des DI satisfaites de C dans **d**. On notera en outre C_k les DI de C de taille k, et I_k les DI de I de taille k.

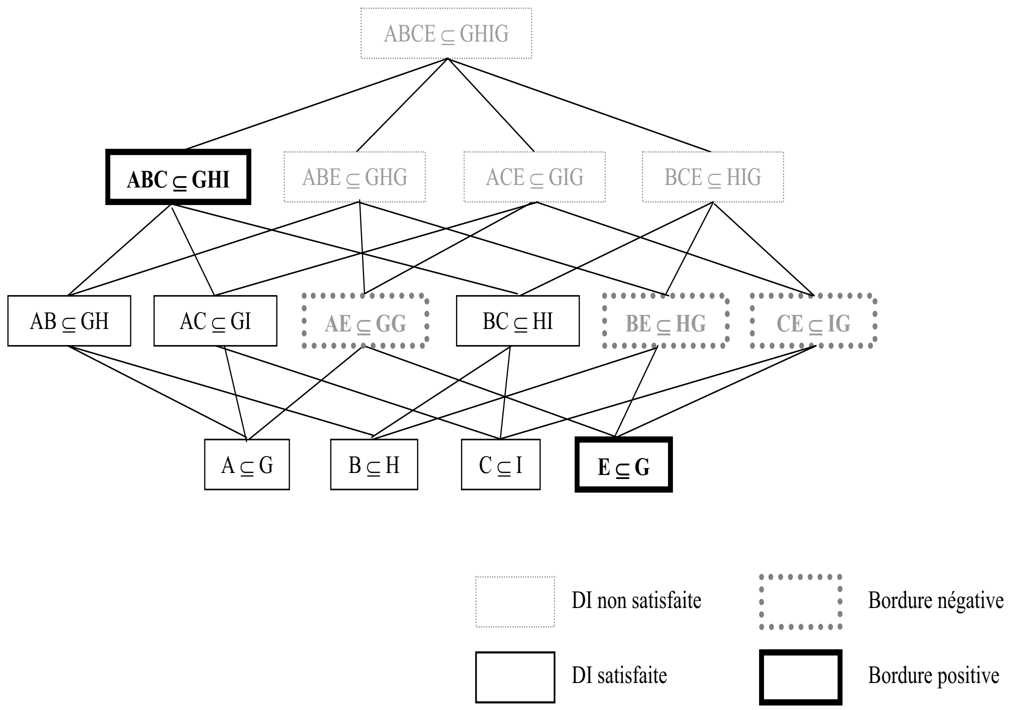

FIG. 4.1 – Espace de recherche des DI de la base de données **movies**

4.1.4 La bordure positive : une solution au problème

Le problème à résoudre est la recherche d'une couverture de l'ensemble des DI satisfaites dans une base de données. L'espace de recherche C défini plus haut est lui-même une réduction de l'espace de recherche exhaustif qui serait composé de *toutes* les expressions de DI. Nous montrons ici que cette réduction n'entraîne pas de perte de connaissance.

Dans un premier temps, nous montrons que la bordure positive $\mathcal{B}d^+(I)$ des DI satisfaites parmi celles de C est une solution juste et complète au problème.

Proposition 2 *Soit i une expression de DI sur $\boldsymbol{R}$. On a*

$$\boldsymbol{d} \models i \Leftrightarrow \mathcal{B}d^+(I) \models i$$

Preuve *Supposons $\mathcal{B}d^+(I) \models i$. Puisque, $\mathcal{B}d^+(I) \subseteq I$ et $\boldsymbol{d} \models I$, on a $\boldsymbol{d} \models \mathcal{B}d^+(I)$ et ainsi $\boldsymbol{d} \models i$.*

Supposons $\boldsymbol{d} \models i$. Si i est triviale, alors $\mathcal{B}d^+(I) \models i$ est toujours vrai (règle de réflexivité). Si i est redondante, alors on peut l'écrire sous la forme $XU \subseteq YV$, avec $U \subseteq V \preceq X \subseteq Y$ et $X \subseteq Y$ non redondante. On a $X \subseteq Y \models i$ (règle de redondance), et il suffit alors de montrer $\mathcal{B}d^+(I) \models X \subseteq Y$. On pose alors dans ce cas $i = X \subseteq Y$.

Soit i' la DI (unique) appartenant à C, et obtenue par permutation de i (et donc $i \models i'$ et $i' \models i$ par la règle de permutation). Puisque $i \models i'$, on a $\mathbf{d} \models i'$ et $i' \in I$. D'après la définition 22 (bordure positive), $\exists j \in \mathcal{B}d^+(I) \mid i' \preceq j$, et donc $\mathcal{B}d^+(I) \models i'$ par projection, ce qui conduit à $\mathcal{B}d^+(I) \models i$. $\square$

Les algorithmes que nous donnerons dans la suite auront pour objectif de découvrir cette bordure positive. A partir de cette connaissance, il suffira ainsi à l'utilisateur final (administrateur, analyste) de tester si $\mathcal{B}d^+(I) \models i$ pour savoir si $\mathbf{d} \models i$, sans accès supplémentaire aux données. Une question se pose toutefois : sachant que le problème de l'inférence des DI est $PSPACE-COMPLET$ dans le cas général (voir les préliminaires en section 1.2.4), quelle est la complexité du test $\mathcal{B}d^+(I) \models i$? La proposition suivante traite du cas particulier de l'inférence des DI à partir de la bordure positive.

Proposition 3 *Soit i une expression de DI quelconque sur $\boldsymbol{R}$. Déterminer si $\mathcal{B}d^+(I) \models i$ est un problème linéaire dans la taille de $\mathcal{B}d^+(I)$.*

Preuve *Considérons l'algorithme 3 page 42 ; celui-ci prend en entrée $\mathcal{B}d^+(I)$ et i sur $\boldsymbol{R}$, puis teste si $\mathcal{B}d^+(I) \models i$.*

Algorithme 3 Inférence d'une DI depuis *bdplus*

Entrée: $\mathcal{B}d^+(I)$, une DI i sur R.
Sortie: Vrai si $\mathcal{B}d^+(I) \models i$, faux sinon.

```
1: if i est triviale then
2:    Retourner VRAI ;
3: end if
4: Calculer i', permutation ordonnée à gauche puis à droite de i ;
5: Enlever les redondances de i'.
6: for all j ∈ Bd+(I) do
7:    if i' ⪯ j then
8:       Retourner VRAI ;
9:    end if
10: end for
11: Retourner FAUX.
```

Cet algorithme reprend le principe de la deuxième partie de la preuve de la proposition 2. Les lignes 4 et 5 correspondent au calcul du "représentant" i' de i dans C ; il faut pour le calculer ordonner la partie gauche de i. En cas de doublon à gauche, on ordonne selon la partie droite. Une fois fait, il est alors simple de détecter et supprimer les redondances en un parcours de la DI obtenue. L'exemple 13 illustre l'algorithme 3.

La complexité de l'algorithme est linéaire dans la taille de la bordure positive.

Il reste à montrer que l'algorithme 3 est juste. Précisons d'abord que $i' \in C$ par construction, et que $i' \models i$ par les règles de permutation et de redondance, et que $i \models i'$ par les règles de permutation et de projection.

Si l'algorithme retourne $VRAI$, alors $\mathcal{B}d^+(I) \models i'$ et donc $\mathcal{B}d^+(I) \models i$. Si la réponse est $FAUX$, alors $\nexists j \in \mathcal{B}d^+(I) \mid i' \preceq j$, donc $i' \notin I$ d'après la définition 22 (bordure positive) et ainsi $\mathbf{d} \not\models i'$. Puisque $i \models i'$, on a forcément $\mathbf{d} \not\models i$ par l'absurde. □

Exemple 13 Considérons l'exemple de la figure 4.1. Supposons que seule la bordure positive est connue sur ce graphe, soit $\mathcal{B}d^+(I) = \{E \subseteq G, ABC \subseteq GHI\}$, cherchons à savoir si la DI $CCA \subseteq IIG$ est inférée par $\mathcal{B}d^+(I)$. On ordonne alors selon la partie gauche puis la partie droite, pour obtenir la DI $ACC \subseteq GII$. En supprimant les redondances, on obtient $AC \subseteq GI$. En un parcours de $\mathcal{B}d^+(I)$, on trouve que $AC \subseteq GI \preceq ABC \subseteq GHI$, donc $\mathcal{B}d^+(I) \models CCA \subseteq IIG$.

De manière informelle, le passage d'une inférence $PSPACE-$*complet* dans le cas général à une inférence linéaire à partir de $\mathcal{B}d^+(I)$ s'explique par le fait que la règle de transitivité n'est pas utilisée dans l'algorithme 3. Ce qui est permis par l'hypothèse que l'ensemble $I \subseteq C$ est l'ensemble de toutes les DI de C satisfaites dans $\mathbf{d}$, donc stable par transitivité.

4.1.5 Décomposition du problème

Nous donnons une propriété de l'espace de recherche défini plus haut, permettant la décomposition du problème de la découverte des DI en plusieurs tâches indépendantes. On peut écrire en effet, si on note $I_{R_1 \to R_2}$ la restriction de I à ses éléments définis de R_1 vers R_2 :

Propriété 2

$$\mathcal{B}d^+(I) = \bigcup_{(R_1,R_2) \in \boldsymbol{R}^2} \mathcal{B}d^+(I_{R_1 \to R_2})$$

Outre la parallélisation naturelle des algorithmes de découverte induite par cette propriété, cette décomposition permet également de diminuer le nombre de DI stockées à chaque étape, améliorant les performances des manipulations telles que les tris, insertions, suppressions, générations de candidats.

4.2 Une approche par niveaux : *Mind*

Nous donnons un algorithme parcourant l'espace de recherche des DI par niveaux, des expressions les plus générales aux plus spécifiques. Un tel mode de parcours permet l'élagage à chaque niveau de l'espace de recherche restant à parcourir, dès lors que des DI non satisfaites sont découvertes. C'est pourquoi nous qualifions dans la suite une telle approche de pessimiste : aucune supposition n'est faite a priori sur la satisfaction des DI, tous les candidats non éliminés sont testés sur la base.

4.2.1 L'algorithme MIND

L'algorithme 4, appelé *Mind* pour "Mining IND", découvre la bordure positive des DI satisfaites dans une base de données. Il prend notamment en entrée la liste des DI unaires découvertes par l'approche décrite dans le chapitre précédent. Nous avons également intégré le calcul de la bordure négative qui n'ajoute pas de complexité à l'algorithme, et qui sera une connaissance importante dans la suite du mémoire.

L'algorithme réalise un parcours "par niveaux" [AS94, HKPT99, MT97] de l'espace de recherche, des DI les plus générales aux plus spécialisées. A partir des DI satisfaites de taille 1, les candidats non éliminés par anti-monotonie de taille 2 sont générés, puis testés. Ensuite, à partir des DI satisfaites au niveau 2, les DI candidates de taille 3 sont générées, et testées sur la base. Les DI candidates de niveau 4 sont alors générées. Le processus se poursuit ainsi, tant qu'il est encore possible de générer de nouveaux candidats.

Algorithme 4 *Mind* : découverte des DI n-aires

Entrée: Une base de données **d** et I_1, l'ensemble des DI unaires satisfaites dans **d**.
Sortie: La bordure positive $\mathcal{B}d^+(I)$ et la bordure négative $\mathcal{B}d^-(I)$ des DI satisfaites dans **d**.

1: $C_2 := GenNext(I_1)$;
2: $k := 2$;
3: $\mathcal{B}d^+(I) = I_1$; $\mathcal{B}d^-(I) = C_1 \setminus I_1$;
4: **while** $C_k \neq \emptyset$ **do**
5:   $I_k = \emptyset$;
6:   **for all** $i \in C_k$ **do**
7:     **if** $\mathbf{d} \models i$ **then**
8:       $I_k = I_k \cup \{i\}$;
9:     **else**
10:       $\mathcal{B}d^-(I) = \mathcal{B}d^-(I) \cup \{i\}$;
11:     **end if**
12:   **end for**
13:   $\mathcal{B}d^+(I) = \mathcal{B}d^+(I) \cup I_k \setminus \{j \in \mathcal{B}d^+(I) \mid \exists i \in I_k, j \prec i\}$;
14:   $C_{k+1} := GenNext(I_k)$;
15:   $k := k + 1$;
16: **end while**
17: **Retourner** $\mathcal{B}d^+(I)$ et $\mathcal{B}d^-(I)$.

La ligne 13 de l'algorithme 4 assure la sortie de la bordure positive $\mathcal{B}d^+(I)$. Ceci aurait tout aussi bien pu faire l'objet d'un traitement global en sortie de l'algorithme, quadratique dans le nombre de DI satisfaites dans C.

Coût de l'algorithme

Le coût d'un tel algorithme correspond au nombre de tests effectuées sur la base, multiplié par le coût d'un test de satisfaction d'une DI [MT97]. Or, le nombre de tests est égal à $|\mathcal{B}d^-(I)| + |I|$.

Exemple 14 Dans l'exemple de la figure 4.1, 7 tests sont effectués par l'algorithme *mind* (on rappelle que les DI unaires sont déjà connues) : l'ensemble des quatre DI n-aires satisfaites, plus les trois DI constituant la bordure négative.

Le coût d'un test sera vu plus en détail dans le chapitre 6 ; il est majoré par $O(n^2)$, n étant le nombre moyen de tuples dans chaque relation.

4.2.2 Génération des candidats

La génération des DI candidates de niveau $k+1$ à partir des DI satisfaites au niveau k peut être réalisée en étendant le principe classique de la génération des candidats des algorithmes par niveaux. L'algorithme 5 réalise cette tâche ; il se compose de deux phases principales : une phase de génération et une phase d'élagage, toutes les deux basées sur la propriété 1. L'archétype est AprioriGen [AS94] pour la génération des motifs fréquents lors de la découverte des règles d'associations. Dans notre cas, la difficulté est de gérer à la fois la génération des parties droites et des parties gauches correspondantes. Dans l'algorithme, on note $i.lhs$ et $i.rhs$ les séquences formant respectivement les parties gauches et droite de la DI i, et $X.rel$ le schéma de relation contenant les attributs de la séquence X.

La phase de génération (lignes 1 à 8), fournit toutes les DI candidates générées de la façon suivante : soient deux DI ayant *le même préfixe* à gauche et à droite, i.e. $R[XA] \subseteq S[YC]$ et $R[XB] \subseteq S[YD]$ avec $|X| = |Y| = k-1$ et $A < B$ (l'ordre entre les attributs C et D n'a pas d'importance). On aura un candidat de la forme : $R[XAB] \subseteq S[YCD]$.

Exemple 15 Considérons la base de données **movies** (figure 3 page 8). La génération de candidats est illustrée par le graphe de la figure 4.1. Par exemple, les deux DI $AB \subseteq GH$ et $AC \subseteq GI$ forment un candidat $ABC \subseteq GHI$.

Cette phase de génération assemble deux DI satisfaites de niveau k ayant même préfixe à gauche et à droite pour en construire une de taille $k+1$; or une DI de taille $k+1$ spécialise $k+1$ DI de taille k. La phase d'élagage (lignes 9 à 15), se charge de vérifier que toutes les DI de taille k spécialisées par le nouveau candidat sont bien satisfaites.

Exemple 16 Aucun cas d'élagage ne se présente dans notre exemple courant de la fi-

Algorithme 5 *GenNext* : Génération des DI candidates de taille $k+1$

Entrée: I_k, DI de taille k.
Sortie: C_{k+1}, ensemble des DI candidates de taille $k+1$

1: **insert into** C_{k+1}
2: **select** $p.lhs.rel$"["$p.lhs[1], p.lhs[2], ..., p.lhs[k], q.lhs[k]$"]"" $\subseteq$ " $p.rhs.rel$"["$p.rhs[1], p.rhs[2], ..., p.rhs[k], q.rhs[k]$"]"
3: **from** I_k p, I_k q
4: **where** $p.lhs.rel = q.lhs.rel$ **and** $p.rhs.rel = q.rhs.rel$
5: **and** $p.lhs[1] = q.lhs[1]$ **and** $p.rhs[1] = q.rhs[1]$
6: **and** ...
7: **and** $p.lhs[k-1] = q.lhs[k-1]$ **and** $p.rhs[k-1] = q.rhs[k-1]$
8: **and** $p.lhs[k] < q.lhs[k]$ **or** $(p.lhs[k] = q.lhs[k]$ **and** $p.rhs[k] < q.rhs[k])$
9: **for all** $c \in C_{k+1}$ **do**
10:   **for all** $i \prec c$ et $|i| = k-1$ **do**
11:     **if** $i \notin I_k$ **then**
12:       $C_{k+1} = C_{k+1} \setminus \{c\}$
13:     **end if**
14:   **end for**
15: **end for**
16: **Retourner** C_{k+1}.

gure 4.1. Toutefois, supposons que la DI $R[BE] \subseteq S[HG]$ soit satisfaite. Pour le niveau 3, la phase de génération de l'algorithme formerait la DI candidate : $R[BCE] \subseteq S[HIG]$. Mais puisque la DI $R[CE] \subseteq S[IG]$ ne serait pas satisfaite, le nouveau candidat serait supprimé par la phase d'élagage de l'algorithme.

Exclusion des doublons L'algorithme considère la définition générale des DI acceptant les doublons à gauche et à droite. Comme on l'a vu dans les préliminaires, cette définition est parfois restreinte aux DI sans doublons. Si on se place dans ce cas, alors la ligne 8 de l'algorithme 5 doit être remplacée par : "**and** $p.lhs[k] < q.lhs[k]$ **and** $p.rhs[k] \neq q.rhs[k]$".

4.3 Une approche hybride : *Zigzag*

On a vu que l'approche par niveaux conduit à tester sur la base, entre autres, l'ensemble des DI satisfaites. Ainsi, si une DI de taille N est à découvrir dans la bordure positive, il faut au minimum tester les 2^N DI satisfaites qui la généralisent, mais ne font pas partie du résultat (elles ne sont pas dans la bordure positive). Une telle approche purement pessimiste est donc systématiquement mise en échec lorsque N augmente.

Un nouveau mode de parcours de l'espace de recherche est présenté dans cette partie.

Après un parcours par niveaux (pessimiste) des candidats les plus généraux, des "sauts optimistes" dans l'espace de recherche seront effectués pour permettre d'atteindre des DI de grande taille. Ces sauts seront effectués à partir des DI *non satisfaites* découvertes jusqu'alors ; un théorème ramènera la génération des grands candidats au problème du calcul des minimaux transversaux d'un hypergraphe. Lorsque les grandes DI ainsi considérées ne seront pas satisfaites, une phase de correction des hypothèses faites sera mise en place, exploitant une mesure de l'erreur commise.

Pour simplifier la présentation des résultats, nous tiendrons compte dans cette partie de la propriété 2 page 43, qui spécifie que l'espace de recherche peut-être divisé en parties indépendantes, une pour chaque couple de relations de la base considérée. Ainsi, l'espace de recherche C est limité à l'ensemble des expressions de DI définies d'une relation r_1 sur un schéma R_1 vers une relation r_2 sur un schéma R_2. Rappelons que $I \subseteq C$ dénote l'ensemble des DI satisfaites de C, et que C_k et I_k dénotent les restrictions respectives de C et I à leurs éléments de taille k.

4.3.1 Justification théorique d'une approche optimiste

Nous justifions de manière théorique la mise en place d'une approche optimiste lorsque les DI connues satisfaites atteignent un certain niveau de spécialisation.

Cette justification peut s'énoncer de manière informelle de la façon suivante : si toutes les généralisations de taille k d'une DI candidate i sont satisfaites, alors i a plus de chances d'être satisfaite lorsque k augmente. Ce résultat se justifie par une règle d'inférence[1] des DF et des DI donnée par :

Proposition 4 *Soit* $2 \leq k < n$ *un entier et* $i = R_1[X] \subseteq R_2[Y], i \in C_n$ *telle que* $\forall j \in C_k, j \prec i,$ *on a* $j \in I_k$.

$$Si\ \exists Y_1 \subset Y \ avec\ |Y_1| = k-1 \ et\ r_2 \models Y_1 \to Y \setminus Y_1, \ alors\ i \in I$$

Preuve *Soit* $i = R_1[X] \subseteq R_2[Y] \in C$ *une expression de DI d'arité* $n \geq 3$, *et un entier* $k < n$. *Supposons que l'ensemble des DI qui généralisent i et dont la taille est égale à k soient satisfaites. Soit enfin* $Y_1 \subset Y$ *tel que* $|Y_1| = k-1$ *et* $r_2 \models Y_1 \to Y \setminus Y_1$.

Posons $Y \setminus Y_1 = B_1 ... B_{n-k+1}$. *Soit* X_1 *la sous-séquence de* X *dont la position des éléments dans* X *correspond à la position des éléments de* Y_1 *dans* Y, *et* $A_1, ..., A_{n-k+1}$ *les éléments de* X *dont les positions dans* X *correspondent aux positions respectives de* $B_1, ..., B_{n-k+1}$ *dans* Y.

[1] La règle d'inférence énoncée par la proposition 4 ne fait pas partie du système de Mitchell [Mit83] habituellement utilisé, mais est inférée par ce système qui est juste et complet. La démonstration proposée ici nous a semblé plus compréhensible et plus courte que celle consistant à utiliser les règles de Mitchell.

Soit $t \in r_1$. *On a* $R_1[X_1A_1] \subseteq R_2[Y_1B_1]$ *satisfaite par hypothèse, puisque cette DI est d'arité* k. *Donc* $\exists u_1 \in r_2$ *tel que* $u_1[Y_1B_1] = t[X_1A_1]$. *De même,* $R_1[X_1A_2] \subseteq R_2[Y_1B_2]$ *est satisfaite, donc* $\exists u_2 \in r_2$ *tel que* $u_2[Y_1B_2] = t[X_1A_2]$. *Or,* $r_2 \models Y_1 \rightarrow B_2$, *donc* $u_1[B_2] = u_2[B_2]$ *puisque* $u_1[Y_1] = u_2[Y_1]$. *Ainsi,* $u_1[Y_1B_1B_2] = t[X_1A_1A_2]$. *En réitérant* $n-k+1$ *fois la même démarche, on met en évidence que* $u_1[Y_1B_1B_2...B_{n_k+1}] = t[X_1A_1A_2...A_{n_k+1}]$, *soit* $u_1[Y] = t[X]$. *Ceci étant vrai pour tout tuple de* r_1, *on a* $\boldsymbol{d} \models R_1[X] \subseteq R_2[Y]$. □

Exemple 17 Soit la DI $i = R_1[ABCDE] \subseteq R_2[FGHIJ]$. Supposons que les 20 DI de taille 3 qui généralisent i soient satisfaites ; alors s'il existe deux attributs de $GHIJKL$ qui déterminent les autres dans leur relation, alors i est satisfaite.

Notons que plus l'entier k est grand, et plus on a de chance que des ensembles d'attributs de taille $k-1$ déterminent les autres, réunissant ainsi les conditions de la propriété 4. En particulier en reprenant les notations utilisées dans la propriété 4, plus l'ensemble Y_1 est grand, et plus il a de chances d'être une clé. Ceci justifie donc l'adoption d'une attitude optimiste d'autant plus que l'on se trouve "haut" dans l'espace de recherche.

4.3.2 La bordure positive optimiste

Nous donnons ici le cadre théorique permettant d'effectuer des sauts optimistes dans l'espace de recherche. Supposons qu'un sous-ensemble C' de C ait été exploré. On appelle $NI \subseteq C'$ les DI de C' non satisfaites dans $\mathbf{d}$. Un tel ensemble de DI non satisfaites permet d'élaguer un certain nombre de candidats dans C par anti-monotonie, et de définir ainsi un *ensemble optimiste de DI non contredites*. Plus formellement, on peut dire :

Définition 24 (Bordure optimiste) Soit $NI \in C$ un ensemble de DI non satisfaites dans $\mathbf{d}$. On appelle *ensemble optimiste de DI* associé à NI, que l'on note $I_{opt}(NI)$, l'ensemble défini par : $I_{opt}(NI) = \{i \in C \mid \nexists j \in NI, j \preceq i\}$. On appelle *bordure positive optimiste* associée à NI l'ensemble des DI les plus spécifiques de $I_{opt}(NI)$, que l'on note $\mathcal{B}d^+(I_{opt}(NI))$.

Lorsque NI sera clair dans le contexte, on notera I_{opt} pour $I_{opt}(NI)$, et donc $\mathcal{B}d^+(I_{opt})$ pour $\mathcal{B}d^+(I_{opt}(NI))$.

Exemple 18 Dans cette section, nous allons considérer l'exemple d'une base de données $\mathbf{d} = \{r_1, r_2\}$ sur un schéma $\mathbf{R} = \{R_1, R_2\}$, avec $R_1 = ABCDE$ et $R_2 = FGHIJ$. Supposons que les DI unaires satisfaites soient : $\{i_1 = A \subseteq F, i_2 = B \subseteq G, i_3 = C \subseteq H, i_4 = D \subseteq I, i_5 = E \subseteq J\}$. Supposons que l'on dispose de la connaissance de la non satisfaction de la DI $CE \subseteq HJ$, soit $NI = \{CE \subseteq HJ\}$. La bordure optimiste associée à

NI est $\mathcal{B}d^+(I_{opt}) = \{ABCD \subseteq FGHI, ABDE \subseteq FGIJ\}$, car ce sont les DI de C les plus spécialisées non invalidées par NI, comme l'illustre la figure 4.2 (la partie de C éliminée par les DI unaires non satisfaites n'est pas représentée).

La proposition suivante permet de réduire NI à ses éléments les plus généraux sans altérer la bordure positive optimiste correspondante.

Proposition 5 *Soit* $NI' = \{i \in NI \text{ t.q. } \nexists j \in NI, j \prec i\}$. *On a* $\mathcal{B}d^+(I_{opt}(NI)) = \mathcal{B}d^+(I_{opt}(NI'))$.

La démonstration découle directement de la définition de la bordure positive optimiste.

Nous allons voir que la bordure positive optimiste peut être calculée en utilisant les notions d'hypergraphe et de transversaux minimaux, définis comme suit (le lecteur pourra se référer, par exemple, à [Ber73] pour une description détaillée des hypergraphes).

Définition 25 (Hypergraphe) Soit V un ensemble fini d'éléments. Tout ensemble de sous-parties E de V définit un *hypergraphe* $\mathcal{H} = (V, E)$, où les éléments de V sont appelés les *sommets* de $\mathcal{H}$ et les éléments de E les *arêtes* de $\mathcal{H}$.

Définition 26 (Transversaux minimaux d'un hypergraphe) Un *transversal* T de $\mathcal{H} = (V, E)$ est un sous-ensemble de V dont l'intersection avec chaque élément de E est non vide. T est *minimal* s'il ne contient aucun autre transversal de $\mathcal{H}$.

Pour faire le lien entre les DI et les hypergraphes, il nous faut d'abord une représentation des DI sous forme d'ensembles, donnée dans la définition suivante :

Définition 27 (Représentation ensembliste des DI) A chaque DI de C on peut associer un ensemble de DI unaires en appliquant la fonction $ens : C \longrightarrow \mathcal{P}(I_1)$ définie par : $ens(i) = \{j \in I_1 \mid j \preceq i\}$.

Parce que les doublons d'attributs ont été permis dans la définition des DI, la fonction ens est bijective, et admet une fonction inverse ens^{-1} qui associe une DI unique de C à tout élément de 2^{I_1}. Dans [GKMT97], il est dit que les DI peuvent être représentées sous forme d'ensemble, mais la condition d'accepter les doublons n'est pas précisée.

Exemple 19 Dans l'exemple de la figure 4.2, on a $ens(CE \subseteq HJ) = \{i_3, i_5\}$ et

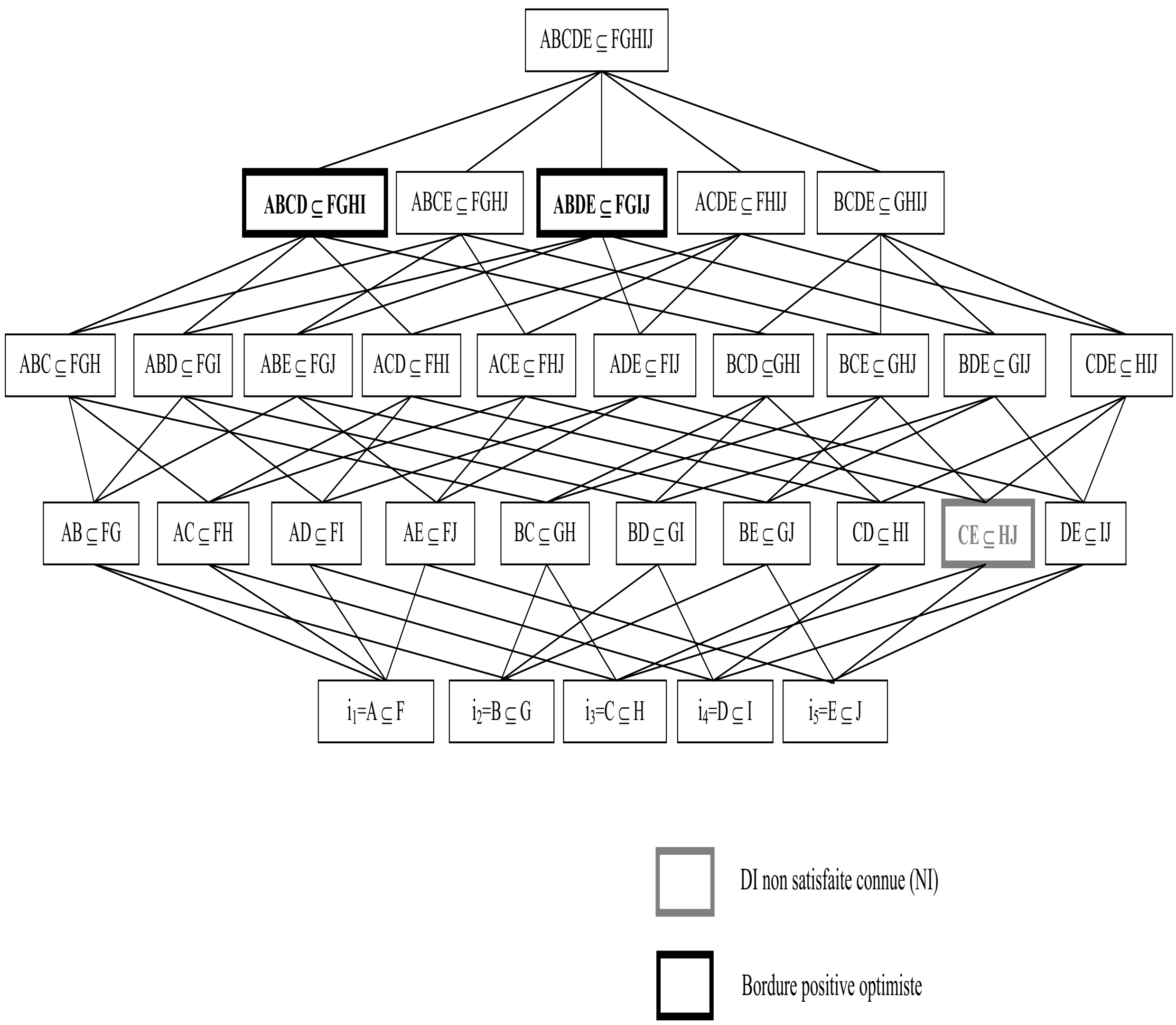

FIG. 4.2 – Illustration de la bordure positive optimiste

$ens^{-1}(i_1, i_2, i_3, i_4) = ABCD \subseteq FGHI$.

La proposition suivante découle directement de la représentation ensembliste des DI et de la définition de la relation '$\preceq$' :

Proposition 6 *Si i et j sont deux expressions de DI, $i \preceq j \Leftrightarrow ens(i) \subseteq ens(j)$.*

Munis de cette représentation, on définit un hypergraphe associé à un ensemble de DI :

Définition 28 (Hypergraphe associé à un ensemble de DI) On appelle hypergraphe associé à $C' \subseteq C$, noté $\mathcal{H}(C')$, l'hypergraphe (V, E) tel que : $V = \{i \in C \mid |i| = 1\}$ et $E = \{ens(i) \mid i \in C'\}$.

Exemple 20 Dans l'ensemble de DI de la figure 4.2, soit $NI = \{CE \subseteq HJ\}$. L'hypergraphe associé à NI est : $\mathcal{H}(NI) = (V, E)$ avec $V = \{i_1, i_2, i_3, i_4, i_5\}$ et $E = \{\{i_3, i_5\}\}$.

On obtient alors une relation entre les transversaux minimaux de l'hypergraphe associé à un ensemble de DI non satisfaites et la bordure positive optimiste associée.

Théorème 3 *Soit NI un ensemble de DI de C non satisfaites dans* **d**. *La bordure positive optimiste associée à NI est donnée par :*

$$\mathcal{B}d^+(I_{opt}(NI)) = \{i \in C \mid \overline{ens(i)} \in \mathcal{T}r(\mathcal{H}(NI))\}$$

où $\overline{ens(i)}$ est le complémentaire dans I_1 de $ens(i)$ et $\mathcal{T}r(\mathcal{H}(NI))$ l'ensemble des transversaux minimaux de $\mathcal{H}(NI)$.

Preuve *Soit $i \in C$*
On montre d'abord que $i \in I_{opt} \Leftrightarrow \overline{ens(i)}$ est un transversal de $\mathcal{H}(NI)$:
$i \in I_{opt}$
$\Leftrightarrow \forall j \in NI, j \not\preceq i$
$\Leftrightarrow \forall j \in NI, ens(j) \not\subseteq ens(i)$
$\Leftrightarrow \forall j \in NI, ens(j) \cap \overline{ens(i)} \neq \emptyset$
$\Leftrightarrow \overline{ens(i)}$ *est un transversal de $\mathcal{H}(NI)$.*

On montre ensuite que i est maximale dans $I_{opt} \Leftrightarrow \overline{ens(i)}$ est un transversal minimal de $\mathcal{H}(NI)$:
Soit $i \in \mathcal{B}d^+(I_{opt})$. Puisque $i \in I_{opt}$, $\overline{ens(i)}$ est un transversal de $\mathcal{H}(NI)$. Supposons qu'il

ne soit pas minimal : $\exists X \in 2^{I_1}$*, X transversal de* $\mathcal{H}(NI)$ *et* $X \subset \overline{ens(i)}$*, ce qui équivaut à* $ens(i) \subset \overline{X}$*. Donc* $ens^{-1}(\overline{X}) \in I_{opt}$ *et* $i \prec ens^{-1}(\overline{X})$ *par la proposition 6, ce qui contredit le fait que* $i \in \mathcal{B}d^+(I_{opt})$*.*
Soit enfin $X \in \mathcal{T}r(\mathcal{H}(NI))$*. X est un transversal, donc* $ens^{-1}(\overline{X}) \in I_{opt}$*. Supposons que* $ens^{-1}(\overline{X})$ *ne soit pas maximale :* $\exists j \in I_{opt} \mid ens^{-1}(\overline{X}) \prec j$*. Donc* $\overline{ens(j)}$ *est un transversal de* $\mathcal{H}(NI)$ *avec* $\overline{X} \subseteq ens(j)$*, soit* $\overline{ens(j)} \subseteq X$*, ce qui contredit le fait que X soit un transversal minimal.* □

Exemple 21 Sur l'ensemble de DI de la figure 4.2, soit $NI = \{CE \subseteq HJ\}$ et $\mathcal{H}(NI) = (V, E)$ avec $V = \{i_1, i_2, i_3, i_4, i_5\}$ et $E = \{\{i_3, i_5\}\}$. Les transversaux minimaux de $\mathcal{H}(NI)$ sont $\{i_3\}$ et $\{i_5\}$. Leurs complémentaires dans I_1 sont respectivement $\{i_1, i_2, i_4, i_5\}$ et $\{i_1, i_2, i_3, i_4\}$. Les DI correspondant à ces ensembles sont : $ABDE \subseteq FGIJ$ et $ABCD \subseteq FGHI$. On peut vérifier sur la figure 4.2 que ces deux DI sont bien les plus spécialisées non généralisées par $CE \subseteq HJ$.

4.3.3 Une approche hybride

La méthode proposée se décompose en deux phases principales : une exploration pessimiste des DI les plus générales jusqu'à un niveau k donné, puis un va et vient entre la bordure négative en construction et la bordure positive optimiste qui lui correspond.

Estimation de l'éloignement de la bordure positive

Lorsque une DI de la bordure positive optimiste est testée et se révèle fausse, une phase de correction doit être réalisée, consistant à parcourir l'espace se trouvant "en dessous" de cette DI. Cet espace peut être parcouru des DI les plus générales aux plus spécifiques, ou dans l'autre sens. Afin d'orienter l'algorithme dans le mode de parcours, nous "estimons" la distance entre la DI non satisfaite et la bordure positive réelle. Nous proposons pour cela d'utiliser la mesure d'erreur g'_3 donnée par [LPT02] :

$$g'_3(R_1[X] \subseteq R_2[Y], \mathbf{d}) =$$
$$1 - \frac{max\{|\pi_X(r')| \text{ telle que } r' \subseteq r \text{ et } (\mathbf{d} - \{r\}) \cup \{r'\} \models R_1[X] \subseteq R_2[Y]\}}{|\pi_X(r)|}$$

Intuitivement, g'_3 est la proportion de valeurs distinctes de X qu'il faudrait supprimer pour obtenir une base de données $\mathbf{d}'$ telle que $\mathbf{d}' \models R_1[X] \subseteq R_2[Y]$. Il s'agit d'une adaptation de la mesure d'erreur g_3, existant dans le cas des dépendances fonctionnelles approximatives [KM95]. Cette mesure d'erreur sera utilisée également plus loin, dans la définition des *DI approximatives*.

On montre d'après la définition de g'_3 et de la relation $\preceq$, que si $j \preceq i$, alors $g'_3(j) \leq g'_3(i)$. Ainsi, lorsqu'une DI i de la bordure positive optimiste est fausse, mais avec une erreur très faible, on peut raisonnablement espérer trouver une DI satisfaite parmi ses généralisations proches, et donc parcourir les généralisations de i des plus spécifiques aux plus générales. A l'inverse lorsque l'erreur est grande, et donc qu'un grand nombre de valeurs contredisent la DI, il apparaît raisonnable de parcourir les DI qui généralisent i des plus générales aux plus spécifiques.

Notons qu'il ne s'agit pas d'une règle sûre ; en effet, la mesure d'erreur n'est pas strictement décroissante lorsque les DI se généralisent, même très petite, elle peut rester constante sur plusieurs niveaux.

L'algorithme Zigzag

L'algorithme *Zigzag* (algorithme 6) recherche la bordure positive des DI d'une relation r_i vers une relation r_j, prenant en entrée l'ensemble des DI unaires satisfaites de r_i vers r_j. Pour trouver la bordure positive de l'ensemble des DI de la base, il faut appliquer l'algorithme *Zigzag* à chaque couple de relations de la base. Comme pour l'algorithme *Mind*, nous donnons également la bordure négative comme résultat, sans accès supplémentaire aux données.

Le principe est d'effectuer un va et vient entre $\mathcal{B}d^-(I)$ et $\mathcal{B}d^+(I)$ qui sont en construction. Dans un premier temps (ligne 1) un parcours purement pessimiste des candidats les plus généraux est effectué en utilisant l'algorithme $MIND$ décrit dans la section 4.2. Jusqu'à un niveau k donné, on initialise ainsi $\mathcal{B}d^+(I)$ et $\mathcal{B}d^-(I)$. La bordure positive optimiste $\mathcal{B}d^+(I_{opt})$ est alors calculée à partir de $\mathcal{B}d^-(I)$ conformément au théorème 3 (ligne 2). L'algorithme se termine lorsque toutes les DI de la bordure positive optimiste se trouvent dans la bordure positive connue (ligne 3). Les DI de $\mathcal{B}d^+(I_{opt}) \setminus \mathcal{B}d^+(I)$ sont évaluées sur la base. Celles qui sont satisfaites sont rajoutées à $\mathcal{B}d^+(I)$, les autres sont divisées en deux groupes suivant l'erreur commise : les "presque vraies" dans $optDI$ et les autres dans $pessDI$. Les DI qui généralisent les DI de $optDI$ sont alors parcourues "par le haut", des plus spécialisées aux plus générales ; les plus spécialisées vraies sont ajoutées à $\mathcal{B}d^+(I)$, les plus générales fausses sont incluses dans $\mathcal{B}d^-(I)$ (lignes 13 à 20). Enfin, "par le bas", les DI de taille $k+1$ qui généralisent les DI de $pessDI$ sont testées, les vraies s'ajoutent à $\mathcal{B}d^+(I)$ et les fausses s'ajoutent à $\mathcal{B}d^-(I)$ (lignes 22 à 25). $\mathcal{B}d^+(I_{opt})$ est ensuite mise à jour pour l'itération suivante (ligne 27).

Exemple 22

La figure 4.3 représente le même espace de recherche que celui de la figure 4.2 et renseigne sur la satisfaction de chaque candidat.

Illustrons sur cette figure l'exécution de l'algorithme 6, avec $k = 2$. Après un parcours par niveaux jusqu'au niveau 2, on a les initialisations :

Algorithme 6 Zigzag : découverte d'une couverture des DI satisfaites

Entrée: une base de données $\mathbf{d} = \{r_1, ..., r_n\}$, k dernier niveau pessimiste et I_1 les DI unaires de r_i vers r_j ;

Sortie: $\mathcal{B}d^+(I)$ bordure positive et $\mathcal{B}d^-(I)$ bordure négative des DI satisfaites de r_i vers r_j.

Initialiser $\mathcal{B}d^+(I)$ et $\mathcal{B}d^-(I)$ par niveaux jusqu'au niveau k ;
Calcul de $\mathcal{B}d^+(I_{opt})$ à partir de $\mathcal{B}d^-(I)$;
while $\mathcal{B}d^+(I_{opt}) \neq \mathcal{B}d^+(I)$ **do**
  $optDI = pessDI = \emptyset$;
  for all $i \in \mathcal{B}d^+(I_{opt}) \setminus \mathcal{B}d^+(I)$ **do**
    if $(g_3'(i, \mathbf{d}) = 0)$ **then** $\mathcal{B}d^+(I) = \mathcal{B}d^+(I) \cup \{i\} \setminus \{j \in \mathcal{B}d^+(I) \mid j \prec i\}$;
    else
      $\mathcal{B}d^-(I) = \mathcal{B}d^-(I) \cup i$;
      if $(g_3'(i, \mathbf{d}) \leq \epsilon$ and $|i| > k + 1)$
      then $optDI = optDI \cup \{i\}$;
      else $pessDI = pessDI \cup \{i\}$;
  end for
  while $optDI \neq \emptyset$ **do**
    $candidats = \cup_{i \in optDI}\{j \mid j \preceq i, |j| = |i| - 1$ and $|j| > k\}$;
    for all $i \in candidats$ **do**
      if $(\mathbf{d} \models i)$ **then** $\mathcal{B}d^+(I) = \mathcal{B}d^+(I) \cup \{i\} \setminus \{j \in \mathcal{B}d^+(I) \mid j \prec i\}$; $candidats = candidats \setminus \{i\}$;
      else $\mathcal{B}d^-(I) = \mathcal{B}d^-(I) \cup \{i\} \setminus \{j \in \mathcal{B}d^-(I) \mid i \prec j\}$;
    end for
    $optDI = candidats$;
  end while
  $C_{k+1} = \cup_{i \in pessDI}\{j \mid j \prec i, |j| = k + 1\}$;
  for all $i \in C_{k+1}$ **do**
    if $(\mathcal{B}d^+(I) \models i$ or $\mathbf{d} \models i)$ **then** $\mathcal{B}d^+(I) = \mathcal{B}d^+(I) \cup \{i\} \setminus \{j \in \mathcal{B}d^+(I) \mid j \prec i\}$;
    else $\mathcal{B}d^-(I) = \mathcal{B}d^-(I) \cup \{i\} \setminus \{j \in \mathcal{B}d^-(I) \mid i \prec j\}$;
  end for
  $k = k + 1$;
  Calcul de $\mathcal{B}d^+(I_{opt})$ à partir de $\mathcal{B}d^-(I)$;
end while
Retourner $\mathcal{B}d^+(I)$ et $\mathcal{B}d^-(I)$.

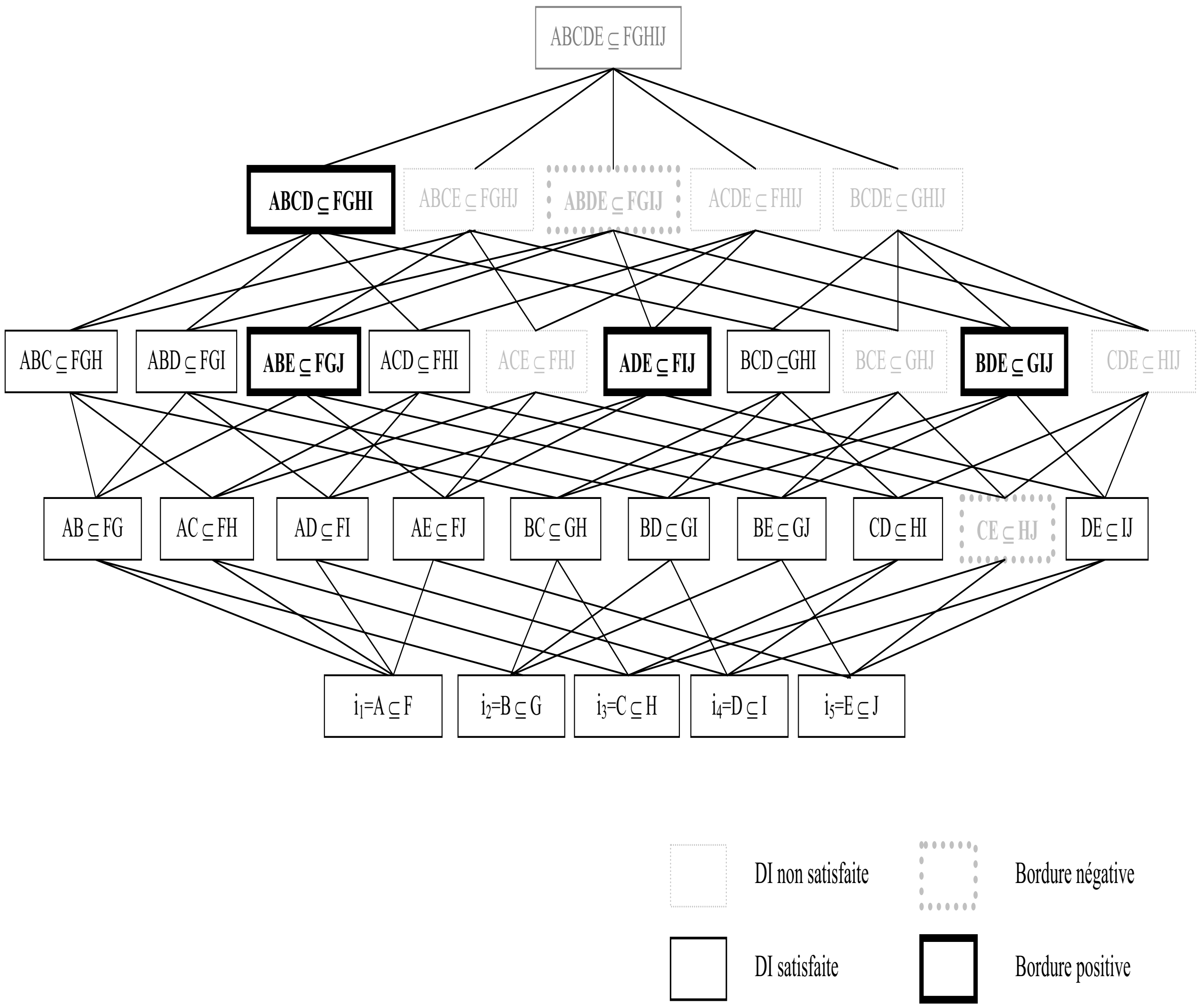

Fig. 4.3 – Un espace de recherche des DI

$\mathcal{B}d^+(I) = \{AB \subseteq FG, AC \subseteq FH, AD \subseteq FI, AE \subseteq FJ, BC \subseteq GH, BD \subseteq GI, BE \subseteq GJ, CD \subseteq HI, DE \subseteq IJ\}$;

$\mathcal{B}d^-(I) = \{CE \subseteq HJ\}$.

$\mathcal{B}d^+(I_{opt})$ est donc calculé à partir de $\mathcal{B}d^-(I)$, et on trouve $\mathcal{B}d^+(I_{opt}) = \{ABCD \subseteq FGHI, ABDE \subseteq FGIJ\}$ (cf. exemple 21 pour les détails).

Ces deux DI sont testées sur la base. $ABCD \subseteq FGHI$, étant satisfaite, est rajoutée à $\mathcal{B}d^+(I)$, les DI qui la généralisent sont supprimées. Ainsi, $\mathcal{B}d^+(I) = \{ABCD \subseteq FGHI, AE \subseteq FJ, BE \subseteq GJ, DE \subseteq IJ\}$. Par contre $ABDE \subseteq FGIJ$ n'étant pas satisfaite, elle est rajoutée à $\mathcal{B}d^-(I)$ et on obtient $\mathcal{B}d^-(I) = \{CE \subseteq HJ, ABDE \subseteq FGIJ\}$.

La mesure d'erreur pour la DI $ABDE \subseteq FGIJ$ n'a pas d'importance, puisque la phase de correction revient dans tous les cas à vérifier les DI de taille 3 qui généralisent $ABDE \subseteq FGIJ$. Seules les DI n'étant pas spécialisées par un élément de $\mathcal{B}d^+(I)$ sont testées sur la base, soit $ABE \subseteq FGJ, ADE \subseteq FIJ$ et $BDE \subseteq GIJ\}$. Ces trois DI sont satisfaites, donc rajoutées à $\mathcal{B}d^+(I)$ qui est mis à jour en conséquence :

$\mathcal{B}d^+(I) = \{ABCD \subseteq FGHI, ABE \subseteq FGJ, ADE \subseteq FIJ, BDE \subseteq GIJ\}$.

Cas des DI sans doublons

Comme pour l'algorithme *Mind*, l'algorithme *Zigzag* peut être adapté à la découverte des DI sans doublons. Pour cela, il suffit d'introduire dans la bordure négative, dès son initialisation (ligne 1 de l'algorithme 6) l'ensemble des DI de taille 2 avec des doublons composées de deux DI unaires satisfaites.

En effet, soit i une DI ayant au moins un doublon à gauche (la justification est identique pour les doublons à droite). i est donc de la forme $R_1[X_1AAX_2] \subseteq R_2[Y_1BCY_2]$. Il y a donc au moins une DI de taille 2, ici $j = R_1[AA] \subseteq R_2[BC]$, qui généralise i. Si j fait partie de la bordure négative, alors i ne peut pas faire partie de la bordure positive optimiste correspondante, d'après la définition 24, et donc $ens(i)$ ne peut pas faire partie des transversaux minimaux de l'hypergraphe associé à la bordure négative (théorème 3).

4.3.4 Calcul des transversaux minimaux

Outre le nombre de DI testées, qui peut être factoriel dans le pire cas, la complexité de l'algorithme réside dans le calcul de la bordure positive optimiste, qui induit le calcul des minimaux transversaux d'un hypergraphe. Il s'agit d'un problème souvent étudié dans la littérature (voir par exemple [EG95] pour une synthèse sur ce problème). Même si sa complexité exact reste encore un problème ouvert à notre connaissance, un algorithme quasi-polynomial dans la taille de la sortie a été proposé dans [FK96].

Nous utilisons la méthode proposée dans [DT95] pour le calcul des transversaux minimaux, reportée dans l'algorithme 7.

Algorithme 7 Calcul des transversaux minimaux d'un hypergraphe

Entrée: Un hypergraphe $\mathcal{H} = (V, E)$;
Sortie: $\mathcal{T}r(\mathcal{H})$, ensemble des transversaux minimaux de $\mathcal{H}$.

$\mathcal{T}r(\mathcal{H}) = \{\emptyset\}$
for all $X \in E$ **do**
  $S = \{Y \in \mathcal{T}r(\mathcal{H}) \mid X \cap Y \neq \emptyset\}$;
  $B = \{Y \in \mathcal{T}r(\mathcal{H}) \mid X \cap Y = \emptyset\}$;
  for all $Y_1 \in B$ **do**
    for all $v \in V$ **do**
      if $\nexists Y_2 \in S \mid Y_2 \subseteq Y_1 \cup \{v\}$ **then**
        $\mathcal{T}r(\mathcal{H}) = (\mathcal{T}r(\mathcal{H}) \setminus Y_1) \cup (Y_1 \cup \{v\})$;
      end if
    end for
  end for
end for
Retourner $\mathcal{T}r(\mathcal{H})$.

Cet algorithme est incrémental par rapport aux arêtes de l'hypergraphe. Cette caractéristique possède un fort intérêt dans notre cas. Un premier calcul de la bordure positive optimiste a lieu après le parcours pessimiste des candidats les plus généraux ; par la suite, les nouvelles arêtes correspondant aux nouvelles DI testées comme non-satisfaites sont prises en compte pour mettre à jour la bordure positive optimiste, sans reconsidérer l'ensemble de la bordure négative.

Une optimisation peut également être apportée au calcul de $\mathcal{B}d^+(I_{opt})$. En effet, à chaque itération, cet ensemble contient à la fois les grandes DI suspectées, mais aussi les DI de plus petite taille satisfaites qui ont déjà été découvertes. L'idée est donc de caractériser plus finement les éléments de $\mathcal{B}d^+(I_{opt})$: ceux qui sont à découvrir, et ceux qui appartiennent à l'ensemble déjà exploré. La proposition suivante se déduit simplement du théorème 3.

Proposition 7 *Soit I_k l'ensemble des DI satisfaites de taille inférieure ou égale à k, NI l'ensemble connu des DI non satisfaites, et n le nombre de DI unaires. On a :*

$$i \in (\mathcal{B}d^+(I_{opt}) \setminus I_k) \Longleftrightarrow \overline{ens(i)} \in \mathcal{T}r(\mathcal{H}(NI)) \text{ et } |\overline{ens(i)}| \leq n - k$$

En pratique, cette condition permet d'optimiser la génération des transversaux minimaux en permettant de ne pas considérer les candidats de taille supérieure à $n - k$.

Chapitre 5

Découverte des DI approximatives

Sommaire

Dans de nombreuses bases de données, et tout particulièrement lorsque les clés étrangères ne sont pas définies et contrôlées lors des opérations de mise à jour, des incohérences peuvent survenir dans les données. Il se peut alors que des contraintes sémantiquement correctes et valides dans le schéma soient contredites dans une instance particulière. Avec les algorithmes proposés plus haut, de nombreuses DI peuvent rester inconnues à cause d'une seule faute de frappe dans la base, et ainsi conduire l'analyste à une compréhension faussée de la sémantique des données.

Nous donnons tout d'abord la définition des dépendances d'inclusion approximatives, basée sur une mesure d'erreur. Puis nous montrons comment adapter les différents algorithmes présentés à la découverte des DI approximatives.

5.1 Définition

Dans un premier temps, nous devons définir une mesure d'erreur indiquant, pour une DI donnée, son degré de satisfaction. Nous utiliserons la mesure d'erreur g_3' donnée dans le chapitre précédent, dont nous rappelons la définition.

$$g_3'(R[X] \subseteq S[Y], \mathbf{d}) = 1 - \frac{max\{|\pi_X(r')| \text{ s.t. } r' \subseteq r \text{ and } (\mathbf{d} - \{r\}) \cup \{r'\} \models R[X] \subseteq S[Y]\}}{|\pi_X(r)|}$$

La forme de la définition est faite pour rappeler la définition donnée dans [KM95] de l'erreur g_3 pour les DF approximatives. Une expression équivalente sous forme algébrique est :

$$g_3'(R[X] \subseteq S[Y], \mathbf{d}) = \frac{|\pi_X(r) \setminus \pi_Y(s)|}{|\pi_X(r)|}$$

Cette expression permet le calcul de g_3' à l'aide par exemple de requêtes SQL.

De cette mesure d'erreur découle de façon naturelle la notion de *satisfaction approximative d'une DI.*

Définition 29 (Satisfaction approximative d'une DI) Soit $\mathbf{d}$ une base de données sur un schéma $\mathbf{R}$, i une DI sur $\mathbf{R}$ et ϵ un entier dans $[0, 1]$. On dit que i est *approximativement satisfaite* dans une base de données $\mathbf{d}$ par rapport à ϵ, noté $\mathbf{d} \models_\epsilon i$, si et seulement si $g_3'(i, \mathbf{d}) \leq \epsilon$.

Les DI approximativement satisfaites sont appelées *DI approximatives.*

Exemple 23 Dans la base de données **movies** page 8, on a $g_3'(F \subseteq H, \mathbf{movies}) = 1/9$. Si on prend $\epsilon = 0.2$, on a $\mathbf{movies} \models_\epsilon F \subseteq H$.

Les deux parties suivantes montrent comment les algorithmes proposés plus haut pour la découverte des DI exactes peuvent être étendus à la découverte des DI approximatives.

5.2 Découverte des DI unaires approximatives

Les DI unaires approximatives peuvent être découvertes à partir de la même réorganisation des données que celle utilisée pour les DI unaires exactes. A partir du contexte d'extraction associé à la base de données en entrée, la proposition 8 donne une caractérisation des dépendances unaires approximatives.

Proposition 8 *Etant donné un seuil ϵ, une base de données $\boldsymbol{d}$, un type de données t dans $\boldsymbol{d}$ et le contexte d'extraction associé $\mathbb{D}_t(d) = (\mathbb{V}, \mathbb{U}, \mathbb{B})$,*

$$d \models_\epsilon A \subseteq B$$

$$\Longleftrightarrow$$

$$1 - \sum_{v \in \mathbb{V} | (v,A) \in \mathbb{B}} \frac{|\{v \in \mathbb{V} \mid (v,A) \in \mathbb{B} \; et \; (v,B) \in \mathbb{B}\}|}{|\{v \in \mathbb{V} \mid (v,A) \in \mathbb{B}\}|} \leq \epsilon$$

où $A, B \in \mathbb{U}$.

A partir de cette caractérisation, les DI unaires approximatives satisfaites peuvent être découvertes en seulement un passage sur la relation binaire. À nouveau, remarquons que, si on voit le contexte d'extraction comme une base de données de transactions, les DI unaires approximatives qui sont satisfaites par rapport à un seuil ϵ correspondent aux *règles d'associations entre singletons d'items* dans cette base de transaction, avec une confiance supérieure à $(1 - \epsilon)$.

Exemple 24 Considérons le type string (cf Figure 3.1) dans l'exemple 8 page 33. L'attribut F apparaît dans 9 lignes de la relation binaire. Dans 8 de ces 9 lignes, l'attribut H apparaît également. L'erreur $g_3'(F \subseteq H, movies)$ est donc $1 - \frac{8}{9} = \frac{1}{9}$, ce qui correspond bien à la proportion de valeurs à enlever de la projection de F pour satisfaire la DI. Si on voit le contexte d'extraction comme une base de données de transactions, alors la confiance de la règle d'association $F \Rightarrow H$ est $\frac{8}{9}$.

L'algorithme 8 calcule l'ensemble des DI unaires approximatives (non triviales) satisfaites dans une base de données, à partir du contexte d'extraction associé. L'idée est similaire à celle utilisée pour les DI unaires exactes. Pour chaque attribut A, on construit un ensemble $rhs(A) = \{< B, n_{AB} >|$ B est du même type que A $\}$ où n_{AB} est le nombre de lignes de la relation binaire contenant à la fois A et B.

L'exemple 25 illustre le déroulement de l'algorithme.

Exemple 25 Considérons le type string (cf Figure 3.1) dans l'exemple 8 page 33. Une phase d'initialisation construit les ensembles suivants :
$rhs(B) = rhs(F) = rhs(H) = \{< B, 0 >, < F, 0 >, < H, 0 >\}$.

Ensuite, la première ligne de la relation binaire est considérée : $l_1 = \{F, H\}$. Les ensembles rhs sont alors mis à jour :
$rhs(B) = \{< B, 0 >, < F, 0 >, < H, 0 >\}$,
$rhs(F) = \{< B, 0 >, < F, 1 >, < H, 1 >\}$,
$rhs(H) = \{< B, 0 >, < F, 1 >, < H, 1 >\}$.

A partir de la seconde ligne $l_2 = \{B, H\}$ on obtient :
$rhs(B) = \{< B, 1 >, < F, 0 >, < H, 1 >\}$,
$rhs(F) = \{< B, 0 >, < F, 1 >, < H, 1 >\}$,
$rhs(H) = \{< B, 1 >, < F, 1 >, < H, 2 >\}$.

Ces opérations sont répétées pour chaque ligne de la relation binaire. Finalement, après

Algorithme 8 découverte des DI unaires approximatives

Entrée: Le contexte d'extraction $(\mathbb{V}, \mathbb{U}, \mathbb{B})$, associé à **d** et t, et un seuil ϵ.
Sortie: AI_1 l'ensemble des DI unaires approximatives satisfaites dans **d** entre les attributs de type t.

for all $A \in \mathbb{U}$ **do**
  $rhs(A) = \{< B, 0 >\mid B \in \mathbb{U}\}$;
end for
for all $v \in \mathbb{V}$ **do**
  for all A t.q. $(v, A) \in \mathbb{B}$ **do**
    for all $< B, n_{AB} > \in rhs(A)$ s.t. $(v, B) \in \mathbb{B}$ **do**
      $n_{AB} = n_{AB} + 1$;
    end for
  end for
end for
for all $A \in \mathbb{U}$ **do**
  for all $< B, n_{AB} > \in rhs(A) \setminus \{< A, n_{AA} >\}$ **do**
    if $1 - (n_{AB}/n_{AA}) \leq \epsilon$ **then**
      $AI_1 = AI_1 \cup \{A \subseteq B\}$;
    end if
  end for
end for
Retourner AI_1.

un parcours, on a :
$rhs(B) = \{< B, 7 >, < F, 1 >, < H, 7 >\}$,
$rhs(F) = \{< B, 1 >, < F, 9 >, < H, 8 >\}$,
$rhs(H) = \{< B, 7 >, < F, 8 >, < H, 16 >\}$.

A partir de ces ensembles, la mesure d'erreur g'_3 peut être calculée pour chaque DI unaire. Par exemple, $g'_3(F \subseteq H, \mathbf{movies}) = 1 - \frac{8}{9}$, $g'_3(B \subseteq H, \mathbf{movies}) = 1 - \frac{7}{7} = 0$.

Si on considère une mesure d'erreur de 0.2, on obient l'ensemble des DI unaires approximatives satisfaites : $AI_1 = \{A \subseteq G, B \subseteq H, C \subseteq I, E \subseteq G, F \subseteq H\}$.

5.3 Découverte des DI n-aires approximatives

5.3.1 Cadre de travail

Le cadre de travail défini pour les DI n-aires exactes peut être utilisé pour les DI n-aire approximatives. L'espace de recherche est le même, et la propriété d'anti-monotonie est toujours valide :

Propriété 3 *Soit* $\mathbf{d}$ *une base de données,* C *l'espace de recherche associé, et* i *et* j *deux expressions de DI dans* C *telles que* $j \preceq i$. *Quelque soit* $\epsilon \in [0; 1]$, *on a :*
- *si* $\mathbf{d} \models_\epsilon i$ *then* $\mathbf{d} \models_\epsilon j$ *ou encore*
- *si* $\mathbf{d} \not\models_\epsilon j$ *then* $\mathbf{d} \not\models_\epsilon i$.

Preuve *Ce résultat provient du fait que la mesure d'erreur est croissante relativement à l'ordre* $\preceq$, *soit :* $j \preceq i \Rightarrow g'_3(j, \mathbf{d}) \leq g'_3(i, \mathbf{d})$, *ce qui découle de la définition de* g'_3 *et de la relation* $\preceq$. □

La bordure positive des DI approximativement satisfaites représente aussi une solution au problème de la découverte des DI approximatives. Elles sont les DI les plus spécialisées dont l'erreur est inférieure à ϵ. De même, la bordure négative se définit comme les DI les plus générales dont la mesure d'erreur est strictement supérieure à ϵ.

L'application des algorithmes *Mind* et *Zigzag* est donc assez naturelle dans ce contexte.

5.3.2 Découverte des DI n-aires approximatives avec *Mind*

Une seule ligne est à changer dans l'algorithme 4 (page 44) : dans la ligne 11, il faut remplacer $\mathbf{d} \models i$ par $\mathbf{d} \models_\epsilon i$. L'algorithme de génération des candidats est inchangé.

5.3.3 Découverte des DI n-aires approximatives avec *Zigzag*

Là encore, l'algorithme *Zigzag* peut s'appliquer presque sans changements, d'autant plus que celui-ci utilisait déjà la mesure d'erreur g_3' pour orienter le parcours dans l'espace de recherche.

Chapitre 6

Expérimentations

Sommaire

Les expérimentations sont réalisées sur un Intel pentium III 500 Mhz, avec 384 MB de RAM et tournant sous Windows 2000 Pro. Les algorithmes sont implantés à l'aide du langage C++ avec utilisation de la STL. Le SGBD est Oracle 9i, et les accès à la base sont réalisés via des drivers ODBC.

6.1 Accès aux données

6.1.1 Construction des contextes d'extraction

La construction du contexte d'extraction est une étape importante dans les performances de la méthode proposée pour la découverte des DI unaires, puisqu'il s'agit de la phase primordiale d'accès aux données. Nous donnons certains détails sur la structure utilisée et la méthode retenue pour la construction du contexte d'extraction.

Structure de données

Une première contrainte est que la relation binaire du contexte d'extraction doit être maintenue triée sur les valeurs, puisque chacune de ces valeurs ne doit apparaître qu'une fois.

Toutefois, pour des raisons de performances, le tri des valeurs est laissé à la charge du SGBD. Les algorithmes de tri implantés dans les SGBD sont en effet extrêmement optimisés, et exploitent souvent un accès privilégié aux données.

La structure retenue est donc une simple liste chaînée ; chaque élément de la liste est une paire $(valeur, attributs)$. L'élément *attributs* est l'ensemble des attributs prenant la valeur *valeur*. Chacun des N attributs est repéré par un entier unique compris entre 0 et $N-1$, permettant le stockage des ensembles d'attributs sous la forme d'un *bitmap* [GMUW00]. Tous les ensembles d'attributs sont ainsi représentés par N bits, le i^{eme} bit prenant la valeur 1 si l'attribut i appartient à l'ensemble, 0 sinon.

Espace mémoire

Soit $\mathbf{d}$ une base de données et t un des types de $\mathbf{d}$, n_t le nombre de valeurs distinctes sur le type t, et N_t le nombre d'attributs définis sur le type t. Enfin, soit c_t le nombre de bits nécessaires au codage d'un élément de type t. La place mémoire de chaque élément (paire) de la liste est $(c_t + N_t)$ bits, ce qui conduit à un nombre total de $n_t.(c_t + N_t)$ bits.

Exemple 26 Supposons que $\mathbf{d}$ soit composée de 500 attributs prenant des valeurs entières, codées sur 32 bits, et que le nombre total de valeurs distinctes prises par ces attributs dans $\mathbf{d}$ soit 3500000. La taille de la relation binaire obtenue sera alors $(32+500) \times 3500000$ soit 232 Méga octets.

On voit donc que, même pour une taille relativement grande de base de données, on peut supposer pouvoir maintenir en mémoire la relation binaire. En outre, un codage plus adapté utilisant par exemple la *compression de bitmaps* (décrite notamment dans [GMUW00]) pourrait considérablement réduire la taille nécessaire pour son stockage. Une telle optimisation n'a pas été étudiée dans le cadre de ce mémoire.

Construction

L'accès aux données est réalisé à l'aide de requêtes SQL. Une requête simple est effectuée pour chaque attribut A, de la forme

```
SELECT DISTINCT A FROM R
```

Les données extraites par le SGBD sont transférées au programme au travers de curseurs. Chaque lecture dans le curseur induit un *changement de contexte* entre la zone mémoire du SGBD et celle du programme, opération réputée coûteuse [STA00]. Afin d'économiser le nombre de changement de contexte, le programme accède aux données par paquets, en utilisant une technique de "Bulk loading".

On dispose ainsi de l'ensemble distinct et trié des valeurs prises par un attribut donné, permettant de mettre à jour la liste chaînée représentant le contexte d'extraction. La liste chaînée étant maintenue triée, cette mise à jour se fait en seulement un parcours.

6.1.2 Test d'une DI sur la base

Le coût d'un algorithme de découverte de dépendances dépend non seulement du nombre de candidats à tester, mais aussi du coût du test d'un candidat sur la base. Dans le cas des DI, le test de base d'une DI revient à parcourir la partie droite pour chaque valeur de la partie gauche, donc en $O(n^2)$, si n est le nombre de tuples dans les relations.

La complexité peut être améliorée en triant les deux parties. Ainsi, le test consiste en deux tris et un parcours des deux parties, donc en $O(2n \times (log(n) + 1))$.

Différentes requêtes SQL peuvent effectuer le test. Les performances varient énormément suivant la nature des données, le degré de satisfaction d'une DI, le plan d'exécution choisi par le SGBD. La solution la plus performante dans nos conditions d'expérimentations a été la requête donnée dans la figure 6.1.

FIG. 6.1 – Requête SQL pour tester la DI $R[X_1...X_n] \subseteq S[Y_1...Y_n]$.

```
SELECT X
FROM r
WHERE NOT EXISTS ( SELECT *
                    FROM s
                    WHERE r.A_1 = s.B_1
                    AND ...
                    AND r.A_n = s.B_n  )
AND ROWNUM < 2
```

Cette requête retourne la première valeur trouvée qui contredit la DI. Il suffit donc de tester la cardinalité du résultat pour connaître la satisfaction de la DI.

Dans le cas des DI approximatives, on doit connaître la proportion de valeurs qui violent la DI. Il faut donc accéder au nombre de valeurs distinctes dans la partie gauche par une requête préliminaire. Ensuite, une requête similaire à celle de la figure 6.1 recherche les valeurs qui contredisent la DI ; toutefois la dernière ligne doit être paramétrée par ϵ, le seuil de satisfaction choisi par l'utilisateur. Si N_v est le nombre de valeurs qui violent la DI et N le nombre de valeurs distinctes dans la partie gauche, la DI est satisfaite si $\frac{N_v}{N} \leq \epsilon$, donc si $N_v \leq N \times \epsilon$. Puisque N est connu, on peut utiliser la requête de la figure 6.2.

FIG. 6.2 – Requête SQL pour tester la DI approximative $R[X_1...X_n] \subseteq S[Y_1...Y_n]$.

```
SELECT X
FROM r
WHERE NOT EXISTS ( SELECT *
                    FROM s
                    WHERE r.A_1 = s.B_1
                    AND ...
                    AND r.A_n = s.B_n  )
AND ROWNUM < (N . epsilon)
```

Le coût du test des DI approximatives est donc supérieur à celui des DI exactes.

6.2 Tests sur la base de données movies

Nous avons testé la faisabilité de nos algorithmes sur la base de données **movies** présentée page 8. La bordure positive $\mathcal{B}d^+(I_{\mathbf{movies}})$ des DI valides dans le schéma est disponible dans la documentation fournie sur le web. Elle est composée de 2 DI de taille 3, 1 DI de taille 2 et 1 DI de taille 1.

Nous avons lancé l'algorithme *Zigzag* sur cette base, avec une approche par niveaux jusqu'au niveau 2. La bordure positive trouvée se compose de : 1 DI de taille 3, 1 DI de taille 2 et 5 DI de taille 1, découvertes en 17 secondes. Parmi elles, 4 DI de taille 1 sont accidentellement satisfaites, et ne font pas partie du schéma. Notons que ces DI accidentelles sont définies exclusivement sur des attributs de type *Date*.

On constate alors que 1 DI de taille 3 n'a pas été découverte par notre programme, bien qu'elle soit valide dans le schéma et donc censée être satisfaite dans toutes les instances. Ce problème est dû à certaines erreurs qui apparaissent dans **movies**. Nous avons alors relancé le programme pour découvrir les DI approximatives, autorisant un taux d'erreur de 0.03. Cette fois, toutes les DI du schéma ont été découvertes en 23 secondes.

6.3 Tests sur des bases de données synthétiques

Pour commencer, remarquons que le nombre de relations dans les bases de tests n'a pas d'importance. En effet, la découverte des DI unaires décharge les données indépendamment de la relation d'où elles proviennent. Ensuite, les parcours de l'espace de recherche des DI n-aires est uniquement dépendant des DI unaires satisfaites et de la bordure à découvrir.

6.3.1 Description des données

Trois séries de base de données synthétiques ont été utilisées, tous les attributs prenant des valeurs de même type (entier) :

- La première série est destinée à observer le comportement de nos algorithmes en fonction du nombre de tuples par relation. Chaque base possède 50 attributs, et la même bordure positive de DI à découvrir : 4 DI de taille 1 et 1 DI de taille 6.
- La deuxième série a pour but de faire varier le nombre d'attributs. Chaque base possède 90000 tuples par relation, et la bordure positive reste inchangée.
- Enfin, la troisième série est composée de trois bases de 50 attributs et 90000 tuples, mais différentes cette fois dans la bordure positive des DI à découvrir :
  - base 3.1 : 10 DI, avec des arités de 2,5,6 et 7 ;
  - base 3.2 : 10 DI, avec des arités de 3,5,6 et 11 ;
  - base 3.3 : 20 DI, avec des arités de 6,8,13,17 et 18 ;

Notons que les bases de données ont été construites en utilisant notamment une implantation de l'algorithme *Chase* [BV84], conçu pour satisfaire exactement un ensemble de DI dans une base de données.

6.3.2 Découverte des DI unaires

Afin de mettre en évidence l'intérêt de l'approche présentée pour la découverte des DI unaires, nous l'avons comparée avec l'approche de type "générer/tester", illustrée dans l'algorithme 1 page 32.

Pour la découverte des DI unaires, la nature de la bordure positive à découvrir n'influe pas sur les performances. Les méthodes sont donc testées uniquement sur les deux premières séries de test, faisant varier le nombre de tuples par relation et le nombre d'attributs. Les temps d'exécution obtenus sont reportés dans les figures 6.3 et 6.4.

On constate que l'approche proposée obtient des temps d'exécution bien meilleurs que l'approche naïve. Les deux programmes ont un temps d'exécution linéaire en fonction du nombre de tuples, mais un rapport d'environ 25 existe entre les deux droites obtenues. Notons que ce rapport correspond à $N/2$ si N est le nombre d'attributs ; ce résultat confirme la prépondérance des entrées/sorties dans le temps d'exécution, puisque qu'il corrobore l'analyse théorique faite dans le chapitre 3.4 page 35.

Lorsque c'est le nombre d'attributs qui varie, on observe de nouveau un comportement

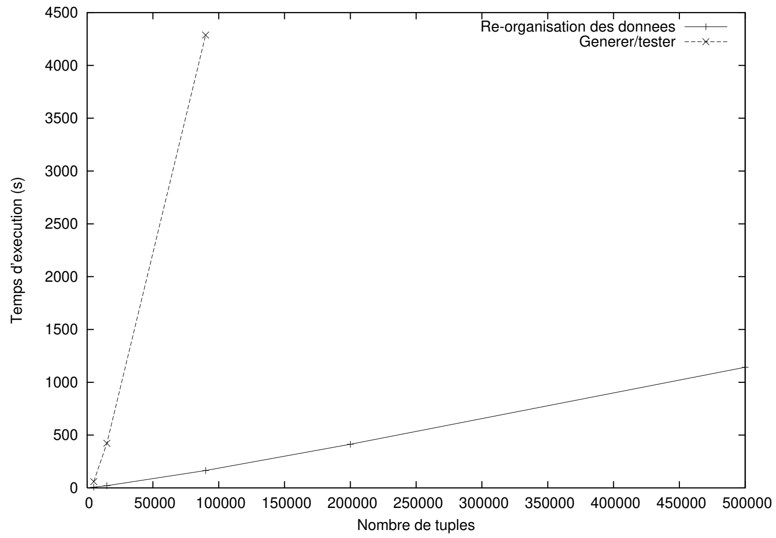

FIG. 6.3 – Temps d'exécution pour la découverte des DI unaires en fonction du nombre de tuples

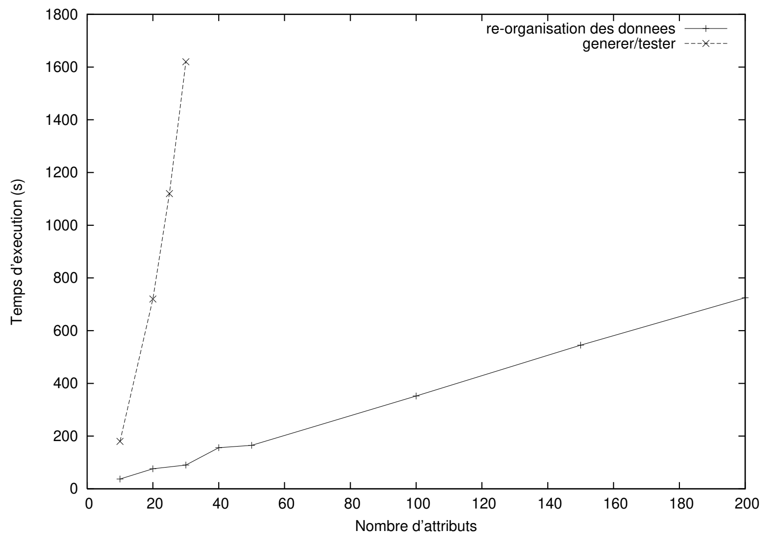

FIG. 6.4 – Temps d'exécution pour la découverte des DI unaires en fonction du nombre d'attributs

linéaire dans le cas de notre approche, alors que l'approche de type "générer/tester" a un comportement quadratique (dû au nombre quadratique de candidats).

6.3.3 Découverte des DI n-aires

La méthode utilisée est l'implantation de l'algorithme *Zigzag*, avec une approche purement optimiste (*Mind*) pour les deux premiers niveaux. Notons que pour les DI n-aires, le nombre d'attributs n'a plus d'importance, puisque l'espace de recherche à parcourir dépend uniquement des DI unaires satisfaites et de la bordure positive à découvrir.

La figure 6.5 montre l'évolution du temps d'exécution pour la découverte de la bordure positive des DI (DI unaires incluses) sur la première série de bases de tests, donc en fonction du nombre de tuples dans les relations.

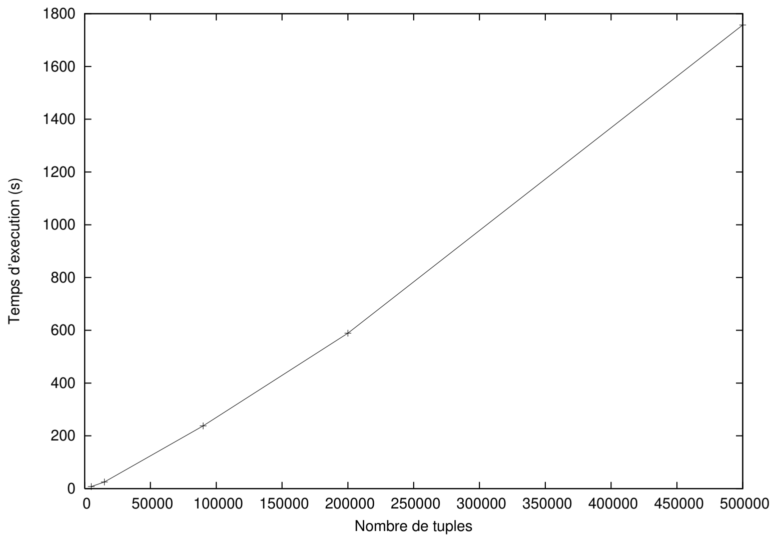

Fig. 6.5 – Temps d'exécution pour une découverte exhaustive des DI en fonction du nombre de tuples

Malgré la complexité inhérente à la découverte des DI, on constate que le temps n'excède pas 30 minutes. En outre, on observe un comportement linéaire du temps d'exécution en fonction du nombre de tuples. Ce comportement s'explique par le fait que la découverte des DI unaires a un comportement linéaire, ainsi que le coût d'un test sur la base.

Ces résultats montrent l'intérêt de notre méthode pour des tailles de DI satisfaites assez petites, ce qui semble être fréquent dans les bases de données réelles.

Enfin, la figure 6.6 donne les temps d'exécution de notre méthode pour la troisième série de bases de tests, donc pour découvrir des bordures positives de DI satisfaites composées de DI de plus grande taille.

Ajoutons qu'avec une approche purement pessimiste par niveaux, le temps d'exécution

Fig. 6.6 – Résultats expérimentaux

base	Temps d'exécution
base 3.1	1 754 s.
base 3.2	3 500 s.
base 3.3	7 729 s.

dans nos conditions expérimentales aurait été de plus d'un an pour la troisième base de données. Ces résultats confirment donc l'intérêt de notre approche pour la découverte de DI relativement grandes.

Quatrième partie

Visualisation par l'exemple des dépendances découvertes

Chapitre 7

Etat de l'art

Sommaire

7.1 Bases de données d'Armstrong pour les DF et les DI

Étant donné un ensemble Σ de DF et de DI sur un schéma de base de données $\mathbf{R}$, une base de données d'Armstrong pour Σ est définie sur $\mathbf{R}$ et vérifie *exactement* les DF et DI induites par Σ.

Définition 30 (Base de données d'Armstrong) Soit $\mathbf{R}$ un schéma de base de données, $\Sigma = F \cup I$ un ensemble de DF et de DI définies sur $\mathbf{R}$, et une base de données $\mathbf{d}$ sur $\mathbf{R}$. $\mathbf{d}$ est une *base de données d'Armstrong* pour Σ si pour toute DF ou DI α sur $\mathbf{R}$, on a :

$$\mathbf{d} \models \alpha \Longleftrightarrow \Sigma \models \alpha$$

Une base de données d'Armstrong est donc une représentation alternative d'un ensemble de contraintes.

Dans le cas d'un ensemble vide de DI, on se ramène à la notion de *relations d'Armstrong* pour un ensemble de DF [Arm74]. Nous donnons dans la suite quelques propriétés et algorithmes de construction des relations et bases de données d'Armstrong.

7.1.1 Relations d'Armstrong pour un ensemble de DF

Une relation d'Armstrong existe pour tout ensemble de DF *standards* en entrée. Une DF non standard est un cas assez particulier de DF, possédant une partie gauche vide [FV83]. Il suffit alors que la partie droite ait *la même valeur pour tous les tuples*, ce qui peut toujours s'exprimer par une contrainte de cardinalité. Dans la suite, nous écrirons "DF" pour "DF standards".

Pour caractériser les relations d'Armstrong à partir de leurs contenus, et non à partir de leurs DF, la notion d'*ensembles en accord* a été définie [BDFS84] :

Définition 31 (Ensembles en accord d'une relation) Soit r une relation sur un schéma R. Soient t_1, t_2 deux tuples distincts de r et X un ensemble d'attributs de R. On dit que t_1 et t_2 sont *en accord* sur X, noté $ag(t_1, t_2) = X$ si $t_1[X] = t_2[X]$ et $\forall A \in R - X$, $t_1[A] \neq t_2[A]$. X est appelé un *ensemble en accord*. On note $ag(r)$ l'ensemble des ensembles en accord de r.

Les ensembles en accord d'une relation sont donc les ensembles d'attributs tels qu'il existe au moins deux tuples égaux sur lui. Notons que chaque ensemble en accord atteste, d'après la définition de la satisfaction d'une DF, de la violation d'un certain nombre de DF.

Exemple 27 Considérons la base de données **movies** (figure 3 page 8). Le troisième tuple et le huitième tuple de la relation *remakes* prennent les mêmes valeurs sur les attributs C et D. L'ensemble CD est donc un ensemble en accord de la relation *remakes*. L'existence de cet ensemble en accord atteste, par exemple, que la DF $CD \longrightarrow B$ est contredite.

Si l'on dispose de l'ensemble de tous les ensembles en accord d'une relation, on a donc une sorte de *représentation de tous les cas de DF violées* dans la base. Ils permettent de donner une caractérisation des relations d'Armstrong :

Théorème 4 *[BDFS84] Soit F un ensemble de DF sur un schéma R. Une relation r sur R est une relation d'Armstrong pour F si et seulement si*

$$GEN(F) \subseteq ag(r) \subseteq CL(F)$$

Ce résultat repose principalement sur deux constatations : les ensembles en accord d'une relation r sont des fermés pour F_r, et r sera une relation d'Armstrong pour F si et seulement si $F_r = F$, soit $GEN(F_r) = GEN(F)$.

Cette caractérisation a permis notamment de donner un algorithme (algorithme 9) pour la construction de relations d'Armstrong. Le principe est de construire dans la relation en construction des tuples en accord sur chaque générateur de F, tout en tâchant de ne pas générer d'autres ensembles en accords [BDFS84].

Algorithme 9 Relation d'Armstrong pour un ensemble de DF

Entrée: Un schéma de relation R, un ensemble de DF F sur R.
Sortie: Une relation d'Armstrong r pour F sur R.

Calculer $GEN(F)$ à partir de F ;
for all $A \in R$ **do**
  $t[A] = 0$;
end for
$r = \{t\}$;
$i = 1$;
for all $X \in GEN(F)$ **do**
  for all $A \in R$ **do**
    if $A \in X$ **then** $t[A] = 0$
    else $t[A] = i$;
  end for
  $r = r \cup \{t\}$;
  $i := i + 1$;
end for
Retourner r ;

Précisons que le nombre de générateurs peut être théoriquement exponentiel dans le nombre de DF [BDFS84] ; toutefois en pratique ce n'est généralement pas le cas. Un algorithme pour le calcul des générateurs à partir d'un ensemble de DF, que nous ne détaillons pas, est donné par exemple dans [MR94a]. Celui-ci utilise une méthode incrémentale, dans laquelle les DF sont ajoutées une à une. Le facteur essentiel déterminant la complexité de cet algorithme est le nombre de générateurs en sortie.

On constate que l'algorithme 9 est linéaire dans le nombre de générateurs et le nombre d'attributs. En outre, la taille de la relation construite sera toujours égale à $|GEN(F)|+1$.

Notons que la relation d'Armstrong ainsi construite n'a pas forcément une taille minimale, il peut être possible d'en construire une plus petite. A notre connaissance, aucun algorithme n'a été proposé pour obtenir une relation d'Armstrong minimale à partir d'un ensemble de DF ; toutefois, une borne inférieure a été donnée dans [BDFS84] :

Théorème 5 *Soit F un ensemble de DF, et r une relation d'Armstrong pour F. On a :*

$$|r| \geq \lceil \frac{1 + \sqrt{1 + 8 \times |GEN(F)|}}{2} \rceil$$

Cette borne correspond au nombre minimal de tuples qui doivent être présents dans r pour pouvoir former au moins autant de couples de tuples que de générateurs de F.

7.1.2 Prise en compte des DI

Une base de données d'Armstrong existe pour tout ensemble de DF et de DI en entrée, mais à notre connaissance il n'existe pas d'algorithme pour la construction de bases de données d'Armstrong dans le cas général. Des algorithmes existent pour des classes particulières de DF et de DI en entrée. Par exemple, [MR94b] donne un algorithme dans le cas où $F \cup I$ sont définis sur $\mathbf{R}$ avec $\mathbf{R}$ en forme normale de Boyce-Codd (FNBC) par rapport à F, et toutes les parties droites des DI dans I sont des clés. Cet algorithme n'étant pas utilisé dans la suite, nous ne donnerons pas plus de détails.

7.2 Relations d'Armstrong pour des relations existantes

La visualisation par l'exemple des DF satisfaites dans une relation existante a été étudiée dans [SM79, MR86b, LPL00]. Il s'agit d'une extension simple de la méthode présentée dans la section 7.1, permettant de prendre en compte l'existence d'une relation réelle.

Le principe est tout d'abord de définir le concept de *relation d'Armstrong pour une relation*. On rappelle que F_r dénote les DF satisfaites dans une relation r.

Définition 32 (Relation d'Armstrong pour une relation) Soit r une relation. Une relation $\overline{r}$ est une relation d'Armstrong pour r si $\overline{r}$ est une relation d'Armstrong pour F_r.

Afin de construire de telles relations, le principe proposé dans [LPL00] est de calculer les générateurs de F_r à partir de la relation r, en utilisant les ensembles maximaux de r, définis comme suit dans [MR86b].

Définition 33 (Ensembles maximaux d'un ensemble de DF) Soit F un ensemble de DF sur un schéma de relation R et $A \in R$ un attribut. Un *ensemble maximal* pour

F et A est un ensemble d'attributs $X \subseteq R$ qui ne détermine pas l'attribut A, et qui est maximal au sens de l'inclusion pour cette propriété. On note $MAX(F, A)$ l'ensemble des maximaux pour F et A et $MAX(F)$ l'ensemble des maximaux pour F défini par : $MAX(F) = \bigcup_{A \in R} MAX(F, A)$.

En fait, les générateurs et les maximaux d'un ensemble de DF constituent *exactement les mêmes ensembles*, soit : $GEN(F) = MAX(F)$ pour tout ensemble F de DF [MR86b]. Si on considère une relation existante r, il suffit donc de calculer $MAX(F_r)$ pour connaître $GEN(F_r)$. L'intérêt de la définition des maximaux est qu'elle permet d'établir une relation directe entre $MAX(F_r)$ et $ag(r)$, les ensembles en accord de r.

Proposition 9 *[LPL02] Soit r une relation sur un schéma R, et A un attribut de R. On a :*

$$max(F_r, A) = Max_{\subseteq}\{X \in ag(r) \mid A \notin X\}$$

Une solution pour calculer une relation d'Armstrong pour une relation r consiste donc à

1. Calculer les ensembles en accord de r ;
2. En déduire les ensembles maximaux (= générateurs) de F_r ;
3. appliquer l'algorithme 9 page 77.

La première étape est déterminante, puisqu'elle comporte les accès aux donnés permettant d'extraire les informations nécessaires de r. Cette étape a une complexité quadratique dans la taille de r, puisqu'il s'agit d'explorer tous les couples de tuples de r. Elle peut être effectuée par exemple au travers d'une requête SQL effectuant une auto-jointure sur r [LPL01].

La deuxième étape se compose d'un algorithme simple déduit de la proposition 9.

Enfin, on a vu que l'algorithme 9 prend des valeurs parmi les nombres entiers pour peupler la relation d'Armstrong ; en effet, aucun contexte n'était alors disponible. Dans [LPL00], les auteurs proposent de puiser les valeurs dans le domaine actif des attributs, afin de construire un exemple plus informatif.

Exemple 28 Considérons la relation *remakes* de la base de données **movies** représentée dans la figure 3. Les ensembles en accord de *remakes* sont :

$$ag(remakes) = \{ABC, CD, D, E, \emptyset\}$$

Puis les maximaux/générateurs de $F_{remakes}$:

$$MAX(F_{remakes}) = GEN(F_{remakes}) = \{ABC, CD, D, E\}$$

L'algorithme 9, légèrement transformé pour puiser les valeurs dans les domaines actifs, peut alors produire la relation d'Armstrong représentée dans la figure 7.1.

FIG. 7.1 – Une base de données d'Armstrong pour *remakes*

ID (A)	$TITLE$ (B)	$YEAR$ (C)	$FRAC$ (D)	$PRID$ (E)	$PRTITLE$ (F)
16	Le voyage	1959	0.9	11	Mademoiselle Fifi
16	Le voyage	1959	0.3	23	Cléopâtre
29	Cléopâtre	1959	0.9	17	Quatre soeurs
28	Des soeurs courageuses	1962	0.9	22	Gilda
15	Les amours de Carmen	1938	0.2	11	Le cercle du mariage

Une première limitation de cette méthode concerne l'existence même de telles relations d'Armstrong. En effet,l'algorithme 9 ne pourra produire une relation d'Armstrong $\overline{r}$ pour r peuplée avec des valeurs des domaines actifs des attributs de r que si [LPL00] :

$$\forall A \in R, |\pi_A(r)| \geq |\{X \in GEN(F_r) \mid A \notin X\}| + 1$$

En effet, un nombre suffisant de valeurs distinctes doivent être disponibles dans le domaine actif de chaque attribut.

Penchons nous maintenant sur l'intérêt même de telles relations d'Armstrong. Leur principale application est attendue dans la compréhension d'une base existante, et plus précisément dans la *compréhension des DF satisfaites dans une base existante*. Or, on constate dans l'exemple 28 que les tuples de la relation d'Armstrong construite sont complètement fictifs.

Supposons qu'un administrateur ait pour tâche de comprendre, puis maintenir la relation *remakes* de la base de données **movies** (figure 3 page 8). De celle-ci il ne possède que les noms de table et des attributs, et les tuples qui peuplent cette base. Un programme de fouille de données lui fournit une couverture des DF satisfaites dans la base : $F = \{B \longrightarrow A, A \longrightarrow BC, F \longrightarrow ABCDE, E \longrightarrow ABCDF, BD \longrightarrow ACEF\}$. Si l'administrateur désire comprendre pourquoi l'attribut $A(ID)$ n'est pas clé, et présente dans ce but la relation de la figure 7.1 à un expert du domaine, celui-ci ne pourra pas se prononcer car le tuple 2 est fictif. Or celui-ci participe à contredire le fait que A est une clé.

Les relations d'Armstrong informatives, définies dans le chapitre suivant, s'affranchissent de ces deux limitations.

Chapitre 8

Relations d'Armstrong informatives : visualisation des DF

Sommaire

8.1 Définitions et propriétés

Les relations d'Armstrong informatives sont définies de la façon suivante :

Définition 34 (Relations d'Armstrong Informatives (RAI)) Soit r et $\overline{r}$ deux relation sur un schéma $\mathbf{R}$. $\overline{r}$ est une relation d'Armstrong informative (RAI) pour r si :
– $\overline{r}$ est une relation d'Armstrong pour r et
– $\overline{r} \subseteq r$.

Intuitivement, il s'agit de garder un sous-ensemble d'une relation possédant exactement le même ensemble de DF. Tous les tuples seront alors issus du monde réel, évitant ainsi les problèmes de tuples fictifs.

Une première propriété concerne l'existence des RAI :

Propriété 4 *Il existe une RAI $\overline{r}$ pour toute relation r.*

En effet, il suffit de constater que toute relation est une RAI pour elle-même.

L'intérêt est maintenant de chercher à construire une RAI plus petite que la relation de départ. Tout d'abord, comme cela a été fait pour les relations d'Armstrong pour un ensemble de DF, on peut donner une caractérisation exacte des RAI à partir de leurs ensembles en accord.

Théorème 6 *Soit r et $\overline{r}$ deux relations sur un schéma $\boldsymbol{R}$, telles que $\overline{r} \subseteq r$. $\overline{r}$ est une relation d'Armstrong informative (RAI) pour r si et seulement si : $GEN(F_r) \subseteq ag(\overline{r})$.*

Preuve

($\Leftarrow$) *Puisque $GEN(F_r) \subseteq ag(\overline{r})$, il reste à s'assurer que $ag(\overline{r}) \subseteq CL(F_r)$ pour que $\overline{r}$ soit une relation d'Armstrong pour F_r (d'après le théorème 4 page 76). Or, $\overline{r} \subseteq r$ donc $ag(\overline{r}) \subseteq ag(r)$. Puisque $ag(r) \subseteq CL(F_r)$, on a $ag(\overline{r}) \subseteq CL(F_r)$.*

De plus, on a $\overline{r} \subseteq r$, donc $\overline{r}$ est une RAI pour r.

($\Rightarrow$) $\overline{r}$ est une relation d'Armstrong pour F_r, donc $GEN(F_r) \subseteq ag(\overline{r}) \subseteq CL(F_r)$. □

De manière informelle, $\overline{r}$ doit posséder au moins un couple de tuples en accord sur chaque ensemble générateur de F_r. Cette caractérisation conduit à l'existence de RAI avec une taille bornée.

Corollaire 1 *Soit r une relation sur un schéma $\boldsymbol{R}$. Il existe au moins une RAI $\overline{r}$ pour r telle que :*
– $|\overline{r}| \geq \lceil \frac{1+\sqrt{1+8\times M}}{2} \rceil$ *et*
– $|\overline{r}| \leq Min(2 \times M, |r|)$ *où* $M = |GEN(F_r)|$.

Preuve *Si on construit une relation $\overline{r}$ avec exactement 2 tuples en accord sur chaque générateur de F_r, la taille de $\overline{r}$ sera toujours inférieure à la borne supérieure du corollaire 1. La borne inférieure, donnée dans [BDFS84], reste valide dans ce cas, puisque $\overline{r}$ est une relation d'Armstrong pour F_r.* □

Précisons que le nombre $2 \times M$ est, en pratique, bien inférieur à la taille de r, ce qui est d'une grande importance pour l'intérêt des RAI.

Dans la suite, on appellera *petite RAI* une RAI dont la taille est bornée par le corollaire 1. Bien que notre problème est plus contraint que dans la définition classique des relations d'Armstrong, la taille maximale des petites RAI est du même ordre que les précédentes propositions [BDFS84, MR86b, DT95, LPL00], soit $2 \times |GEN(F_r)|$ au lieu de $|GEN(F_r)| + 1$.

Fig. 8.1 – Petite RAI pour la relation *remakes*

$\overline{remakes}$

ID (A)	$TITLE$ (B)	$YEAR$ (C)	$FRAC$ (D)	$PRID$ (E)	$PRTITLE$ (F)
16	Le voyage	1959	0.9	11	Mademoiselle Fifi
28	Des soeurs courageuses	1938	0.9	17	Quatre soeurs
16	Le voyage	1959	0.3	17	Quatres soeurs
12	Young at heart	1938	0.9	30	Casablanca

8.2 Construction de petites RAI

Un algorithme (algorithme 10) peut être déduit du théorème 6 et du corollaire 1, pour construire une petite RAI étant donnée une relation r et les générateurs de F_r en entrée.

Algorithme 10 Relations d'Armstrong informatives

Entrée: une relation r, $GEN(F_r)$;
Sortie: une petite RAI $\overline{r}$ pour r ;

$\overline{r} = \emptyset$
for all $X \in GEN(F_r)$ **do**
  Soit $t_1, t_2 \in r$ tels que $ag(t_1, t_2) = X$;
  $\overline{r} = \overline{r} \cup \{t_1\} \cup \{t_2\}$
end for
Retourner $\overline{r}$.

La complexité de cet algorithme est en $O(|GEN(F_r)| \times |r|^2)$. Notons que cette complexité est majorée par $O(|r|^4)$; en effet, on a vu plus haut que les générateurs de F_r sont une sous-famille des ensembles en accord de r, dont le nombre est majoré par $|r|^2$.

Exemple 29 Les générateurs de la relation *remakes* sont :$\{ABC, CD, D, E\}$.

L'algorithme 10 pourrait produire la RAI $\overline{remakes}$ de la figure 8.1.

L'intérêt des RAI relativement aux précédentes solutions apparaît alors ; un expert cinéphile pourra, à la demande par exemple de l'administrateur de la base, donner son avis sur certaines questions précises. Par exemple, dans le cas qui nous intéresse, l'expert en cinéma saura immédiatement confirmer que le film "Le voyage" est à la fois un remakes du film "Mademoiselle Fifi" et du film "Quatres soeurs", confirmant ainsi que $ID(A)$ n'est pas une clé de la relation.

Ou encore l'administrateur peut s'interroger sur le fait que $PRID$ ne détermine pas $PRTITLE$ dans la liste des dépendances qui lui sont proposées. Encore une fois, ce sera à l'expert du domaine de décider, à partir de la relation exemple, si les deux films "'Quatre soeurs"' et "Quatres soeurs" ne sont pas en fait les mêmes, à une erreur de saisie près.

Remarque sur le calcul des générateurs de F_r

Nous n'avons pas inclus dans l'algorithme 10 le calcul des générateurs, même si ceux si ne sont généralement pas une donnée connue a priori. Ce choix permet notamment de ne pas prendre en compte ce calcul dans l'estimation de la complexité de notre algorithme. Il s'agit en effet d'un problème connexe au notre, et plusieurs cas de figure peuvent se présenter :

1. Les DF ne sont pas connues. Dans ce cas, les générateurs doivent être extraits directement de la relation, comme cela est fait dans [LPL02], à l'aide des ensembles maximaux issus eux-mêmes des ensembles en accord (voir chapitre 7 page 75). Dans ce cas, le calcul des générateurs est polynomial dans le nombre de tuples de la relation.
2. Les DF satisfaites dans r sont connues à l'avance, par exemple à l'issue d'une phase de fouille de données dédié aux dépendances fonctionnelles (par exemple [FS99, HKPT99, LPL02, NC01, WGR01]). Des algorithmes [MR94a] permettent alors de calculer l'ensemble des générateurs de F_r à partir de F_r. La complexité de ce calcul dépend principalement du nombre de générateurs en sortie. Si dans certains cas pathologiques, ce nombre est exponentiel dans le nombre de DF, ce n'est toutefois pas le cas dans la plupart des situations réelles ; en outre, le calcul des générateurs se fait sans aucun accès aux données.

8.3 Nombre de petites RAI

L'algorithme 10 n'est pas déterministe ; les couples de tuples sont choisis au hasard pour chaque générateur. En général, un grand nombre de petites RAI peuvent être construites pour une relation donnée. Le nombre de petites RAI possibles que l'on peut construire est borné par le corollaire suivant.

Corollaire 2 *Soit une relation r, le nombre n de petites RAI pour r est borné par :*

$$1 \leq n \leq \prod_{X \in GEN(F_r)} |\{(t_1, t_2) \mid t_1, t_2 \in r, ag(t_1, t_2) = X\}|$$

Exemple 30 Pour chaque ensemble générateur de la relation *remakes*, plusieurs couples de tuples peuvent être choisis. Si on note t_i le i^{eme} tuple de *remakes* dans l'ordre d'apparition dans la figure 3, on a :

Pour ABC : (t_1, t_6) ou (t_7, t_9)
Pour CD : (t_3, t_8)
Pour D : (t_1, t_3), (t_1, t_5), (t_1, t_8), (t_2, t_6), (t_3, t_5) ou (t_5, t_8)
Pour E : (t_3, t_6)

Soit au total 12 possibilités. Certaines d'entre elles sont identiques, c'est pourquoi le corollaire 2 donne une borne supérieure et non pas un nombre exacte de RAI distinctes.

On peut construire par exemple :

$\overline{remakes}_1 = \{t_1, t_3, t_6, t_8\}$ ou
$\overline{remakes}_2 = \{t_3, t_5, t_6, t_7, t_8, t_9\}$

Remarquons que $\overline{remakes}_1$ atteind la borne inférieure donnée dans le corollaire 1.

Le nombre de combinaisons peut être très grand en pratique. Par exemple, pour la vraie relation *remakes*, nous verrons dans les expérimentations que le nombre de choix possibles est de 71.10^6 pour choisir les couples de tuples à insérer dans la RAI, alors que les DF satisfaites dans cette relation ont seulement 9 générateurs.

Or, toutes les petites RAI n'ont pas la même taille, comme le montre l'exemple 30. Un challenge intéressant est naturellement de construire la plus petite. Ce problème d'optimisation est considéré dans la section suivante.

8.4 Le problème de la RAI minimale

8.4.1 Définition du problème

Les relations d'Armstrong informatives minimales sont définies comme suit :

Définition 35 (Relations d'Armstrong informatives minimales) Soit une relation r ; $\overline{s}$ est une RAI minimale pour r s'il n'existe pas de RAI $\overline{r}$ pour r telle que $|\overline{r}| < |\overline{s}|$.

Cette définition soulève donc un nouveau problème d'optimisation combinatoire, formulé de la façon suivante :

Problème RAI_MIN : *Soit r une relation. Construire une RAI minimale $\overline{r}$ pour r.*

Pour faciliter la compréhension du problème, on peut le représenter sous la forme d'un graphe coloré $G = (V, E)$. Chaque tuple membre d'un couple en accord sur un des générateurs correspond à un élément de V. Une arête est placée entre deux tuples qui forment un couple en accord sur un générateur. Une couleur, correspondant au générateur en question, est affectée à l'arête. Il y a donc une couleur par générateur.

Exemple 31 Si on considère la relation *remakes*, dont les couples de tuples par générateur sont donnés dans l'exemple 30, on obtient le graphe représenté dans la figure 8.2.

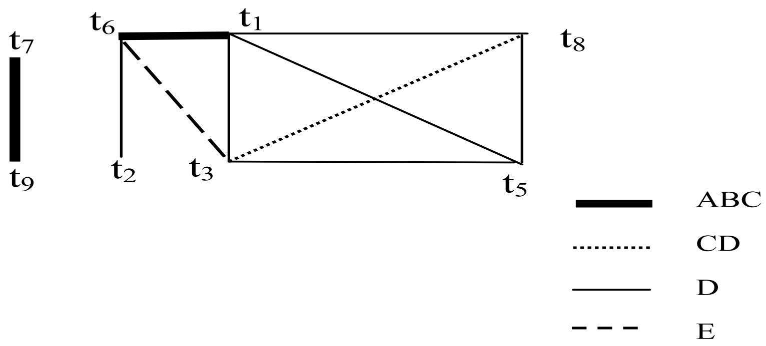

FIG. 8.2 – Graphe représentant les couples de tuples par générateur dans la relation *remakes*

Avec cette représentation, le problème est donc de trouver le plus petit sous-graphe (en nombre de sommets) couvrant toutes les couleurs.

8.4.2 Complexité du problème

Soit le problème de décision suivant :

Problème RAI_MIN_HYP : *Soit r une relation et k un entier. Existe-t-il une RAI* $\overline{r}$ *pour r telle que* $|\overline{r}| \leq k$ *?*

L'obtention d'une solution en temps polynomial pour le problème RAI_MIN conduit à l'obtention d'une solution en temps polynomial pour RAI_MIN_HYP, qui consisterait à construire une RAI minimale $\overline{r}$ pour r et retourner OUI si $|\overline{r}| \leq k$, NON sinon.

Nous avons alors cherché à cerner la complexité du problème RAI_MIN_HYP, et en particulier à savoir si ce problème est NP-complet. Afin d'exploiter la représentation du problème sous forme de graphe, comme l'illustre la figure 8.2, nous avons montré la NP-complétude du problème suivant :

Problème $MIN\_ARETE\_COULEUR$: Soit un graphe coloré $G = (V, E)$, et k un entier. Existe-t-il un sous-graphe $G' = (V', E')$ de G, tel que toutes les couleurs de G apparaissent dans G', et tel que $|V'| \leq k$?

Proposition 10 *Le problème* $MIN\_ARETE\_COULEUR$ *est NP-complet.*

Preuve $MIN\_ARETE\_COULEUR$ *est dans NP. En effet, étant donnée* $s \subseteq r$ *avec*

$|s| \leq k$, un algorithme peut vérifier en temps polynomial si pour chaque $X \in GEN(F_r)$ il existe $t_1, t_2 \in s$ tel que $ag(t1, t2) = X$.

On réduit alors le problème SAT [GJ79] au problème $MIN\_ARETE\_COULEUR$. Soit $F = \{C_1, \ldots, C_p\}$ une instance quelconque de SAT. Chaque clause est un ensemble de variables notées $C_i = \{x_{i,1}, ..., x_{i,n_i}\}$. On suppose que $n_i > 1, i = 1..p$.

On construit un graphe $G = (V, E)$ de la façon suivante : Pour chaque clause $C_i \in F$ et chaque variable $x_{i,j} \in C_i$, on crée dans V à sommet noté $[i, j]$. On crée ensuite une arête $([i, j], [i', j'])$ dans E si $i' \neq j'$ et $x_{i,j} \neq \overline{x}_{i',j'}$. On affecte la couleur (i, i') à l'arête $([i, j], [i', j'])$. On a ainsi une couleur pour chaque couple de clause, soit $p(p-1)/2$ couleurs. Enfin, on pose $k = p$ pour obtenir une entrée de $MIN\_ARETE\_COULEUR$. Cette transformation peut être faite en temps polynomial.

Supposons maintenant que $MIN\_ARETE\_COULEUR$ soit satisfiable. Il existe alors un sous-graphe $G' = (V', E')$ de G tel que $|V'| \leq p$, dans lequel toutes les couleurs de G apparaissent. Puisqu'il y a $= p(p-1)/2$ couleurs, E' contient exactement $p(p-1)/2$ arêtes. On en conclut que $|V'| = p$ et que chaque paire de sommets dans V' forme une arête dans E' (et ainsi G' est une clique). Pour chaque $[i, j] \in V'$, on prend alors la variable correspondante dans $x_{i,j}$ dans le but de construire une solution S pour F. On obtient un ensemble de p variables appartenant deux à deux à des clauses distinctes (il y a donc une variable pour chaque clause), et si $x_{i,j} \in S$ alors $\overline{x_{i,j}} \notin S$. Ainsi S est une solution pour F, et F est satisfiable.

Supposons que F est satisfiable, et qu'on peut donc trouver un ensemble S de p variables $x_{i,j}$, une pour chaque clause, tel que si $x_{i,j} \in S$ alors $\overline{x_{i,j}} \notin S$. Pour chaque $x_{i,j} \in S$, on prend le sommet correspondant $[i, j]$ dans V et on construit ainsi l'ensemble V'. Si on considère le sous-graphe $G' = (V', E')$ de G induit par V', c'est une solution pour le problème $MIN\_ARETE\_COULEUR$, et ainsi P est satisfiable. □

Nous avons inclus cette démonstration dans [DLP01] pour montrer que le problème de la construction d'une RAI minimale est NP-complet. Il s'avère en fait que le problème RAI_MIN_HYP n'est pas équivalent au problème $MIN\_ARETE\_COULEUR$. Si toute entrée de RAI_MIN_HYP peut permettre de construire un graphe en entrée de $MIN\_ARETE\_COULEUR$, comme l'illustre l'exemple 31, l'inverse n'est pas le cas. Certains graphes colorés ne peuvent pas être transformés en entrée de RAI_MIN_HYP, comme l'illustre l'exemple suivant.

Exemple 32

Soit le graphe représenté dans la figure 8.3.

Supposons que les ensembles X et Y correspondent aux générateurs distincts d'une relation r, et que t_1, t_2, t_3, t_4 sont des tuples de r tels que $ag(t_1, t_2) = ag(t_2, t_4) = ag(t_3, t_4) = X$ et $ag(t_1, t_3) = ag(t_2, t_3) = Y$. D'après la définition des ensembles en accord, on peut dé-

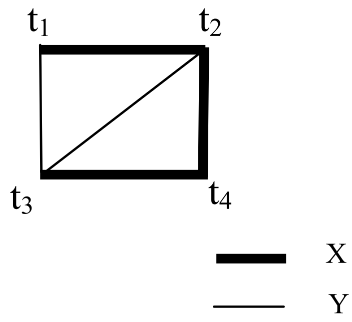

FIG. 8.3 – Un graphe coloré

duire que $t_2[X] = t_3[X]$, donc $Y \subseteq X$. On montre de même que $X \subseteq Y$. Ainsi, $X = Y$, ce qui contredit le fait que X et Y sont distincts.

Le problème RAI_MIN_HYP ne correspond donc dans sa représentation graphique qu'à des classes particulières de graphes colorés, qui restent à définir. Il s'agit donc d'un problème au pire $NP-complet$, mais dont la complexité exacte reste un problème ouvert.

8.4.3 Modélisation en PLNE

Dans cette section, nous allons donner une représentation du problème de la construction d'une RAI minimale sous la forme d'un programme linéaire en nombres entiers.

Considérons la représentation du problème sous la forme d'un graphe telle que décrite précédemment. A chaque sommet t_i dans V, on associe une variable entière $x_i \in \{0, 1\}$, et à chaque arête (t_i, t_j) dans E une variable $y_{c,k} \in \{0, 1\}$, telle que $c \in C$ identifie le générateur (la couleur) sur lequel t_i et t_j sont en accord, et k est un nombre identifiant l'arête de cette couleur. On note n_c le nombre d'arêtes sur une couleur c. Chaque variable prend la valeur 1 si l'arête ou le sommet correspondant est retenu dans la solution, 0 sinon. Les contraintes suivantes spécifient notre problème :

1. Contraintes triviales, pour spécifier que les variables ne peuvent prendre comme valeurs entières que 0 ou 1 :
$$\forall x_i, 0 \leq x_i \leq 1$$
$$\forall c, \forall k, 0 \leq y(c, k) \leq 1$$
2. Si une arête est retenue dans la solution, alors les deux sommets extrémités doivent être retenus. On a : si $y_{c,k}$ représente (t_i, t_j),
$$y(c, k) \leq \frac{1}{2}x_i + \frac{1}{2}x_j$$

3. toutes les couleurs doivent apparaître :

$$\forall c \in C, \sum_{k=1}^{n_c} y(c,k) \geq 1$$

Afin de minimiser le nombre de sommets dans la solution retenue, l'expression à minimiser est :

$$\sum_{i=1}^{|V|} x_i$$

Le nombre de variables total est donc $|V| + |E|$. Le nombre de contraintes est :

1. $|V| + |E|$ pour les contraintes triviales ;
2. $|E|$ pour les contraintes reliant les variables arêtes aux variables sommets ;
3. $|GEN(F_r)|$ pour les contraintes d'apparition des couleurs.

Nous avons utilisé ce programme pour générer des RAI minimales, à l'aide du logiciel CPlex 7.1. Les résultats expérimentaux sont présentés dans le chapitre 10.

8.5 Heuristique pour la construction de petites RAI

Une heuristique peut améliorer l'efficacité de l'algorithme 10 en terme de taille du résultat, tout en gardant l'idée de ne considérer *qu'une fois chaque générateur* pour obtenir un temps d'exécution intéressant.

L'heuristique utilisée est basée sur le principe de considérer les générateurs dans l'ordre "des plus contraints aux moins contraints", soit dans l'ordre croissant du nombre de couples associés. On traite ainsi en dernier les générateurs pour lesquels on a plus de chance d'avoir au moins déjà un tuple inséré dans la relation.

Le traitement de chaque générateur consiste en un parcours des couples associés, afin d'élire le meilleur en fonction de l'état courant de la relation en construction. Trois cas sont possibles :

- Aucun tuple n'est encore présent, il faut insérer un nouveau couple de tuples ;
- Un élément d'un couple est présent, il faut insérer un nouveau tuple ;
- Un couple est déjà entièrement présent dans la RAI en construction, aucun tuple n'est à ajouter.

Lorsque plusieurs couples ou tuples sont éligibles, un d'entre eux est alors choisi au hasard.

La mise en place de cette méthode nécessite donc la connaissance *a priori* des ensembles de couples en accord sur chaque générateur, puisque c'est leur nombre qui va déterminer l'ordre de considération des générateurs. Il faut pouvoir également les parcourir efficacement, pour chaque générateur.

Exemple 33

FIG. 8.4 – Couples en accord sur chaque générateur pour *remakes*

$GENERATEUR$	$tuple1$	$tuple2$
ABC	t_1	t_6
ABC	t_7	t_9
CD	t_3	t_8
D	t_1	t_3
D	t_1	t_5
D	t_1	t_8
D	t_2	t_6
D	t_3	t_5
D	t_5	t_8
E	t_3	t_6

Appliquée à la relation *remakes*, dont les couples par générateurs sont donnés dans la figure 8.4, l'heuristique commence par considérer les générateurs CD puis E (le choix entre les deux est fait au hasard) puisqu'un seul couple leur est associé. Les tuples t_3, t_6 et t_8 sont donc insérés dans la RAI en construction. Ensuite, le générateur ABC est considéré : le tuple t_6 étant déjà dans la RAI, le couple (t_1, t_6) est sélectionné, et ainsi le tuple t_1 est inséré. Enfin, pour le générateur D, on constate que le couple (t_1, t_3) est déjà présent dans la RAI en construction, donc aucun nouveau tuple ne doit être inséré.

La relation construite est donc :

ID (A)	$TITLE$ (B)	$YEAR$ (C)	$FRAC$ (D)	$PRID$ (E)	$PRTITLE$ (F)
16	Le voyage	1959	0.9	11	Mademoiselle Fifi
28	Des soeurs courageuses	1938	0.9	17	Quatre soeurs
16	Le voyage	1959	0.3	17	Quatres soeurs
12	Young at heart	1938	0.9	30	Casablanca

L'heuristique aboutit dans cet exemple simple à une RAI minimale pour *remakes* ; ce n'est pourtant pas le cas en général, puisqu'aucune remise en question des choix réalisés n'est effectuée lors du traitement de nouveaux générateurs, et qu'une part non négligeable du choix est laissée au hasard.

Ajoutons pour finir que l'utilisation de l'heuristique n'est pas toujours possible ; en effet, le nombre de couples en accord sur les générateurs de r, qui est majoré par $O(|r|^2)$, est parfois tel qu'il est impossible de les stocker efficacement. C'est par exemple le cas lorsque des attributs impliqués dans les générateurs ont peu de valeurs distinctes dans leur projection.

8.6 Discussion sur la présence de valeurs NULL

La présence de valeurs NULL n'est pas un obstacle à la construction des RAI. Elles peuvent apparaître ou non dans la relation.

Soit r une relation sur un schéma R, et considérons deux tuples $t, u \in r$ tels que $ag(t, u) = X, X \in GEN(F_r)$. S'il existe $A \in R \setminus X$ tel que $t[A]$ est $NULL$, alors cette valeur NULL apparaîtra dans la relation en construction, en cas de sélection du couple (t, u). On peut d'ailleurs imaginer que la présence de valeur NULL ou non participe à la sélection des tuples, selon le choix de l'utilisateur. En effet, si une relation possède beaucoup de valeurs NULL, il peut être intéressant que l'échantillon qui la représente en possède aussi pour être plus fidèle.

Une question annexe intéressante qui se pose est de savoir si on peut avoir $ag(t, u) = X$ si $A \in X$, $t[A]$ ou $u[A]$ étant NULL. Deux tuples peuvent-ils être en accord sur une valeur NULL ? L'un ou l'autre choix est mis en oeuvre par deux variantes simples de la requête pour la découverte des ensembles en accord. Il s'agit donc d'un choix à faire en amont sur le problème plus général de la découverte des DF satisfaites dans une base, une option par exemple proposée à l'analyste ou à l'administrateur sur la gestion des valeurs NULL :

- Une valeur NULL ne contredit aucune DF : dans ce cas, aucun ensemble en accord n'est généré à partir d'une valeur NULL. Ceci revient à choisir que NULL est différent de toute autre valeur, même de NULL. Cette démarche possède un double avantage : elle n'exclut pas de DF dans le doute, et le nombre d'ensembles en accord est réduit. En contrepartie, le nombre de DF satisfaites sera plus important, augmentant ainsi les DF satisfaites "par hasard".
- Une valeur NULL peut contredire une DF, auquel cas NULL est égal à tout. L'avantage est d'avoir un nombre réduit de DF satisfaites, et d'éviter au maximum le bruit dans les résultats. Mais un tuple possédant une valeur NULL sera en accord avec tous les autres tuples de la base... le nombre d'ensembles en accord risque de devenir énorme, risquant de faire augmenter également le nombre de générateurs (bien qu'il n'y ait pas de relation claire entre ces deux nombres). En outre, une seule valeur NULL risque de contredire de nombreuses DF qui auraient pu être intéressantes.

On peut alors imaginer une solution intermédiaire, qui consisterait à laisser choisir à l'utilisateur la conduite à tenir pour chaque attribut. Une perspective plus fine pourrait être de différencier les valeurs NULL suivant plusieurs types (voir notamment [Zan84]), selon par exemple qu'elles sont manquantes ou inexistantes.

Chapitre 9

Les bases de données d'Armstrong informatives : visualisation des DF et des DI

Sommaire

Ce chapitre étend la notion de relations d'Armstrong informatives présentée dans la partie précédente. On considère maintenant en entrée non pas les relations une à une, mais des bases de données dans leur ensemble. Si le but recherché est uniquement la préservation des DF, alors il suffit d'appliquer la proposition précédente à chacune des relations de la base de données considérée. Une telle démarche ne serait toutefois pas satisfaisante, car l'échantillon construit ne refléterait pas la sémantique des liens entre relations, qui est véhiculée par les DI. Des algorithmes ont été proposés dans la partie III pour découvrir les DI satisfaites dans une base de données ; il s'agit maintenant de construire un échantillon de la base sans altérer ces DI satisfaites tout en gardant l'ensemble des DF satisfaites.

Dans un premier temps, nous définissons les bases de données d'Armstrong pour les bases de données, puis étendons cette définition aux bases de données d'Armstrong informatives (BDAI). Dans le cas d'un ensemble de DI non circulaire, l'existence d'une BDAI avec une taille bornée sera montrée.

Dans l'ensemble de ce chapitre, nous noterons $\mathbf{d} = \{r_1, ..., r_n\}$ une base de données sur un schéma $\mathbf{R} = \{R_1, ..., R_n\}$, $F_{\mathbf{d}}$ et $I_{\mathbf{d}}$ sont respectivement l'ensemble des DF et DI satisfaites dans $\mathbf{d}$.

9.1 Bases de données d'Armstrong pour des bases de données

La définition d'une base de données d'Armstrong pour une base de données étend celle d'une base de données d'Armstrong pour un ensemble de DF et de DI. La différence provient du fait que les DF et les DI concernées sont l'ensemble des DF et des DI satisfaites dans une base de données de départ.

Définition 36 (Base de données d'Armstrong pour une base de données) Une base de données d'Armstrong pour $\mathbf{d}$ est une base de données $\overline{\mathbf{d}}$ sur $\mathbf{R}$ telle que $\overline{\mathbf{d}}$ est une base de données d'Armstrong pour $F_{\mathbf{d}} \cup I_{\mathbf{d}}$.

Il s'agit donc d'une représentation alternative des dépendances satisfaites dans une base de données.

Dans le cas général de construction de bases de données d'Armstrong pour un ensemble $F \cup I$ de dépendances, on est confronté aux problèmes d'interactions entre F et I. Cette notion d'interaction est définie dans la section 1.2.4 page 22 : rappelons que dans le cas général, le problème d'inférence de contraintes à partir d'un ensemble de DF et de DI est indécidable.

Le problème qui se pose est donc le suivant : comment construire une base de données satisfaisant exactement des ensembles F et I, puisque il n'est pas possible de savoir quelles nouvelles dépendances vont découler de la réunion de ces ensembles ?

Nous allons voir que ce problème n'en est en fait pas un dans notre contexte. En effet, les ensembles $F_{\mathbf{d}}$ et $I_{\mathbf{d}}$ que nous possédons en entrée sont des couvertures de *toutes les dépendances satisfaites dans une base existante*. Aucune interaction n'est donc possible entre eux, comme le formalise la proposition suivante.

Proposition 11 *Une base de données $\overline{\mathbf{d}}$ est une base de données d'Armstrong pour $\mathbf{d}$ si et seulement si $F_{\overline{\mathbf{d}}} = F_{\mathbf{d}}$ et $I_{\overline{\mathbf{d}}} = I_{\mathbf{d}}$.*

Preuve *Il suffit de montrer que $F_{\mathbf{d}}$ et $I_{\mathbf{d}}$ n'ont pas d'interaction. Soit α une dépendance (DI ou DF) telle que $F_{\mathbf{d}} \cup I_{\mathbf{d}} \models \alpha$. Puisque $d \models F_{\mathbf{d}}$ et $d \models I_{\mathbf{d}}$, on a $d \models \alpha$. Puisque $F_{\mathbf{d}}$ et $I_{\mathbf{d}}$ sont les ensembles de* toutes *les DF et DI satisfaites dans d, on a $F_{\mathbf{d}} \models \alpha$ ou $I_{\mathbf{d}} \models \alpha$. Donc $F_{\mathbf{d}}$ et $I_{\mathbf{d}}$ n'ont pas d'interaction.* □

A l'instar des relations d'Armstrong pour les relations, une solution pour construire de telles bases de données serait d'appliquer à $F_{\mathbf{d}}$ et $I_{\mathbf{d}}$ les algorithmes existants [MR94b]. Mais là encore cette solution serait limitée dans son caractère de représentation de la base initiale, car les tuples seraient fictifs. C'est pourquoi nous étendons maintenant cette notion à celle, plus contrainte, de base de données d'Armstrong informative.

9.2 Bases de données d'Armstrong informatives

Une base de données d'Armstrong informative (BDAI) est une base de données d'Armstrong pour une base de données, mais avec une contrainte supplémentaire : tous les tuples sont issus de la base de données de départ, plutôt que d'être construits arbitrairement avec les valeurs des domaines actifs.

Dans la suite, nous étendons la définition classique de l'opérateur d'inclusion $\subseteq$ aux bases de données : Soit $\mathbf{d} = \{r_1, ..., r_n\}$ et $\overline{\mathbf{d}} = \{\overline{r}_1, ..., \overline{r}_n\}$ deux bases de données sur le même schéma. Nous supposons, sans perte de généralité, que les schémas de relation sont classés dans un certain ordre (ici, déterminé par l'indice). On dit que $\overline{\mathbf{d}}$ est incluse dans $\mathbf{d}$, noté $\overline{\mathbf{d}} \subseteq \mathbf{d}$, si pour chaque $\overline{r}_i \in \overline{\mathbf{d}}$, $\overline{r}_i \subseteq r_i$.

Définition 37 (Bases de données d'Armstrong informatives) Une base de données d'Armstrong informative (BDAI) $\overline{\mathbf{d}}$ pour $\mathbf{d}$ est telle que :
- $\overline{\mathbf{d}}$ est une base de données d'Armstrong pour $\mathbf{d}$ et
- $\overline{\mathbf{d}} \subseteq \mathbf{d}$.

Ainsi, l'idée est de mettre en évidence un sous-ensemble, pour chaque relation dans $\mathbf{d}$, satisfaisant exactement les mêmes DF et DI que la base de départ. Les tuples dans la BDAI refléteront alors une grande partie de la sémantique des données initiales, tout en gardant une signification cohérente dans le "monde réel" sous-jacent.

L'existence de tels sous-ensembles est assurée puisque toute base de données est une BDAI pour elle-même. Toutefois, l'intérêt est de construire des BDAI dont la cardinalité est inférieure à celle de la base de départ. Nous donnons dans la suite un algorithme pour construire de petites BDAI.

9.3 Construction de petites BDAI

D'après ce qui précède (proposition 11 et définition 37), le problème à traiter peut s'énoncer ainsi : "Construire *un sous-ensemble* $\overline{\mathbf{d}}$ de $\mathbf{d}$ tel que $F_{\overline{\mathbf{d}}} = F_{\mathbf{d}}$ et $I_{\overline{\mathbf{d}}} = I_{\mathbf{d}}$."

Dans la suite, nous supposons avoir construit une RAI pour chacune des relations de

$\mathbf{d}$. On dispose ainsi d'une base de données que nous appellerons $\dot{\mathbf{d}}$, telle que

$$\dot{\mathbf{d}} \subseteq \mathbf{d} \text{ et } F_{\dot{\mathbf{d}}} = F_{\mathbf{d}}$$

Nous supposons également que la découverte des DI a été effectuée par les méthodes présentées plus haut, et donc que l'on dispose de $\mathcal{B}d^+(I)$ et $\mathcal{B}d^-(I)$, respectivement la bordure positive et la bordure négative des DI ordonnées à gauche satisfaites dans $\mathbf{d}$ (voir le chapitre 4.1 page 37 pour plus de détails sur ces ensembles).

La démarche générale de la méthode proposée est décrite dans l'algorithme 11.

Algorithme 11 Base de données d'Armstrong informative

Entrée: $\mathbf{d}$, $\dot{\mathbf{d}}$, $\mathcal{B}d^+(I)$ et $\mathcal{B}d^-(I)$;
Sortie: une BDAI $\overline{\mathbf{d}}$ pour $\mathbf{d}$;
1: $\hat{\mathbf{d}} = InvalidIND(\dot{\mathbf{d}}, \mathcal{B}d^-(I), \mathbf{d})$; – *prise en compte des DI non satisfaites*
2: $\overline{\mathbf{d}} = ValidIND(\hat{\mathbf{d}}, \mathcal{B}d^+(I), \mathbf{d})$; – *prise en compte des DI satisfaites*
3: **Retourner** $\overline{\mathbf{d}}$.

Ainsi, l'algorithme se décompose en deux étapes, ayant pour objectif final que $\overline{\mathbf{d}}$ satisfasse toutes les DI de $\mathcal{B}d^+(I)$, et aucune autre. Chacune des étapes consiste à rajouter des tuples de $\mathbf{d}$ dans la base en construction, ainsi : $\dot{\mathbf{d}} \subseteq \hat{\mathbf{d}} \subseteq \overline{\mathbf{d}} \subseteq \mathbf{d}$.

Nous donnons dans la suite le détail de chacune des étapes, ainsi que les propriétés nécessaires assurant la correction de l'algorithme.

9.3.1 Cas des DF satisfaites et non satisfaites

La base de données $\dot{\mathbf{d}} \subseteq \mathbf{d}$ en entrée de l'algorithme 11 satisfait la propriété $F_{\dot{\mathbf{d}}} = F_{\mathbf{d}}$. La proposition suivante assure que cette propriété est conservée dans la suite de l'algorithme, quelque soient les nouveaux tuples de $\mathbf{d}$ que l'on insère dans $\dot{\mathbf{d}}$.

Proposition 12 *Pour toute base de données $\hat{\mathbf{d}}$ telle que $\dot{\mathbf{d}} \subseteq \hat{\mathbf{d}} \subseteq \mathbf{d}$, on a $F_{\hat{\mathbf{d}}} = F_{\mathbf{d}}$.*

Preuve *Soit $\dot{\mathbf{d}} = \{\dot{r}_1, ..., \dot{r}_n\} \subseteq \mathbf{d} = \{r_1, ..., r_n\}$ telle que $\forall i \in \{1, ..., n\}$, $\dot{r}_i$ est une RAI pour r_i. D'après le théorème 6 page 82, $GEN(F_{r_i}) \subseteq ag(\dot{r}_i)$.*

Soit $\dot{r}_i \subseteq \hat{r}_i \subseteq r_i$. On a forcément $ag(\dot{r}_i) \subseteq ag(\hat{r}_i)$, donc $GEN(F_{r_i}) \subseteq ag(\hat{r}_i)$.

Puisque $\hat{r}_i \subseteq r_i$, $\hat{r}_i$ est une RAI pour r_i, toujours d'après le théorème 6. □

Exemple 34 La figure 9.1 représente la base de données $\dot{\textbf{movies}}$, composée de relations d'Armstrong informatives pour chacune des relations de la base de données **movies** (figure 3 page 8). Ces relations sont construites à partir des ensembles de générateurs suivants : $\{ABC, CD, D, E\}$ pour *remakes* et $\{H, I\}$ pour *movies*.

FIG. 9.1 – Base de données **movies**, telle que $F_{\dot{\mathbf{movies}}} = F_{\mathbf{movies}}$.

$\dot{remakes}$

ID (A)	*TITLE* (B)	*YEAR* (C)	*FRAC* (D)	*PRID* (E)	*PRTITLE* (F)
16	Le voyage	1959	0.9	11	Mademoiselle Fifi
28	Des soeurs courageuses	1938	0.9	17	Quatre soeurs
16	Le voyage	1959	0.3	17	Quatres soeurs
12	Young at heart	1938	0.9	30	Casablanca

$\dot{movies}$

FILMID (G)	*TITLE* (H)	*YEAR* (I)
12	Young at heart	1938
23	Cléopâtre	1934
28	des soeurs courageuses	1938
29	Cléopâtre	1962

Cette base de données vérifie la propriété $F_{\dot{\mathbf{movies}}} = F_{\mathbf{movies}}$. D'après la proposition 12, cette propriété reste satisfaite quelque soient les tuples de **movies** que l'on rajoute dans $\dot{\mathbf{movies}}$.

9.3.2 Les DI non satisfaites

La ligne 1 de l'algorithme 11 a pour but de contredire les DI non satisfaites dans $\mathbf{d}$ afin de construire $\hat{\mathbf{d}}$ telle que $\dot{\mathbf{d}} \subseteq \hat{\mathbf{d}} \subseteq \mathbf{d}$ et $I_{\hat{\mathbf{d}}} \subseteq I_{\mathbf{d}}$. Pour contredire les DI, nous utiliserons la notion de *tuple disqualifiant* définie par :

Définition 38 (tuple disqualifiant) Soit $R_i[X] \subseteq R_j[Y]$ une DI sur $\mathbf{R}$. Un tuple $t \in r_i$ est un *tuple disqualifiant* pour $R_i[X] \subseteq R_j[Y]$ si $t[X] \notin \pi_Y(r_j)$.

Ainsi, pour construire $\hat{\mathbf{d}}$, il faut insérer dans $\dot{\mathbf{d}}$ un seul tuple disqualifiant pour chaque DI non satisfaite. En fait, grâce à la propriété d'anti-monotonie des DI, il suffit de contredire la couverture négative $\mathcal{B}d^-(I)$ des DI satisfaites dans $\mathbf{d}$. Cette démarche conduit à l'algorithme 12.

Pour chaque DI de $\mathcal{B}d^-(I)$, l'algorithme 12 cherche si un tuple disqualifiant appartient à la base de données en construction (ligne 3). Si ce n'est pas le cas, un tel tuple est choisi pour cette DI puis inséré dans $\hat{\mathbf{d}}$ (lignes 4 et 5). La complexité de cet algorithme est linéaire dans la taille de $\mathcal{B}d^-(I)$.

Algorithme 12 *InvalidIND*

Entrée: $\mathbf{d}$, $\mathcal{B}d^-(I)$, $\dot{\mathbf{d}}$;
Sortie: $\hat{\mathbf{d}}$ telle que $\hat{\mathbf{d}} \subseteq \mathbf{d}$, $F_{\hat{\mathbf{d}}} = F_{\mathbf{d}}$, $I_{\hat{\mathbf{d}}} \subseteq I_{\mathbf{d}}$;

```
d̂ = ḋ ;
for all i = R_i[X] ⊆ R_j[Y] ∈ Bd⁻(I) do
   if (π_X(r̂_i) − π_Y(r_j)) = ∅ then
      Trouver un tuple disqualifiant t pour i ;
      r̂_i = r̂_i ∪ {t} ;
   end if
end for
Retourner d̂.
```

Exemple 35 Sur l'exemple courant, la base de données $\hat{\mathbf{movies}}$ est d'abord initialisée à $\dot{\mathbf{movies}}$ représentée dans la figure 9.1.

$\mathcal{B}d^-(I_{\mathbf{movies}})$ est composée de 12 DI, que nous ne détaillons pas (10 DI unaires et 2 DI de taille 2). Considérons plus particulièrement deux cas significatifs :
– pour la DI $A \subseteq E \in \mathcal{B}d^-(I_{\mathbf{movies}})$, l'ensemble $\pi_A(\hat{remakes}) - \pi_E(remakes)$ n'est pas vide, ce qui signifie qu'un tuple disqualifiant pour cette DI est déjà dans $\hat{remakes}$.
– pour la DI $H \subseteq B \in \mathcal{B}d^-(I_{\mathbf{movies}})$, $\pi_H(\hat{movies}) - \pi_B(remakes)$ est vide, et un tuple disqualifiant pour cette DI doit être choisi dans $movies$, par exemple $< 11, MademoiselleFifi, 1944 >$, et est inséré dans $\hat{movies}$.
Cette dernière DI est la seule restant à contredire dans la base en construction, et ainsi l'algorithme 12 pourrait construire la base de données $\hat{\mathbf{movies}}$ représentée dans la figure 9.2, telle que $\hat{\mathbf{movies}} \subseteq \mathbf{movies}$, $F_{\hat{\mathbf{movies}}} = F_{\mathbf{movies}}$ et $I_{\hat{\mathbf{movies}}} \subseteq I_{\mathbf{movies}}$.

La proposition suivante assure que les propriétés de $\hat{\mathbf{d}}$ seront conservées dans la suite de l'algorithme, quelque soient les tuples de $\mathbf{d}$ que l'on insère dans $\hat{\mathbf{d}}$:

Proposition 13 *Pour toute base de données $\overline{\mathbf{d}}$ telle que $\hat{\mathbf{d}} \subseteq \overline{\mathbf{d}} \subseteq \mathbf{d}$, on a $F_{\overline{\mathbf{d}}} = F_{\mathbf{d}}$ et $I_{\overline{\mathbf{d}}} \subseteq I_{\mathbf{d}}$.*

Preuve $F_{\overline{\mathbf{d}}} = F_{\mathbf{d}}$ *provient de la proposition 12. Puisque $\hat{\mathbf{d}} \subseteq \overline{\mathbf{d}}$, il y a dans $\overline{\mathbf{d}}$ au moins un tuple disqualifiant pour chaque DI de $\mathcal{B}d^-(I)$, et ainsi $I_{\overline{\mathbf{d}}} \subseteq I_{\mathbf{d}}$.* □

9.3.3 Les DI satisfaites

La ligne 2 de l'algorithme 11 a pour but de forcer les DI satisfaites dans $\mathbf{d}$ afin de construire une base de données $\overline{\mathbf{d}}$ telle que $\hat{\mathbf{d}} \subseteq \overline{\mathbf{d}} \subseteq \mathbf{d}$, $\overline{\mathbf{d}}$ étant une BDAI pour $\mathbf{d}$, i.e. $F_{\overline{\mathbf{d}}} = F_{\mathbf{d}}$ et $I_{\overline{\mathbf{d}}} = I_{\mathbf{d}}$.

Le principe est de considérer une par une chaque DI de $\mathcal{B}d^+(I)$, et de tester si elle est

FIG. 9.2 – Base de données $\hat{\mathbf{movies}}$, telle que $F_{\hat{\mathbf{movies}}} = F_{\mathbf{movies}}$ et $I_{\hat{\mathbf{movies}}} \subseteq I_{\mathbf{movies}}$.

$\hat{remakes}$

ID (A)	*TITLE* (B)	*YEAR* (C)	*FRAC* (D)	*PRID* (E)	*PRTITLE* (F)
16	Le voyage	1959	0.9	11	Mademoiselle Fifi
28	Des soeurs courageuses	1938	0.9	17	Quatre soeurs
16	Le voyage	1959	0.3	17	Quatres soeurs
12	Young at heart	1938	0.9	30	Casablanca

$\hat{movies}$

FILMID (G)	*TITLE* (H)	*YEAR* (I)
12	Young at heart	1938
23	Cléopâtre	1934
28	des soeurs courageuses	1938
29	Cléopâtre	1962
11	Mademoiselle Fifi	1944

satisfaite ou non dans la base en construction. Si ce n'est pas le cas, elle doit être "forcée" dans la base en construction, à l'aide de la fonction $ForceDI$ (algorithme 13).

Algorithme 13 $ForceDI$: force la satisfaction d'une DI

Entrée: $\mathbf{d}$, $\overline{\mathbf{d}} \subseteq \mathbf{d}$ et $i = R_i[X] \subseteq R_j[Y] \in \mathcal{B}d^+(I)$;
Sortie: $\overline{\mathbf{d}}$ telle que $\overline{\mathbf{d}} \subseteq \mathbf{d}$ et $\overline{\mathbf{d}} \models i$;
1: **for all** $v \in \{\pi_X(\overline{r}_i) - \pi_Y(\overline{r}_j)\}$ **do**
2: Soit $t \in r_j$ tel que $t[Y] = v$;
3: $\overline{r}_j = \overline{r}_j \cup \{t\}$;
4: **end for**
5: **Retourner** $\overline{\mathbf{d}}$

On doit maintenant se poser la question suivante : existe-t-il un ordre pour forcer les DI, de telle sorte qu'elles ne soient considérées qu'une fois chacune ? Dans la suite, nous considérons que l'ensemble $\mathcal{B}d^+(I)$ est non circulaire, avant d'aborder le cas de DI circulaires.

Supposons qu'une DI $i = R_i[X] \subseteq R_j[Y]$ ait été forcée dans $\overline{\mathbf{d}}$. Puisque l'on ne supprime jamais de tuple, la seule façon de contredire i est d'ajouter des tuples dans la partie gauche, soit dans $\overline{r}_i$. Cette action n'aura lieu que si une DI entrante dans R_i a été forcée. Ainsi, si un schéma de relation R_i n'a pas de DI entrante dans $\mathcal{B}d^+(I)$, chaque DI allant de R_i vers un autre schéma de relation peut être définitivement forcée.

L'algorithme 14 est basé sur ce principe. Si R est un schéma de relation, on note $\mathcal{B}d^+(I)_{\rightarrow R}$ l'ensemble des DI de $\mathcal{B}d^+(I)$ qui entrent dans R, et $\mathcal{B}d^+(I)_{R\rightarrow}$ l'ensemble des DI de $\mathcal{B}d^+(I)$ qui sortent de R.

Algorithme 14 $ValidIND$

Entrée: $\mathbf{d}$,$\hat{\mathbf{d}}$ et $\mathcal{B}d^+(I)$.
Sortie: $\overline{\mathbf{d}}$, BDAI pour $\mathbf{d}$;

1: $\overline{\mathbf{d}} = \hat{\mathbf{d}}$;
2: **for all** R tel que $\mathcal{B}d^+(I)_{\rightarrow R} = \emptyset$ **do**
3:     **for all** $i \in \mathcal{B}d^+(I)_{R\rightarrow}$ **do**
4:         $\overline{\mathbf{d}} = ForceDI(\mathbf{d}, \overline{\mathbf{d}}, i)$;
5:         $\mathcal{B}d^+(I) = \mathcal{B}d^+(I) - \{i\}$;
6:     **end for**
7: **end for**
8: **Retourner** $\overline{\mathbf{d}}$.

Les schémas de relation sans DI entrantes doivent être considérés en premier (ligne 2) : les DI qui en sortent sont forcées (ligne 4) et peuvent être retirées de la couverture des DI (ligne 6). L'action se répète jusqu'à ce qu'aucune DI ne reste dans $\mathcal{B}d^+(I)$, ce qui est assuré puisque $\mathcal{B}d^+(I)$ est non circulaire. L'algorithme est globalement linéaire dans la taille de $\mathcal{B}d^+(I)$, au calcul près de $\mathcal{B}d^+(I)_{\rightarrow R}$ et $\mathcal{B}d^+(I)_{R\rightarrow}$ qu'on peut raisonnablement négliger.

Exemple 36 Dans notre exemple, la base de données $\overline{\textbf{movies}}$ est d'abord initialisée à $\widehat{\textbf{movies}}$, représentée dans la figure 9.2. On rappelle que $\mathcal{B}d^+(I_{\textbf{movies}})$ est composée de 2 DI : $\mathcal{B}d^+(I_{\textbf{movies}}) = \{ABC \subseteq GHI, E \subseteq G\}$. La première relation considérée est donc la relation *remakes* puisqu'elle n'a pas de DI entrante. On force d'abord $ABC \subseteq GHI$ en insérant le tuple $< 16, \text{Le voyage}, 1959 >$ dans $\overline{movies}$, puis la DI $E \subseteq G$ en insérant les tuples $< 17, \text{Quatre soeurs}, 1938 >$ et $< 30, \text{casablanca}, 1944 >$ dans $\overline{movies}$.

Les deux DI sont alors supprimées, et l'algorithme se termine. On obtient finalement la base de données $\overline{\textbf{movies}}$, BDAI pour $\mathbf{d}$, représentée dans la figure 9.3.

La complexité de la construction d'une BDAI par l'algorithme 11 est donc linéaire dans le nombre $(|GEN(F_{\mathbf{d}})| + |\mathcal{B}d^-(I)| + |\mathcal{B}d^+(I)|)$).

Cas d'un ensemble de DI circulaire

La méthode est élaborée de telle façon que chaque étape soit définitive, chaque élément de $\mathcal{B}d^-(I)$ et $\mathcal{B}d^+(I)$ n'étant considéré qu'une seule fois. Toutefois, dans le cas où $\mathcal{B}d^+(I)$ est circulaire, cette démarche ne semble plus possible. Forcer une DI entrante dans une relation R peut amener à contredire une autre DI sortante de R, elle même forcée auparavant. Dans ce cas, nous n'avons d'autre méthode que de traiter en boucle les cycles jusqu'à obtenir un état stable. Il est impossible dans ce cas de prévoir le nombre de fois que les DI seront parcourues.

FIG. 9.3 – Base de données $\overline{\textbf{movies}}$, BDAI de **movies**.

$\overline{remakes}$

ID (A)	*TITLE* (B)	*YEAR* (C)	*FRAC* (D)	*PRID* (E)	*PRTITLE* (F)
16	Le voyage	1959	0.9	11	Mademoiselle Fifi
28	Des soeurs courageuses	1938	0.9	17	Quatre soeurs
16	Le voyage	1959	0.3	17	Quatres soeurs
12	Young at heart	1938	0.9	30	Casablanca

$\overline{movies}$

FILMID (G)	*TITLE* (H)	*YEAR* (I)
12	Young at heart	1938
23	Cléopâtre	1934
28	Des soeurs courageuses	1938
29	Cléopâtre	1962
11	Mademoiselle Fifi	1944
16	Le voyage	1959
17	Quatre soeurs	1938
30	Casablanca	1944

Rappelons toutefois que l'algorithme se termine à coup sûr, au pire en obtenant la base de départ. Ce qui n'est pas le cas pour le problème moins contraint de la construction de bases de données d'Armstrong "classiques", pour lequel, à notre connaissance, aucun algorithme n'existe dans le cas de DI circulaires en entrée.

9.4 Taille des petites BDAI

Nous appellerons *petite BDAI* une BDAI construite avec la méthode proposée ci-dessus. Des bornes peuvent être définies pour la taille des petites BDAI. Si $\mathcal{B}d^+(I)_{R_i \to R_j}$ est le sous-ensemble de $\mathcal{B}d^+(I)$ dont les éléments sont définis de schéma R_i vers le schéma R_j, et que $\mathcal{B}d^+(I)$ est non circulaire, on peut écrire :

Propriété 5 *Soit* $\overline{\boldsymbol{d}} = \{\overline{r}_1, ..., \overline{r}_n\}$ *une petite BDAI pour* $\boldsymbol{d} = \{r_1, ..., r_n\}$ *sur un schéma* $\boldsymbol{R} = \{R_1, ..., R_n\}$ *. On a :*

$|\overline{\boldsymbol{d}}| = \sum_{i=1}^{n} |\overline{r}_i|$ *avec,* $\forall i = 1, \ldots, n$ *:*

$\lceil \frac{1+\sqrt{1+8\times|GEN(F_{r_i})|}}{2} \rceil \leq |\overline{r}_i|$

et

$|\overline{r}_i| \leq Min(|r_i|, 2 \times |GEN(F_{r_i})| + \sum_{j=1}^{n} |\mathcal{B}d^-(I)_{R_i \rightarrow R_j}| + \sum_{j=1}^{n} |\mathcal{B}d^+(I)_{R_j \rightarrow R_i}| \times |\overline{r}_j|)$

Preuve *La justification de ces bornes provient directement de la méthode proposée pour la construction des petites BDAI.*

La borne inférieure pour chaque relation correspond à celle donnée pour les petites RAI ; elle est atteinte dans le cas où la prise en compte des DI n'ajoute aucun tuple supplémentaire.

La borne supérieure pour une relation $\overline{r}_i$ est obtenue lorsque les conditions suivantes sont réunies :

- *La petite RAI $\dot{r}_i$ de départ a atteint elle-même sa borne supérieure $(2 \times |GEN(F_{r_i})|$;*
- *La contradiction des DI non satisfaites à conduit à l'ajout d'un tuple disqualifiant pour chaque DI de $\mathcal{B}d^-(I)$ qui sort du schéma R_i de r_i ;*
- *Pour forcer chaque DI de $\mathcal{B}d^+(I)$ de la forme $R_j[X] \subseteq R_i[Y]$, il a fallu ajouter une valeur par tuple de $\overline{r}_j$.*

Naturellement, si la taille de la relation initiale est inférieure à cette borne, alors c'est elle qui devient la borne supérieure. □

On constate que la définition de la borne supérieure est récursive ; pour chaque relation, elle prend en compte la taille des relations qui la "précèdent" dans les DI de $\mathcal{B}d^+(I)$. Pour évaluer la valeur de cette borne il faut donc considérer les schémas de relations comme cela est fait dans l'algorithme 14, en commençant par les relations n'ayant pas de DI entrantes. Cette définition récursive interdit l'évaluation de la borne pour les ensembles de DI circulaires. Dans ce cas, comme on l'a vu, une borne supérieure est donnée par la taille de la relation initiale.

Les expérimentations montreront que les bornes données dans la propriété 5 sont assez larges. Une caractérisation plus fine de la taille des petites BDAI est un problème ouvert.

Chapitre 10

Expérimentations

Sommaire

Nous avons appliqué ces différents résultats aux relations de la base de données **movies** présentée dans l'introduction de ce mémoire (p. 8). Il s'agit non seulement de montrer la faisabilité en pratique de nos propositions, mais aussi d'en mesurer l'intérêt potentiel : nous allons nous intéresser en particulier au pouvoir de réduction des RAI dans le cas de **movies**.

10.1 Relations d'Armstrong informatives

10.1.1 Calcul des générateurs et des bornes

Nos algorithmes n'incluent pas le calcul des générateurs, qui est un problème connexe. Dans nos expérimentations ceux-ci sont déduits à partir du calcul des ensembles en accord ; pour la relation *casts*, ce calcul a demandé environ 1 heure. Une perspective pratique intéressante serait d'implanter différentes méthodes pour la découverte des DF telles que [HKPT99, NC01], dont les performances dépendent des caractéristiques des bases considérées. Des comparaisons peuvent être trouvées dans [NC01, LPL02].

Le tableau 10.1 donne alors le nombre de générateurs pour chaque relation, ainsi que les bornes qui en découlent pour la taille des petites RAI (corollaire 1 page 82). La dernière colonne indique le nombre de possibilités pour construire une petite RAI, calculé à partir du nombre de couples par générateur et du corollaire 2.

Relation	nb. de generateurs	borne min	borne sup	nb. de choix possibles
actors	82	16	164	$\simeq 2.8 \times 10^{27}$
casts	7	7	14	$\simeq 20 \times 10^{9}$
movies	44	13	88	$\simeq 5.6 \times 10^{10}$
people	44	11	88	$\simeq 6.7 \times 10^{15}$
remakes	9	7	18	$\simeq 71 \times 10^{6}$
studios	19	8	38	27 648

FIG. 10.1 – Nombre de générateurs et bornes sur les IAR pour la BD **movies**

On constate que ces résultats sont très corrects en ce qui concerne la taille des petites RAI : le nombre de générateurs étant dans chaque cas très inférieur au nombre de tuples dans la relation, la borne supérieure est elle-même très petite. Signalons en particulier le cas surprenant de la relation *cast* contenant 45442 tuples, pour laquelle on est assuré de construire une RAI avec un maximum de 14 tuples, soit une réduction par un facteur supérieur à 3000, sans altérer les dépendances fonctionnelles satisfaites.

Ces résultats renforcent l'intérêt de l'algorithme 10 page 83 qui construit au hasard une petite RAI : facile à mettre en œuvre et peu coûteux, il construit une RAI généralement très petite.

10.1.2 Utilisation de l'heuristique

Toutefois, il est intéressant aussi de remarquer l'écart conséquent entre les bornes inférieures et les bornes supérieures des petites RAI. Et en particulier le fait que les bornes inférieures, issues d'un calcul logarithmique, sont extrêmement petites dans tous les cas. La volonté de ne pas s'arrêter à une construction au hasard est donc renforcée par ces résultats.

Nous avons appliqué l'heuristique présentée dans la section 8.5 pour la construction de petites RAI. Les résultats sont reportés dans la dernière colonne du tableau 10.2. Pour améliorer leur lisibilité, nous avons reporté la taille des relations initiales, ainsi que les bornes pour les petites RAI.

Même si les tailles obtenues sont inférieures dans tous les cas à la borne supérieure, on reste malheureusement très loin de la borne inférieure. Mais l'utilisation d'une heuristique n'est pas la seule raison de cet écart ; les bornes inférieures sont en fait beaucoup trop optimistes. Elles correspondent en effet à des cas extrêmement favorables, où il est possible de trouver un ensemble de tuples *qui soient tous en accord deux à deux sur un générateur différent.* Si on se rapporte à la représentation du problème sous la forme d'un graphe coloré (page 85), ces cas correspondent à la présence dans le graphe d'une clique dans laquelle toutes les couleurs sont représentées exactement une fois. En présence d'une

Relation	taille	borne min	borne sup	RAI avec heuristique
actors	6728	16	164	145
casts	45442	7	14	13
movies	11405	13	88	84
people	3304	11	88	79
remakes	1188	7	18	17
studios	195	8	38	33

FIG. 10.2 – Taille des RAI construites avec l'heuristique

relation donnée, deux raisons peuvent expliquer que la taille des RAI minimales soit éloignée de la borne minimale :

1. Il n'existe pas d'ensemble de tuples dans la relation qui permette d'atteindre la borne ;
2. La *structure des générateurs* rend l'existence de tels ensembles impossible. A notre connaissance, il n'existe pas de borne inférieure pour la taille des relations d'Armstrong prenant en compte la structure des maximaux (nombre d'intersections entre maximaux, taille)

   Exemple 37 On peut illustrer ce deuxième point par un exemple soulevé dans [BDFS84], où les générateurs sont de la forme : $GEN(F) = \{A, AB, ABC, \ldots\}$. Dans ce cas, les auteurs démontrent qu'il est impossible de construire une relation d'Armstrong de taille inférieure à $|GEN(F)| + 1$.

10.1.3 Calcul d'une RAI minimale avec CPlex

Nous avons vu que nous ne disposions pas d'algorithme efficace pour calculer une RAI minimale. De plus il est impossible de parcourir toutes les possibilités pour déterminer la plus petite, comme l'illustre la dernière colonne du tableau 10.1. Pour avoir toutefois une idée de la taille de la RAI minimale, nous avons utilisé le programme linéaire modélisant le problème de la RAI minimale, donné page 88.

Une fois que l'ensemble des générateurs et des couples de tuples correspondants sont connus, il est aisé de générer un fichier d'entrée pour CPlex 7.1. Celui-ci utilise alors une méthode de *Branch and Bound* [Wol98] pour résoudre le problème.

Les résultats sont reportés dans la figure 10.3. La deuxième colonne donne le nombre de variables du problème, la troisième le nombre de contraintes. Nous avons reporté la taille de la RAI calculée avec l'heuristique dans la quatrième colonne, afin de faciliter la comparaison. La troisième colonne donne la taille de la solution optimale retournée par CPlex, et la quatrième le temps écoulé pour la résolution.

Relation	nb de variables	nb de contraintes	RAI avec heuristique	RAI minimale	temps (s.)
actors	987	473	145	142	3
casts	995	716	13	12	9015
movies	381	186	84	83	2
people	647	295	79	76	2
remakes	460	297	17	15	2
studios	94	61	33	33	1

FIG. 10.3 – Résultat apportés par la PLNE

La première constatation est qu'une solution est trouvée pour les six relations de **movies**, en seulement quelques secondes dans presque tous les cas. Toutefois, dans le cas de la relation *casts* qui induit un plus grand nombre de contraintes, la résolution a été beaucoup plus longue. On peut donc raisonnablement penser que dans le cas d'instances plus grandes, cette méthode n'aboutira pas. Ajoutons enfin que ces résultats ont été obtenus sans une étude plus approfondie du problème linéaire ; en particulier, certaines contraintes supplémentaires pourraient éventuellement être découvertes. L'idée serait de trouver des "coupes" efficaces de l'espace de recherche, pour atteindre plus rapidement des solutions entières par un algorithme dit de "branch and cut" [Wol98]. Précisons que l'obtention de telles coupes découle bien souvent d'une étude dédiée au problème, dans le domaine de l'optimisation combinatoire.

Mais en ce qui concerne la taille du résultat, force est de constater que celle-ci est relativement élevée : nous sommes définitivement très loin de la borne inférieure convoitée. Nous avons alors une idée de la qualité de l'heuristique : les résultats sont très proches de l'optimale, et une solution est fournie dans tous les cas en temps raisonnable.

10.2 BD d'Armstrong informatives

Nous avons ensuite construit une base de données informatives pour **movies**, à partir des RAI calculées précédemment par l'heuristique et des DI découvertes précédemment. $\mathcal{B}d^-(I_{\mathbf{movies}})$ est composée de 3317 DI et $\mathcal{B}d^+(I_{\mathbf{movies}})$ de 7 DI, dont 6 sont non circulaires, et une est interne à la relation *studios*. La figure 10.4 donne l'évaluation des bornes pour chaque relation d'une petite BDAI (colonne 5) *en ne tenant compte que des DI non circulaires*. Les deuxième et troisième colonnes détaillent, pour chaque relation, le nombre de DI sortantes dans $\mathcal{B}d^-(I_{\mathbf{movies}})$ et le nombre de DI entrantes dans $\mathcal{B}d^+(I_{\mathbf{movies}})$. Rappelons que ces deux nombres sont utilisés dans le calcul de la borne supérieure (proposition 5 page 101).

On constate alors qu'on est assuré de construire une BDAI de taille inférieure à 5784 pour **movies**, ce qui correspond à une réduction par un facteur de 11 en conservant les mêmes DF et DI (non circulaires).

Après exécution de l'algorithme nous avons donné, pour chaque étape, la cardinalité

Relation	DI sortantes dans $\mathcal{B}d^{-}(I_{\mathbf{movies}})$	DI entrantes dans $\mathcal{B}d^{+}(I_{\mathbf{movies}})$	Borne sup
actors	684	2	1512
casts	397	0	397
movies	684	2	1495
people	627	2	1455
remakes	414	0	414
studios	511	0	511
TOTAL	3317	6	5784

FIG. 10.4 – Bornes pour la taille des petites BDAI pour **movies**

de la BDAI en construction dans la figure 10.5.

	movies	$\dot{movies}$	$\widehat{movies}$	$\overline{movies}$
cardinalité	68262	374	374	434

FIG. 10.5 – Nombre de tuples dans la construction d'une BDAI pour **movies**

La cardinalité de $\dot{movies}$ est reportée dans la colonne 3 du tableau 10.5, et correspond à la somme des cardinalités de chaque IAR calculée précédemment. Toutes les DI de $\mathcal{B}d^{+}(I_{\mathbf{movies}})$ ont été considérées, y compris la DI circulaire interne à *studios* (celle-ci n'a conduit qu'à l'ajout de deux tuples).

On constate alors que la taille obtenue pour la BDAI de **movies** est très intéressante, et correspond à un facteur de réduction de 150. Le principal élément expliquant l'écart avec la bordure supérieure théorique est qu'un tuple disqualifiant n'est pas systématiquement rajouté pour chaque DI de $\mathcal{B}d^{-}(I)$: l'algorithme vérifie d'abord qu'un tel tuple n'a pas déjà été inséré lors de la construction des RAI. De fait, aucun tuple disqualifiant n'a dû être inséré : pour chaque DI de la bordure négative, cela était déjà fait.

Au demeurant il est difficile de prévoir le nombre exact de tuples à rajouter pour contredire ou satisfaire les DI. Toutefois, dans de nombreux cas, on peut imaginer que beaucoup de DI de la bordure négative sont déjà disqualifiées dès la construction des RAI, ce qui conduirait à rendre la borne positive souvent surestimée.

Précisons que la construction de la BDAI à partir des RAI ne prend qu'environ une minute, en tenant compte du calcul de la bordure positive et la bordure négative des DI de **movies** (17 secondes).

Cinquième partie

Conclusion et perspectives

Conclusion

Deux objectifs principaux peuvent justifier la volonté de découvrir les dépendances fonctionnelles (DF) et les dépendances d'inclusion (DI) satisfaites dans une base de données relationnelle existante. Le premier est la compréhension de la *sémantique du schéma de la base*, dont les applications sont nombreuses : normalisation des bases, intégration de données, définition de vues. Le deuxième est une meilleure connaissance de la structure des données dans une instance particulière, à un instant donné, point de départ de travaux sur l'optimisation de requêtes ou la définition automatique d'index.

Alors que la découverte des DF a reçu une grande attention, la découverte des DI est restée plutôt inexplorée. Nous avons présenté des algorithmes pour la découverte d'une couverture de toutes les DI satisfaites dans une base de données, ainsi que des DI approximativement satisfaites. Nous avons défini un espace de recherche partiellement ordonné, suffisant pour résoudre ce problème. La propriété de satisfaction des DI est anti-monotone dans cet espace, et permet l'élaboration de différents d'algorithmes de parcours dont l'objectif est toujours d'éviter de considérer un grand nombre de candidats. Nous avons proposé trois algorithmes, complémentaires, pour explorer cet espace de recherche.

- Le premier considère les DI unaires. Cette étape est déterminante, car les DI unaires satisfaites "définissent" en quelque sorte la suite du parcours, en constituant la base de l'espace de recherche restant. Les expérimentations montrent qu'une approche naïve est beaucoup trop coûteuse en pratique ; notre algorithme en revanche permet de traiter efficacement ce sous-problème .
- A partir des DI unaires satisfaites, l'algorithme *Mind* réalise un parcours rigoureux, par niveaux, de l'espace de recherche. Une DI n'est considérée que si toutes celles qui la précèdent dans l'espace ont été elles-mêmes testées comme satisfaites.
- Enfin, l'algorithme *Zigzag* effectue, après un début de parcours par niveaux, des sauts dans l'espace de recherche, justifiés par une propriété du modèle relationnel. Ces sauts sont techniquement rendus possibles par une représentation ensembliste des DI, ramenant le problème à la génération des transversaux minimaux d'un hypergraphe. Une mesure d'erreur sur les grandes DI non satisfaites permet d'estimer la distance qui les sépare des DI satisfaites, afin d'orienter la suite du parcours.

De fait, l'algorithme *Zigzag* intègre les deux premiers algorithmes pour parcourir l'espace de recherche. Il est paramétrable sur deux points : le niveau à partir duquel on commence à effectuer des sauts, et le seuil permettant d'évaluer la distance des grandes DI fausses aux DI satisfaites. L'intérêt de cet algorithme dépasse la simple découverte des DI, puisque il est applicable à d'autres tâches de fouille de données. Les conditions d'application sont un espace de recherche partiellement structuré, une propriété recherchée anti-monotone, et une représentation des candidats sous la forme d'ensembles. Ces conditions sont notamment réunies dans la découverte des motifs fréquents, et la découverte de la base canonique des dépendances fonctionnelles.

Après l'étape de découverte proprement dite, se pose le problème de la présentation des DF et DI découvertes à l'utilisateur. Elle doit lui permettre de comprendre les dépendances satisfaites et les dépendances non satisfaites, et l'aider à sélectionner celles qui l'intéressent en fonction du contexte applicatif. Comme moyen de visualisation, nous proposons de présenter à l'utilisateur de petits échantillons de la base, vérifiant exactement les mêmes DF et DI. Nous avons appelé ces échantillons des *bases de données d'Armstrong informatives* (BDAI). Elles sont une extension de la notion classique de bases de données d'Armstrong ; leur particularité est qu'elles sont peuplées de tuples parfaitement réels, ce qui leur confère un caractère informatif supplémentaire. On peut voir ces échantillons comme un ensemble d'illustrations de toutes les dépendances satisfaites et non satisfaites dans la base. Elles peuvent aussi servir à tester différents processus de restructuration des données (par exemple de normalisation), les changements définitifs étant appliqués plus tard à la base réel.

Le contexte applicatif dans lequel nous avons inscrit ces travaux est l'aide à l'administration et au réglage logique des bases de données relationnelles, au travers du projet "DBA Companion" [DLPT03]. Un prototype universitaire a été réalisé [LDP02, LDP04], dans lequel les propositions présentées ici sont intégrées, ou en cours d'intégration. L'utilisateur se connecte à une base de données via une interface graphique ; il peut alors rechercher les ensembles en accords, les clés, les DF, les DI, ou construire des bases de données d'Armstrong informatives. Une version de l'outil est disponible sur le web à l'adresse `http ://www.isima.fr/ demarchi/dbacomp`.

Perspectives

Quelques travaux peuvent-être envisagés pour compléter les résultats présentés dans ce mémoire. Pour la découverte des DI, une étude complète de la complexité de l'algorithme *Zigzag*, en fonction notamment de la taille de la couverture proposée en sortie. Nous prévoyons également une évaluation du comportement de l'algorithme *Zigzag* en fonction de ses deux paramètres d'entrée. On peut aussi envisager l'étude d'un ajustement automatique de ces paramètres ; en particulier, à partir de quel niveau adopter une attitude optimiste ? Une piste éventuelle serait une estimation de la probabilité qu'une grande DI soit satisfaite, en fonction du nombre de valeurs distinctes dans les parties droites et gauches et du nombre de tuples dans les relations.

Une autre perspective naturelle concerne la maintenance incrémentale de la couverture des DI, lors des différentes mises à jour de la base. Rappelons aussi la perspective d'adapter l'algorithme *Zigzag* à d'autres problèmes de fouille de données, en particulier la découverte des motifs fréquents dans des bases de transactions.

Nous avons également soulevé le problème de la construction d'une relation d'Armstrong informative de taille minimale, l'étude précise de sa complexité, et l'élaboration éventuelle d'une algorithme efficace pour le résoudre. Ajoutons d'ailleurs que le problème est également ouvert, à notre connaissance, en ce qui concerne les relations d'Armstrong de taille minimale, dans leur définition classique à partir d'une ensemble de DF. Ce dernier problème est notamment lié à la définition d'une borne minimale plus fine que celle existante, prenant en compte la nature des ensembles générateurs en entrée. Bien sûr les mêmes questions se posent pour les bases de données d'Armstrong informatives de taille minimale, que nous n'avons pas abordées.

Le caractère applicatif de ces relations exemples appelle également la réalisation de tests supplémentaires sur des bases de données réelles, la difficulté étant, on l'a dit, de se procurer de telles bases. Là encore se pose le problème de la maintenance incrémentale des bases de données d'Armstrong informatives, lorsque varie l'ensemble des DF ou l'ensemble des DI satisfaites dans la base originale.

Une perspective à plus long terme concerne l'exploitation des DF et DI découvertes. Comment leur connaissance pourrait conduire à l'optimisation de certaines tâches de fouille de données réalisées sur les données ? Par exemple en réduisant la quantité de données à fouiller, en évitant certains attributs, en déduisant des connaissances sans accéder aux données. Au delà du modèle relationnel, la définition des DF et DI peut s'adapter à toute représentation des données sous forme de tableaux attributs/valeurs ; on peut imaginer que, dans certains domaines (biologie, physique nucléaire), la connaissance apportée par les DF et DI possède un intérêt en soi pour les experts.

Bibliographie

[ABDT95] M. Albrecht, E. Buchholz, A. Düsterhöft, and B. Thalheim. An informal and efficient approach for obtaining semantic constraints using sample data and natural language processing. In L. Libkin and B. Thalheim, editors, *Semantics in Databases*, volume 1358 of *Lecture Notes in Computer Science*, pages 1–28. Springer, 1995.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, Reading, Mass., 1995.

[Arm74] W. W. Armstrong. Dependency structures of database relationships. In J. L. Rosenfeld, editor, *International Federation for Information Processing (IFIP'74), Stockholm, Sweden*, pages 580–583, 1974.

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *International Conference on Very Large Data Bases (VLDB'94), Santiago de Chile, Chile*, pages 487–499. Morgan Kaufmann, 1994.

[Bay98] R. J. Bayardo. Efficiently mining long patterns from databases. In L. M. Haas and A. Tiwary, editors, *ACM SIGMOD Conference, Seattle, USA*, pages 85–93, 1998.

[Bay99] S. D. Bay. The UCI KDD Archive [`http ://kdd.ics.uci.edu`]. Technical report, Irvine, CA : University of California, Department of Information and Computer Science, 1999.

[BB95] S. Bell and P. Brockhausen. Discovery of constraints and data dependencies in databases (extended abstract). In N. Lavrac and S. Wrobel, editors, *European Conference on Machine Learning (ECML'95), Crete, Greece*, volume 912 of *Lecture Notes in Computer Science*, pages 267–270. Springer, 1995.

[BDFS84] C. Beeri, M. Dowd, R. Fagin, and R. Statman. On the structure of Armstrong relations for functional dependencies. *Journal of the ACM*, 31(1) :30–56, 1984.

[Ber73] C. Berge. *Graphs and Hypergraphs.* North Holland, Amsterdam, 1973.

[BTP+00] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L. Lakhal. Mining frequent patterns with counting inference. *ACM SIGKDD Exploration*, 2(2) :66–75, 2000.

[BV84] C. Beeri and M. Vardi. A proof procedure for data dependencies. *Journal of the ACM*, 31(4) :718–741, 1984.

[CdS83] M. A. Casanova and J. E. Amaral de Sa. Designing entity-relationship schemes for conventional information systems. In C. G. Davis, S. Jajodia, P. A. Ng, and R. T. Yeh, editors, *International Conference on Entity-Relationship Approach (ER'83)*, pages 265–277. North Holland, 1983.

[CFP84] M. Casanova, R. Fagin, and C. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. *Journal of Computer and System Sciences*, 24(1) :29–59, 1984.

[CGK+99] Q. Cheng, J. Gryz, F. Koo, T. Y. Cliff Leung, L. Liu, X. Qian, and B. Schiefer. Implementation of two semantic query optimization techniques in DB2 universal database. In M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, editors, *International Conference on Very Large Data Bases (VLDB'99), Edinburgh, Scotland, UK*, pages 687–698. Morgan Kaufmann, 1999.

[Che76] P. P. Chen. The entity-relationship model - toward a unified view of data. *ACM Transaction On Database Systems*, 1(1) :9–36, 1976.

[CK85] S. Cosmadakis and P. Kanellakis. Equational theories and database constraints. In *ACM Symposium on Theory of Computing (STOC'85)*, pages 273–284, Providence, Rhode Island, 1985.

[CKS86] S. S. Cosmadakis, P. C. Kanellakis, and N. Spyratos. Partition semantics for relations. *Journal of Computer and System Sciences*, 93(2) :203–253, 1986.

[CN98] S. Chaudhuri and V. R. Narasayya. Autoadmin 'what-if' index analysis utility. In L. M. Haas and A. Tiwary, editors, *ACM SIGMOD Conference, Seattle, Washington, USA*, pages 367–378, 1998.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Communication of the ACM*, 13(6) :377–387, 1970.

[Cod74] E. F. Codd. Recent investigations in relational data base systems. In J. L. Rosenfeld, editor, *International Federation for Information Processing (IFIP'74), Stockholm, Sweden*, pages 1017–1021, 1974.

[CTF88] M. A. Casanova, L. Tucherman, and A. L. Furtado. Enforcing inclusion dependencies and referencial integrity. In F. Bancilhon and D. J. DeWitt, editors, *International Conference on Very Large Data Bases (VLDB'88), Los Angeles, California, USA*, pages 38–49. Morgan Kaufmann, 1988.

[CW01] T. Calders and J. Wijsen. On monotone data mining languages. In G. Ghelli and G. Grahne, editors, *International Workshop on Database Programming Languages (DBPL'01), Frascati, Italy*, LNCS 2397. Springer, 2001.

[CWA99] I. Comyn-Wattiau and J. Akoka. Relational datahase reverse engineering. *Networking and Information Systems*, 2(3) :345, 1999.

[DJMS02] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure ; or, how to build a data quality browser. In *ACM SIGMOD Conference 2002*, Madison, Wisconsin, USA, 2002.

[DLP01] F. De Marchi, S. Lopes, and J.-M. Petit. Informative armstrong relations : Application to database analysis. In *Bases de Données Avancées (BDA'01), Agadir, Maroc*, pages 211–217, 2001.

[DLP02a] F. De Marchi, S. Lopes, and J.-M. Petit. Efficient algorithms for mining inclusion dependencies. In C. S. Jensen, K. G. Jeffery, J. Pokorný, S. Saltenis, E. Bertino, K. Böhm, and M. Jarke, editors, *International Conference on Extending Database Technology (EDBT'02), Prague, Czech Republic*, volume 2287 of *Lecture Notes in Computer Science*, pages 464–476. Springer, 2002.

[DLP02b] F. De Marchi, S. Lopes, and J.-M. Petit. Samples for understanding data-semantics in relations. In M.-S. Hacid, Z. W. Ras, D. A. Zighed, and Y. Kodratoff, editors, *International Symposium on Methodologies for Intelligent Systems (ISMIS'O2), Lyon, France*, volume 2366 of *Lecture Notes in Artificial Intelligence*, pages 565–573. Springer, 2002.

[DLPT03] F. De Marchi, S. Lopes, J.-M. Petit, and F. Toumani. Analysis of existing databases at the logical level : the DBA companion project. *ACM Sigmod Record*, 32(1) :47–52, 2003.

[DP03a] F. De Marchi and J.-M. Petit. Construction de petites bases de données d'Armstrong informatives. *Revue d'intelligence artificielle RSTI-RIA*, 17, January 2003.

[DP03b] F. De Marchi and J-M. Petit. Zigzag : a new algorithm for discovering large inclusion dependencies in relational databases. In *International Conference on Data Mining (ICDM'03), Melbourne, Florida, USA*. IEEE Computer Society, 2003.

[DP03c] F. De Marchi and J.-M. Petit. Zigzag : un nouvel algorithme pour la découverte de grandes dépendances d'inclusion. In *Journées Bases de Données Avancées, Lyon, France*, page to appear, 2003.

[DRLP01] F. De Marchi, M. Rivon, S. Lopes, and J.-M. Petit. Mind : Algorithme par niveaux de découverte des dépendances d'inclusion. In *Inforsid 2001, Martigny, Swiss*, May 2001.

[DT95] J. Demetrovics and V.D. Thi. Some remarks on generating armstrong and inferring functional dependencies relation. *Acta Cybernetica*, 12(2) :167–180, 1995.

[EG95] T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6) :1278–1304, 1995.

[FK96] M. L. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3) :618–628, 1996.

[FPSSU96] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.

[FS99] P. A. Flach and I. Savnik. Database dependency discovery : A machine learning approach. *AI Communications*, 12(3) :139–160, 1999.

[FV83] R. Fagin and M.Y. Vardi. Armstrong databases for functional and inclusion dependencies. *Information Processing Letters*, 16 :13–19, 1983.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability.* W. H. Freeman and Company, New York, USA, 1979.

[GKMT97] D. Gunopulos, R. Khardon, H. Mannila, and H. Toivonen. Data mining, hypergraph transversals, and machine learning. In *Symposium on Principles of Database Systems (PODS'97), Tucson, Arizona*, pages 209–216. ACM Press, 1997.

[GMUW00] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database System Implementation.* Prentice-Hall, 2000.

[Gry98] J. Gryz. Query folding with inclusion dependencies. In *International Conference on Data Engineering (ICDE'98), Orlando, Florida, USA*, pages 126–133. IEEE Computer Society, 1998.

[GW99] B. Ganter and R. Wille. *Formal Concept Analysis.* Springer-verlag, 1999.

[GZ01] K. Gouda and M. J. Zaki. Efficiently mining of maximal frequent itemsets. In N. Cercone, T. Young Lin, and X. Wu, editors, *International Conference on Data Mining (ICDM'01), San Jose, USA*. IEEE Computer Society, 2001.

[HKPT99] Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. TANE : An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2) :100–111, 1999.

[KA96] A. J. Knobbe and P. W. Adriaans. Discovering foreign key relations in relational databases. In *European Meeting on Cybernetics and Systems Research*, pages 961–966, 1996.

[KCV83] P.C. Kanellakis, S.S. Cosmadakis, and M.Y. Vardi. Unary inclusion dependencies have polynomial time inference problems. In *Symposium on Theory of Computing (STOC'83)*, pages 264–277, Boston, Massachussets, 1983.

[KM95] J. Kivinen and H. Mannila. Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, 149(1) :129–149, 1995.

[KMRS92] M. Kantola, H. Mannila, K. J. Räihä, and H. Siirtola. Discovering functional and inclusion dependencies in relational databases. *International Journal of Intelligent Systems*, 7 :591–607, 1992.

[KR03] A. Koeller and E. A. Rundensteiner. Discovery of high-dimentional inclusion dependencies (poster). In *Poster session of International Conference on Data Engineering (ICDE'03)*. IEEE Computer Society, 2003.

[LDP02] S. Lopes, F. De Marchi, and J.-M. Petit. DBA companion : un outil pour l'analyse de bases de données. In P. Pucheral, editor, *Journées Bases de Données Avancées (BDA'02), session démonstrations, Evry, france*, pages 523–527, `http ://www.isima.fr/~demarchi/dbacomp/`, 2002.

[LDP04] S. Lopes, F. De Marchi, and J.-M. Petit. DBA Companion : A tool for logical database tuning. In *Demo session of International Conference on Data Engineering (ICDE'04)*. IEEE Computer Society, 2004.

[LK98] D.-I Lin and Z. M. Kedem. Pincer search : A new algorithm for discovering the maximum frequent set. In H.-J. Schek, F. Saltor, I. Ramos, and G. Alonso, editors, *Extending Database Technology (EDBT'98), Valencia, Spain*, volume 1377 of *Lecture Notes in Computer Science*, pages 105–119. Springer, 1998.

[LL99] M. Levene and G. Loizou. *A Guided Tour of Relational Databases and Beyond*. Springer, 1999.

[LLSV99] D. Laurent, J. Lechtenbörger, N. Spyratos, and G. Vossen. Complements for data warehouses. In *International Conference on Data Engineering (ICDE'99), 23-26 March 1999, Sydney, Austrialia*, pages 490–499. IEEE Computer Society, 1999.

[LPL00] S. Lopes, J.-M. Petit, and L. Lakhal. Efficient discovery of functional dependencies and armstrong relations. In C. Zaniolo, P. C. Lockemann, M. H. Scholl, and T. Grust, editors, *International Conference on Extending Database Technology (EDBT'00), Konstanz, Germany*, volume 1777 of *Lecture Notes in Computer Science*, pages 350–364. Springer, 2000.

[LPL01] S. Lopes, J.-M. Petit, and L. Lakhal. A framework for understanding existing databases. In M. E. Adiba, C. Collet, and B. C. Desai, editors, *International Database Engineering and Applications Symposium (IDEAS'01), Grenoble, France*, pages 330–338, July 2001.

[LPL02] S. Lopes, J.-M. Petit, and L. Lakhal. Functional and approximate dependencies mining : Databases and FCA point of view. *Special issue of JETAI*, 14(2/3) :93–114, 2002.

[LPT99] S. Lopes, J.-M. Petit, and F. Toumani. (short paper) Discoverf of "Interesting" Data Dependencies from a Workload of SQL Statements. In J. M. Zytkow and J. Rauch, editors, *Principles of Data Mining and Knowledge Discovery (PKDD'99)*, volume 1704 of *Lecture Notes in Computer Science*, pages 430–435, Prague, Czech Republic, 1999.

[LPT02] S. Lopes, J.-M. Petit, and F. Toumani. Discovering interesting inclusion dependencies : Application to logical database tuning. *Information System*, 17(1) :1–19, 2002.

[LV00] M. Levene and M. W. Vincent. Justification for inclusion dependency normal form. *IEEE Transactions on Knowledge and Data Engineering*, 12(2) :281–291, 2000.

[MHH+01] R. J. Miller, M. A. Hernández, L. M. Haas, L. Yan, C. T. H. Ho, R. Fagin, and L. Popa. The clio project : Managing heterogeneity. *ACM SIGMOD Record*, 30(1) :78–83, 2001.

[Mit83] J. C. Mitchell. The implication problem for functional and inclusion dependencies. *Information and Control*, 56(3) :154–173, 1983.

[MM90] V.M. Markowitz and J.A. Makowsky. Identifying Extended Entity-Relationship Object Structures in Relational Schemas. *Transactions on Software Engineering*, 16(1) :777–790, 1990.

[MR86a] H. Mannila and K.-J. Räihä. Inclusion dependencies in database design. In *International Conference on Data Engineering (ICDE'86), Los Angeles, California, USA*, pages 713–718. IEEE Computer Society, 1986.

[MR86b] H. Mannila and K.-J. Räihä. Design by example : An application of armstrong relations. *Journal of Computer and System Sciences*, 63(2) :126–141, 1986.

[MR94a] H. Mannila and K. J. Räihä. Algorithms for inferring functional-dependencies from relations. *Data and Knowledge Engineering*, 12(1) :83–99, 1994.

[MR94b] H. Mannila and K. J. Räihä. *The Design of Relational Databases.* Addison-Wesley, second edition, 1994.

[MT97] H. Mannila and H. Toivonen. Levelwise Search and Borders of Theories in Knowledge Discovery. *Data Mining and Knowledge Discovery*, 1(1) :241–258, 1997.

[NC01] N. Novelli and R. Cicchetti. Functional and embedded dependency inference : a data mining point of view. *Information System*, 26(7) :477–506, 2001.

[PBTL99] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1) :25–46, 1999.

[PTBK96] J.-M. Petit, F. Toumani, J.-F. Boulicaut, and J. Kouloumdjian. Towards the reverse engineering of denormalized relational databases. In S. Y. W. Su, editor, *International Conference on Data Engineering (ICDE'96), New Orleans, Louisiana*, pages 218–227. IEEE Computer Society, 1996.

[QGMW96] D. Quass, A. Gupta, I. S. Mumick, and J. Widom. Making views self-maintainable for data warehousing. In *International Conference on Parallel and Distributed Information Systems (PDIS'96), Miami Beach, Florida, USA*, pages 158–169. IEEE Computer Society, 1996.

[SM79] A. M. Silva and M. A. Melkanoff. A method for helping discover the dependencies of a relation. In H. Gallaire, J.-M. Nicolas, and J. Minker, editors, *Advances in Data Base Theory (ADBT'81)*, pages 115–133, Toulouse, France, 1979.

[STA00] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems : Alternatives and implications. *Data Mining and Knowledge Discovery*, 4(2/3) :89–125, 2000.

[WGR01] C. Wyss, C. Giannella, and E. Robertson. Fastfds : A heuristic-driven depth-first algorithm for mining functional dependencies from relation instances. In Y. Kambayashi, W. Winiwarter, and M. Arikawa, editors, *Data Warehousing and Knowledge Discovery (DaWaK'01), Munich, Germany*, volume 2114 of *Lecture Notes in Computer Science*, pages 101–110, 2001.

[Wol98] L. Wolsey. *Integer programming.* Wiley, 1998.

[Zan84] C. Zaniolo. Database relations with null values. *Journal of Computer and System Sciences*, 28(1) :142–166, 1984.