UNIVERSITÉ DU QUÉBEC À MONTRÉAL

BASES DE RÈGLES MULTI-NIVEAUX

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

CHRISTIAN PAGÉ

FÉVRIER 2008

REMERCIEMENTS

Remercier est d'autant plus facile que l'on est conscient de ne pas être arrivé là tout seul : Derrière le succès de tout étudiant se presse une foule de gens qui ont contribué à sa réussite.

Je tiens tout d'abord à remercier M. Robert Godin, professeur titulaire au département d'informatique de l'Université du Québec à Montréal et mon directeur de recherche : c'est un informaticien, mathématicien et pédagogue hors pair qui a su quel niveau de proximité dans l'encadrement convenait à chacun de ses étudiants du laboratoire.

Qui dit « Laboratoire » dit « professeurs ». Je voudrais aussi remercier les professeurs du LATECE dont, particulièrement, M. Hafedh Mili dont les visites étaient toujours stimulantes et M. Petko Vatchev qui a maintes fois éclairé ma lanterne en analyse formelle de concepts.

Je m'en voudrais énormément d'oublier M. Roger Villermaire, directeur du programme de maîtrise en informatique de l'Université du Québec à Montréal, qui a accepté de donner une chance à un étudiant qui n'avait pas le « profil traditionnel ».

Qui dit « Laboratoire » dit également « étudiants ». Merci à mes camarades du LATECE, étudiants ou stagiaires, toujours prêts à échanger autour d'un café conseils, trucs et astuces diverses : Céline, Alain, Cyril, Jean-François, Jeanne, Aniss, Ghizlane, Kevin, Radouane, Romdhane, Nizar, Mohammed, Abdeltif, Guitta et Anne-Élizabeth.

Qui dit « étudiant » dit aussi « famille ». Réaliser ces travaux et rédiger ce mémoire n'auraient jamais été possible sans l'appui, les encouragements et le soutien indéfectibles de mon épouse Nicole et la patience et compréhension de mes enfants, Sébastien et Alexandre, sans oublier le regard et l'affection de mes parents.

Remercier est finalement d'autant plus difficile que l'on risque de commettre des oublis. Je tiens donc à m'excuser auprès de tous ceux et celles dont j'ai bien involontairement tu le nom. Soyez assurés que votre contribution n'en fut pas moins grande ni appréciée.

Merci donc à vous toutes et à vous tous, et de tout cœur.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

RÉSUMÉ

La fouille de données est définie comme le traitement d'une grande quantité de données afin d'y extraire des connaissances non triviales et utiles. Cette analyse permet de dégager de la masse d'informations des tendances, des regroupements de données et de formuler des hypothèses. Un des domaines de la fouille de données est la recherche de règles d'association. Les algorithmes utilisés en recherche de règles d'association ont généralement l'inconvénient de ne pouvoir identifier des règles dont un des termes est infréquent, mais qui appartient à une catégorie qui, elle, l'est. Les règles d'association multi-niveaux permettent d'identifier les associations impliquant des termes dont les niveaux de généralisation/spécialisation diffèrent.

Les algorithmes de recherche de règles d'association multi-niveaux présentés à ce jour ont en commun la génération d'un nombre souvent très grand de règles redondantes.

Notre contribution dans cette étude est constituée de la conception de deux algorithmes de recherche de règles d'association mutli-niveaux basés sur l'analyse formelle de concepts, ce qui permet de restreindre la génération des règles d'association aux seules règles informatives maximales. Nous avons également réalisé l'implémentation de ces deux algorithmes, en plus de celle d'un autre algorithme utilisé aux fins de comparaison dans la littérature. Nous avons finalement comparé expérimentalement ces trois implémentations et les résultats obtenus confirment l'intérêt de l'approche basée sur l'analyse formelle de concepts, tout en illustrant l'effet des optimisations apportés au traitement.

Mots clés : treillis de Galois (treillis de concepts), analyse formelle de concepts, fouille de données (data mining), règles d'association, base de règles, règles d'association multi-niveaux (règles d'association généralisées), base de règles multi-niveaux (bases de règles généralisées)

CHAPITRE I

INTRODUCTION

Cette section consiste en une introduction à la fouille de données et au problème particulier qui nous intéresse, la fouille de règles d'associations multi-niveaux.. Les notions générales de fouille de données, de taxonomie et d'analyse formelle de concepts sont introduites ici et un survol des principaux travaux recensés dans la littérature scientifique complète la section.

Sauf mention explicite, les définitions de ce chapitre sont tirées du livre de Robert Godin (Godin, 2006).

1.1 Fouille de données

La fouille de données, aussi appelée prospection de données, est définie comme le traitement d'une grande quantité d'informations afin d'y extraire des connaissances non triviales et utiles (Han et Kamber, 2001). Cette analyse permet de dégager de la masse d'informations des tendances, des regroupements de données et de formuler des hypothèses. La fouille de données comprend principalement les catégories suivantes (Godin, 2006) :

- La classification qui vise à identifier, au sein d'une population, à quelle catégorie appartient un individu. Les classes sont connues au préalable et un jeu de données dit « d'entraînement » est fourni. Les règles d'association du type *individu→classe* sont découvertes et servent à prédire à quelle(s) classe(s) appartiendront de nouveaux individus. Un autre type de classification est la classification par rétropropagation (Han et Kamber, 2001). Cet algorithme utilise un réseau de neurones (ensemble d'unités réparties en couches et capables d'opérations d'entrée/sortie et reliées entre elles par des liens pondérés). Un jeu d'essai est soumis au réseau et les résultats du traitement sont acheminés à la couche d'entrée

pour permettre d'affiner la pondération des liens (c'est la rétropropagation). Le traitement se poursuit jusqu'à ce que la classification obtenue pour le jeu d'essai corresponde aux résultats escomptés (fin de la phase d'apprentissage), puis les données expérimentales sont traitées à leur tour. Les logiciels d'aide au diagnostic médical constituent une application de la classification.

- La prévision numérique qui vise à découvrir des relations numériques existant entre des individus (par exemple, *degres_celsius*=(9/5 x *degres_farenheit*) + *32*). Pour ce faire, la prévision numérique utilise des analyses de régression numérique et d'autres méthodes, telles les réseaux de neurones et les machines à vecteur support et noyau (ces dernières étant des algorithmes d'apprentissage utilisées en classification et en regroupement dans des problèmes de classification non linéaire) (Cortes et Vapnik, 1995) . Une fois les relations caractérisées, les résultats peuvent ensuite être appliqués à de nouvelles données.

- Le regroupement (*cluster analysis*) vise à découvrir et caractériser des regroupements naturels d'individus. Contrairement à la classification, le nombre et la nature des classes n'est pas connu au préalable. La désambiguation de termes techniques dans la littérature biomédicale (Berardi *et al.*, 2005) et l'appariement d'ontologies dans le cadre du web sémantique (Jiang et Tan, 2006) sont des exemples d'application du regroupement.

- La découverte de patrons fréquents vise à découvrir des regroupements dans une population. Contrairement aux autres catégories de fouille de données vues plus haut, la découverte de patrons fréquents ne tente pas de caractériser ces regroupements. Une fois découverts, les patrons fréquents sont souvent utilisés par les autres types de fouille de données, en particulier dans la découverte de règles d'association.. Les *itemsets* (ensembles d'attributs ou *items*) sont un exemple de patrons.

- La découverte de patrons infréquents (*outlier analysis*) vise à découvrir des regroupements qui diffèrent significativement des autres regroupements présents dans une population, tout en restant peu nombreux. Ce type d'analyse utilise des

méthodes statistiques et est entre autres utilisé dans la détection de fraudes : par exemple, l'utilisation par un individu d'une carte de crédit pour acheter un bien ou service différent significativement de ses transactions habituelles peut indiquer que cette carte de crédit a été volée.

- La découverte de règles d'association qui, typiquement, recherche des liens entre des éléments de paniers d'achat de consommateurs. Par exemple, la règle « *imprimante* »⟶ « *écran* » indique que les consommateurs qui achètent une imprimante achètent également un écran. Parmi les applications des règles d'association, on retrouve également la correction d'erreurs dans les flux de données, la prédiction de pannes et, en gestion de réseaux, la détection d'incidents par l'analyse de journaux d'accès (logs) (Jiang et Gruenwald, 2006).
- L'analyse de l'évolution des données qui recherche des liens chronologiques afin de discerner des phénomènes périodiques ou des tendances séquentielles. Un exemple de cette analyse est la prédiction des tendances boursières, ou encore l'étude des changements climatiques

.

Un exemple d'application du fouille de données est l'analyse de paniers d'achats (market basket analysis), où l'on examine l'ensemble des produits contenus dans le panier d'achats de consommateurs lors de leur passage à la caisse, c'est-à-dire lorsqu'ils complètent leur transaction.

Le contenu d'un panier d'achats est défini comme une *transaction*. Chaque transaction est donc composée d'un identifiant qui lui est propre –généralement un numéro- et d'au moins un item. Un ensemble de transactions est appelé *base de transactions* (Han et Kamber, 2001). La figure suivante décrit un exemple de base de transactions.

transaction #1={imprimante, logiciel bureautique}
transaction #2={imprimante laser, écran}
transaction #3={imprimante, imprimante laser, souris, écran}
transaction #4={traitement de texte}
transaction #5={souris, logiciel}
transaction #6={souris}
transaction #7={routeur, écran}
transaction #8={souris, logiciel, logiciel bureautique}
transaction #9={imprimante}

Figure 1.1 : Exemple de base de transactions

L'examen du contenu du panier permet d'identifier quels items sont généralement achetés ensemble (donc associés) par les consommateurs (exemple : l'association *imprimante laser→écran*). Les propriétaires du commerce peuvent ensuite exploiter ces informations (dans notre cas : les items qui sont fréquemment associés au sein d'un même panier d'achats). Ainsi, des items fréquemment associés peuvent être disposés sur des étalages voisins afin d'en favoriser l'achat simultanné ; *a contriori*, ces items peuvent être délibérément éloignés pour favoriser l'achat d'autres produits disposés entre ces deux items fréquemment associés, etc... (Han et Kamber, 2001)

1.2 Bases pour les règles d'association

Un itemset est un ensemble d'attributs. Le support d'un itemset est égal au pourcentage de transactions qui contiennent cet itemset.

Une règle d'association $I_1 \Rightarrow I_2$ signifie que la présence de I_1 dans l'itemset d'une transaction implique la présence de I_2 dans l'itemset. La partie gauche I_1 est apelée l'antécédent et la partie droite I_2, le conséquent de la règle.

Une base de règles d'association d'une base de transactions est constituée des ensembles générateurs de toutes ses règles d'association, ainsi que de leurs supports et confiances (Pasquier, 2000) .

Le support d'une règle d'association $I_1 \Rightarrow I_2$ est défini comme étant le pourcentage des transactions qui contiennent tous les items des itemsets de la partie gauche et droite de la règle.

La confiance d'une règle d'association $I_1 \Rightarrow I_2$ est définie comme étant la proportion des transactions qui contiennent le conséquent I_2 parmi celles qui contiennent l'antécédent I_1. En probabilités, cette notion correspond à celle de probabilité conditionnelle (sachant que A est vrai, quelle est la probabilité que B le soit aussi).

Puisque chaque règle d'association est caractérisée par son support et sa confiance, il est possible de restreindre une base de règles d'association aux seules règles ayant un support supérieur ou égal à un seuil de support minimal. De même, les règles ayant une confiance de 100% sont dites « règles exactes », alors que les autres sont dites « règles approximatives ». Les bases de règles d'association ne contenant que des règles exactes sont dites « bases de règles exactes », alors que les autres sont dites « bases de règles approximatives ». Luxemberger (1991) a élaboré une définition formelle des bases de règles approximatives (Luxenberger, 1991). Comme notre étude porte sur les bases de règles d'association exactes, c'est sur celles-ci que nous élaborerons plus en détail.

Dans les algorithmes de la famille d'Apriori, la première étape dans la construction d'une base de règles d'association exactes de support supérieur ou égal à un seuil de support minimal est le recensement de l'ensemble des itemsets fréquents d'une base de transactions. La construction d'une base de règles d'association à partir de ces itemsets fréquents est, un problème NP-difficile (Godin et Missaoui, 1994), en plus de générer un grand nombre de règles peu informatives parce que redondantes ou incomplètes.

Par exemple, si un itemset fréquent était {*souris, écran, logiciel*}, nous pourrions en extraire entre autres les règles suivantes : {*souris*}⇒{*écran, logiciel*} et {*souris, écran*}⇒{*logiciel*}. Clairement, la deuxième règle est une conséquence logique de la première et, de plus, elle semble indiquer que *écran* est nécessaire pour que *logiciel* soit

présent, alors que la première règle prouve le contraire. Certaines règles obtenues à partir d'un itemset fréquent sont donc intéressantes, alors que d'autres ne le sont pas.

Dans une perspective d'application pratique des règles d'association, une règle sera d'autant plus intéressante que ses conclusions seront quantitativement grandes pour une prémisse quantitativement petite. Cela équivaut à privilégier les règles où la taille de l'antécédent est minimale et celle du conséquent est maximale, ce qui correspond à l'ensemble des règles informatives maximales. Ces règles, de la forme $r_1 : I_1 \Rightarrow I_2\text{-}I_1$, sont telles qu'il n'existe pas de règle $r_2 : J_1 \Rightarrow J_2\text{-}J_1$ ayant même support et même confiance que r_1 et telle que $r_1 \neq r_2$ et que $J_1 \subseteq I_1$ et $I_2 \subseteq J_2$ (Godin, 2006).

Par exemple, si nous considérons la règle {*souris*}⇒{*écran, logiciel*}, celle-ci ne sera pas une règle informative maximale si la règle {*souris*}⇒{*écran, logiciel, imprimante*} existe (pour autant que les deux règles aient même support et même confiance), puisque {*souris*}⊆{*souris*} et que {*écran, logiciel*}⊆{*écran, logiciel, imprimante*}.

Le défi devient donc de générer efficacement des bases ne contenant que des règles informatives maximales.

1.3 Itemset fermé et treillis de concepts

Considérons l'ensemble O des transactions, l'ensemble A des items et les fonctions $\varphi: O \rightarrow A$ (qui décrit l'ensemble des attributs associés à un objet particulier) et $\psi : A \rightarrow O$ (qui décrit l'ensemble des objets associés à un attribut particulier). Dans notre exemple (Figure 1), nous aurions O={*1,2,3,4,5,6,7,8,9*} et A={*routeur, imprimante, imprimante laser, souris, logiciel, logiciel bureautique, écran*}. De même, $\varphi(\{1\})$={*imprimante, logiciel bureautique*} et ψ({*souris*})={*3,5,6,8*}.

Un itemset I est dit *fermé* si il est formé de tous les items communs aux transactions auxquelles il correspond, c'est à dire si $I=\varphi(\psi(I))$. Par exemple, l'itemset{*imprimante, logiciel bureautique*} est fermé puisqu'il correspond à l'ensemble des items communs aux

transactions qui le contiennent (dans ce cas-ci, {*1*}), alors que l'itemset {*imprimante*} ne l'est pas, puisque $\varphi(\{1\})=\{$*imprimante, logiciel bureautique*$\}$.

L'intérêt des itemsets fermés pour la recherche de règles informatives maximales est que les règles d'association que l'on peut en extraire sont des règles informatives maximales (pour autant que l'antécédent de ces règles soit minimal), puisque la taille d'un itemset fermé est évidemment toujours maximale par rapport à l'ensemble des transactions qui le contiennent.

Tous les itemsets fréquents peuvent être déduits à partir de l'ensemble des itemsets frermés fréquents. En effet, si l'on considère un itemset fermé fréquent {*a,b,c,d*} auxquel est associé un ensemble d'objets {*1,2,3,4*} (on suppose qu'aucun autre attribut n'est associé à cet ensemble d'objets), alors tous les sous-ensembles de {*a,b,c,d*} sont également associés à {*1,2,3,4*} et ont donc même fréquence. Comme cette propriété est évidemment vraie pour tous les itemsets fermés fréquents, ceux-ci constituent donc une représentation concise de l'ensemble des itemsets fréquents. Pour la même raison, un itemset fréquent a le même support que sa fermeture, soit la cardinalité de l'ensemble des transactions qui y est associé par la fonction ψ.

Ces propriétés sont souvent utilisées dans la génération de bases (Ben Yahia *et al.*, 2006; Pasquier, 2000;Taouil *et al.*, 2000).

L'ensemble des itemsets fermés constitue une structure appelée treillis de Galois. Cette structure tire son nom de la correspondance de Galois qui existe entre les fonctions ψ et φ (Wille, 1982, Carpineto et Romano, 2004). L'analyse formelle de concepts –une branche de l'étude des treillis appliqués- repose sur l'utilisation des treillis de Galois.

Les éléments d'un treillis de Galois sont des concepts fermés, composés d'un ensemble d'objets *o* et d'un ensemble d'attributs *a* tels que $\psi(\varphi(o))=a$ et $\varphi(\psi(a))=o$. Autrement dit, *a* est fermé par rapport à *o* et *o* est fermé par rapport à *a*.

Dans notre exemple, ({*5,8*},{*souris, logiciel*}) serait un concept fermé, puisque {*5,8*} est fermé par rapport à {*souris, logiciel*} et que {*souris, logiciel*} est fermé par rapport à {*5,8*}. Inversement, ({*5*},{*souris, logiciel*}) et ({*5,8*},{*souris*}) ne sont pas des concepts fermés, pour les mêmes raisons.

1.4 Base générique pour règles d'association exactes

La notion de base de règles d'association exactes, telle que vue plus haut, ne tient pas compte du support, ce qui entraîne certaines difficultés car la redondance d'une règle peut changer selon qu'elle est considérée au simple niveau logique ou avec son support. Par exemple, la règle $\{souris, logiciel\} \Rightarrow \{écran\}$ semble redondante par rapport à la règle $\{logiciel\} \Rightarrow \{écran\}$, alors que le support de $\{souris, logiciel\} \Rightarrow \{écran\}$ pourrait être de 2/10 et que celui de $\{logiciel\} \Rightarrow \{écran\}$ pourrait être de 4/10. On ne pourrait donc pas déduire le support de la deuxième règle à partir de la première.

Pour éviter ce type de situation, Pasquier (2000) a introduit la notion de base générique pour règles d'association exactes. Cette base est constituée de l'ensemble des règles dont le support est supérieur à un seuil minimal et qui sont de la forme $IG \Rightarrow IG'' - IG$ où IG est un itemset générateur qui n'est pas fermé et où $IG \neq IG''$(Pasquier, 2000)

Un générateur d'un itemset fermé I est un sous-ensemble de I tel que sa fermeture soit l'itemset fermé I lui-même. (Pasquier, 2000)

1.5 Générateurs minimaux et recherche de règles d'association

Pour définir la notion d'itemset fermé et de générateur, considérons la relation d'équivalence (notée []) produite par l'ensemble des itemsets I qui correspondent au même ensemble de transactions T par l'opérateur ' (qui associe une transaction à ses objets) :

$$[T]=\{I \mid I' = T\}$$

Il en découle que tous les éléments de $[T]$ possèdent la même fermeture.

Par rapport à $[T]$, le fermé est l'élément maximal et un générateur est un élément minimal.Alors qu'il ne peut y avoir qu'un seul fermé dans $[T]$, on peut retrouver plusieurs

éléments minimaux, lesquels correspondent dans la classe d'équivalence aux générateurs (Godin, 2006)

Dans le cadre de la recherche de règles d'associations, les générateurs minimaux sont le complément des itemsets fermés fréquents. En effet, puisque leur intension est minimale par rapport à leur fermeture, les règles d'association dont ils sont les antécédents auront un conséquent maximal et sont donc, du point de vue de l'utilisateur, les plus intéressantes, puisqu'ils sont les prémisses minimales des règles informatives maximales (Pasquier, 2000).

1.6 Taxonomie

L'intérêt des taxonomies dans le cadre de la fouille de règles d'association réside dans le fait que, pour qu'une association entre un groupe d'items soit détectable, tous ces items doivent être retrouvés dans un ensemble de transactions en nombre suffisant. Or, si l'association *imprimante→papier* est relativement facile à détecter dans les transactions d'un détaillant en électronique, elle le devient beaucoup moins dès que le nombre de types d'imprimantes offerts augmente par rapport au nombre de variétés de papiers, ce qui a pour effet, tout en conservant le support de l'ensemble des papiers, de diluer le support individuel des produits de cette catégorie.

Une solution à ce problème consiste à permettre les associations entre produits et catégories de produits, ou même entre catégories de produits regroupés au sein de *taxonomies*. Un exemple de taxonomie est illustré par la Figure 2.

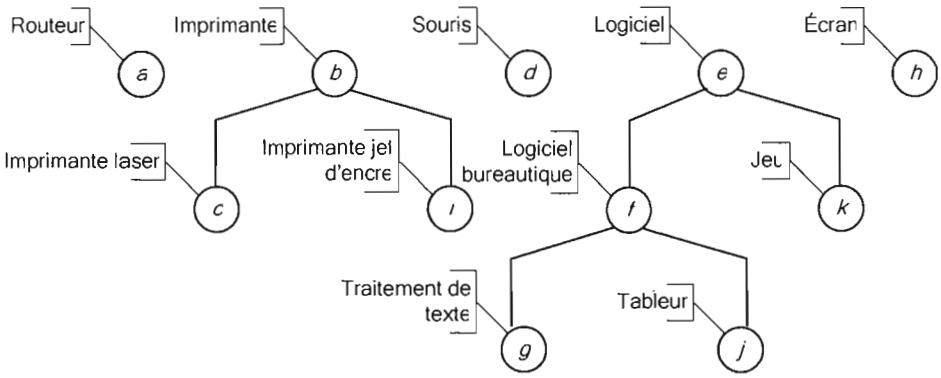

Figure 1.2 : Exemple de taxonomie. Les lettres contenues dans les pastilles indiquent l'abréviation assignée au nom de l'item (par exemple, «*a*» signifie «Routeur»).

Dans le cadre ce cette étude, une taxonomie regroupe les items au sein d'une hiérarchie pouvant être caractérisée par une relation d'ordre partiel allant du général (ex : « logiciel») au particulier (ex : « traitement de texte») . Si $a < b$, alors a est l'ancêtre de b (on aurait donc : « logiciel » < « traitement de texte »). Deux items n'ayant aucun ancêtre commun sont dits *incomparables*. Un attribut n'ayant aucun ancêtre est appelé *racine*.

1.7 Règles d'association multi-niveaux

Dans une règle d'association multi-niveaux, une relation taxonomique est définie sur l'ensemble des items formant les itemsets. Ainsi, les termes d'une telle règle peuvent être de niveaux de généralisation/spécialisation différents (par exemple, traitement de texte → imprimante). Si tous les items d'un ensemble de transactions sont des racines, il n'existera évidemment alors aucune différence entre les règles d'association « classiques » et les règles d'association multi-niveaux pouvant être déduites de ces transactions.

1.8 État de l'art

Les principaux écueils de la recherche de patrons fréquents sont le nombre exponentiel d'itemsets « candidats » (pouvant ou non être fréquents et dont la fréquence est à déterminer) et les accès à la base de transactions située en mémoire secondaire. Un algorithme de recherche de patrons fréquents ne pourra donc être efficace que dans la mesure où il évitera au maximum la génération (et donc le test) d'itemsets candidats et les parcours de la base de transactions. Les stratégies employées sont donc de deux types : l'élimination préalable des itemsets candidats dont on peut prédire mathématiquement qu'ils ne seront pas fréquents et la compression de la base de transactions par l'utilisation de structures de données appropriées afin de la faire résider en mémoire primaire.

Les algorithmes de recherche de patrons fréquents diffèrent généralement entre eux principalement dans la façon dont ils explorent l'espace des itemsets : en *largeur* , ou en *profondeur*,. L'ordre (selon le sens du parcours, de gauche à droite ou de haut en bas) dans lequel les itemsets sont considérés, ainsi que la façon dont les itemsets dont on peut prédire le non-intérêt sont évités sont d'autres critères utilisés. Les approches en largeur et en profondeur correspondent respectivement à un examen de la base de transactions objet par objet et attribut par attribut. Puisqu'il s'agit de la façon la plus générale de décrire l'approche des algorithmes étudiés, nous l'avons utilisé aux fins de première classification. Les autres critères mentionnés sont utilisés dans la description individuelle des algorithmes.

Dans un but d'illustration, supposons que l'ensemble $\{a,b,c,d\}$ contienne la totalité des items présents dans l'ensemble de la base de transactions. Les relations d'inclusion entre les sous-ensembles de $\{a,b,c,d\}$ (donc de l'ensemble des itemsets) peuvent être représentées par la figure suivante.

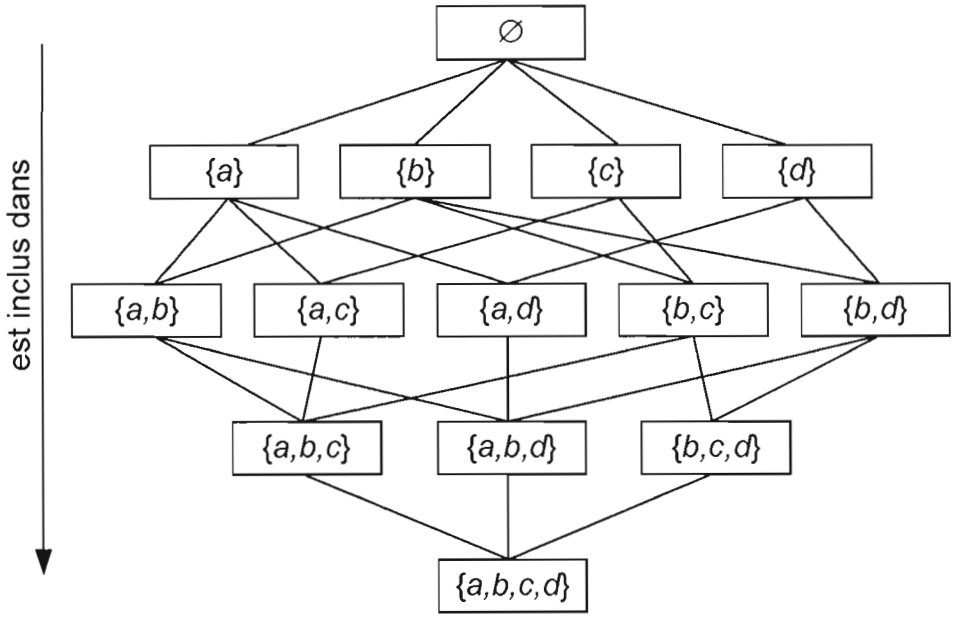

Figure 1.3: Représentation des sous-ensembles de l'ensemble {a,b,c,d}

Un algorithme de parcours en profondeur considèrera d'abord les sous-ensembles ayant mêmes préfixes (par exemple { *a* }, {*a,b*}, {*a,b,c*}), alors qu'un algorithme de parcours en largeur (aussi appelé « horizontal » ou « par niveaux ») considèrera d'abord les sous-ensembles de même taille,. Ces deux ordres de parcours sont illustrés par la figure suivante.

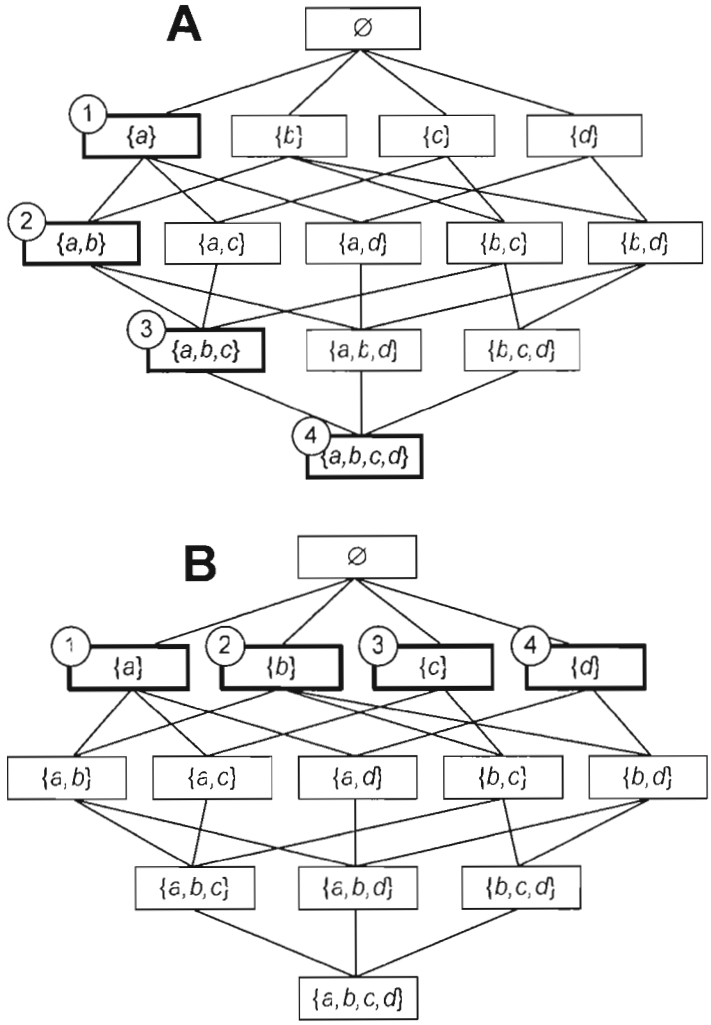

Figure 1.4 : Ordre de traitement des sous-ensembles de l'ensemble {a,b,c,d} dans un algorithme de parcours en profondeur (A) et en largeur (B). Les sous-ensembles sont traités dans l'ordre 1, 2, 3, 4, etc...

Les itemsets candidats sont traités dans l'ordre lexicographique, ce qui assure qu'il n'existera au cours du traitement qu'un seul chemin permettant de passer d'un itemset à l'autre.

L'algorithme Apriori (Agrawal et Srikant, 1994) est un algorithme de parcours en largeur, alors que l'algorithme FP-growth (Han et al., 2000) est un algorithme de parcours en profondeur.

1.8.1 Algorithmes de parcours en largeur de l'arborescence des itemsets

L'algorithme Apriori est basé sur les propriétés suivantes : Premièrement, tous les sous-ensembles de taille *k-1* d'un patron fréquent de taille *k* doivent être eux-mêmes fréquents (par exemple, si on suppose que l'itemset $\{a\}$ n'est pas fréquent, l'élimination préalable des itemsets de la Figure 3 dont $\{a\}$ est un sous-ensemble donne la Figure 5) et, deuxièmement, tous les patrons dont un sous-ensemble est infréquent sont eux-mêmes infréquents.

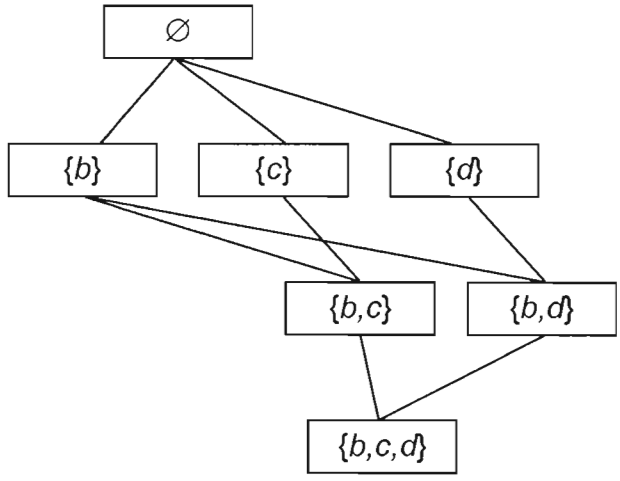

Figure 1.5 : Représentation des sous-ensembles de l'ensemble $\{a,b,c,d\}$ après élagage des sous-ensembles contenant l'élément *a*.

Les patrons sont conservés en mémoire primaire, alors que la base de transactions réside en mémoire secondaire.

Les patrons fréquents de longueur 1 sont d'abord identifiés à l'aide d'un parcours de la base de transactions. L'étape suivante est la génération de patrons « candidats » (donc potentiellement fréquents) à partir des patrons fréquents de taille maximale déjà trouvés. Une fois la génération terminée, un parcours de la base de transactions permet d'éliminer les patrons candidats infréquents. Les patrons candidats fréquents sont alors ajoutés à la liste des patrons fréquents.

Le processus est répété jusqu'à ce que plus aucun patron candidat ne s'avère fréquent.

L'avantage principal de cet algorithme est qu'il se prête bien au traitement de grandes bases de transactions, puisqu'il n'est pas limité par la mémoire primaire. Les inconvénients de l'algorithme Apriori résident dans le grand nombre de comparaisons faites lors de la génération des patrons candidats et dans la nécessité d'accéder à la mémoire secondaire lors des parcours de la base de transactions.

L'algorithme Apriori a été utilisé par Srikant et Agrawal (1995) comme base pour plusieurs algorithmes multi-niveaux tels que Basic (essentiellement l'algorithme Apriori avec une augmentation taxonomique des transactions) Cumulate (décrit au chapitre 2) et EstMerge. Cet algorithme applique l'heuristique d'Apriori (la propriété d'antimonotonicité : qui fait qu'un itemset non-fréquent ne peut pas être un sous-ensemble d'un itemset fréquent) à la taxonomie en élaguant tous les itemsets candidats qui contiennent un descendant taxonomique d'un item non fréquent.

Plusieurs autres algorithmes utilisent une approche similaire à celle d'Apriori, soit un parcours en largeur et la génération et le test d'itemsets candidats, tout en y apportant des optimisations (évaluation de la fréquence des itemsets candidats par opérations ensemblistes sur les transactions : AprioriTID de Agrawal et Srikant (1994), arrêt de l'évaluation de la fréquence d'un itemset candidat dès que le seuil de fréquence minimale est atteinte (Brin *et al.*, 1997), commencer directement par la recherche des itemsets maximaux –fréquents ou non- par intersection ensembliste des identifiants de transactions puis détermination de la fréquence des candidats et élagage des candidats infréquents (Savasere *et al.*, 1995)).

Une famille d'algorithmes dérivés d'Apriori vise la recherche de règles d'association impliquant des itemsets infréquents et emploie des seuils de support minimaux différents pour empêcher la génération de candidats comportant certains items. Ces algorithmes sont dits « à supports minimaux multipes » tels que Mms-Apriori (Liu *et al.*, 1999). Des algorithmes de recherche de règles d'association multi-niveaux à supports minimaux multiples en sont également dérivés : MMS_Cumulate et MMS_Stratify (Tseng et Lin, 2001).

Dans les algorithmes Close et A-Close (Pasquier et al., 1999), les itemsets ne sont pas considérés simplement comme itemsets potentiellement fréquents, mais plutôt en tant que générateurs d'itemsets fermés. Si la fermeture d'un nouveau générateur a déjà été rencontrée, celui-ci peut alors être ignoré. L'avantage de cette approche est que le nombre de candidats à évaluer est considérablement réduit, puisqu'on procède à un élagage considérable de l'espace de recherche en ne conservant que les itemsets fermés fréquents, en se basant sur le fait qu'un itemset fermé est une représentation concise de tous ses sous-ensembles.

1.8.2 Algorithmes de parcours en profondeur de l'arborescence des itemsets

Les algorithmes de parcours en profondeur ont en commun que la base de transactions est conservée en mémoire primaire, ce qui entraîne une diminution du temps de traitement par une réduction considérable des accès à la mémoire secondaire. Cette diminution est cependant contrebalancée par des difficultés lors de l'application de l'implémentation de ces algorithmes à des bases de transactions de taille bien supérieure à celle des jeux d'essai (*mise à l'échelle*). De nombreux algorithmes de parcours en profondeur (ex : Eclat, Viper, Mafia) compilent pour chaque item l'ensemble des identifiants des transactions où il est présent (TIDset). Le support des itemsets candidats peut être calculé en déterminant la cardinalité de l'intersection des TIDset des items présents. Les algorithmes de parcours en profondeur sont réputés plus efficaces que les algorithmes de parcours en largeur, pour

autant que la base de transactions (ou sa représentation) puisse être conservée en mémoire primaire.

L'algorithme Eclat découle de la volonté de ses auteurs (Zaki *et al.*, 1997) de réduire au maximum la taille de la base de transactions dans le but de faciliter la mise à l'échelle. Pour ce faire, les TIDsets sont remplacés par des DIFFsets qui sont des intersections de TIDsets. Comme la taille d'un DIFFset est toujours comprise entre 0 et celle du plus grand des TIDsets considérés, le gain d'espace peut être considérable. Les auteurs remarquent également que ce gain d'espace tend à être proportionnel à la densité de la base de transactions. Une autre variante de cette approche de réduction de taille est l'algorithme Viper (Shenoy *et al.*, 2000) où les TIDsets sont réduits à un tableau de bits.

L'algorithme Prutax (Hipp *et al.*, 1998) utilise une approche hybride, alliant l'emploi d'intersections de TIDsets –comme Eclat- au parcours en largeur –comme Apriori- et à l'organisation des attributs selon une taxonomie –comme Cumulate. Un candidat est élagué dans les cas suivants :

- au moins un des sous-ensembles du candidat est infréquent,
- le candidat est constitué d'un item et d'un de ses ancêtres taxonomiques(dans le cas d'un 2-itemset),
- le parent taxonomique d'au moins un des items du candidat est infréquent

De plus, si deux candidats X et Y ont même cardinalité et que l'on peut obtenir X en remplaçant un (ou plus) des items de Y par un de ses ancêtres taxonomiques, alors Y est élagué.

L'algorithme Mafia (Burdick *et al.*, 2001) utilise une réorganisation dynamique de l'arborescence des itemsets en se basant sur le support des itemsets ; si un fils a le même support que son père, il le remplace dans l'arborescence. Cette opération a pour avantage d'accélérer le traitement en réduisant considérablement le nombre d'itemsets candidats à considérer.

L'algorithme FP-growth (Han *et al.*, 2000) propose de résoudre le problème de l'évaluation des itemsets candidats en éliminant la génération de ces itemsets ; les patrons avérés

fréquents sont stockés dans une arborescence de préfixes (FP-tree). Une fois le traitement terminé, les itemsets fréquents sont extraits en parcourant l'arborescence. FP-growth possède une variante multi-niveaux :FP-tax (Pramudiono et Kitsuregawa, 2004). Dans cet algorithme, chaque transaction est augmentée par l'ajout de tous les ancêtres taxonomiques de ses items.

L'algorithme Closet (Pei *et al.*, 2000) utilise également une arborescence de préfixes, mais plutôt aux fins de générer les itemsets fermés fréquents . Cette opération se fait par un parcours ascendant de l'arborescence des préfixes au cours duquel les items sont parcourus en ordre croissant de fréquence ; les itemsets fermés correspondants à l'ordre décroissant sont alors générés. Cette approche est avantageuse pour les données denses.

L'algorithme Charm (Zaki et Hsiao, 2002) effectue un parcours en profondeur d'une arborescence dont les noeuds sont composés d'un itemset fermé et de son TIDset. Il existe une relation d'équivalence entre itemsets définie par le partage de préfixes communs de même longueur. Cette relation d'équivalence permet le partage du parcours de l'arborescence en sous-problèmes indépendants (puisque ces sous-arbres n'appartiennent pas à la même classe d'équivalence). Dans l'algorithme Charm, un itemset est fréquent si il est un sous-ensemble d'un itemset fermé fréquent dont le TIDset associé a une taille supérieure ou égale au seuil de fréquence minimale.

Jiang et Tan ont récemment (Jiang et Tan, 2006) proposé l'algorithme gp-close, lequel applique la fermeture au problème de la sur-généralisation des règles , problème posé par le fait qu'un item de niveau taxonomique i ait le même support que son parent de niveau taxonomique i-1). Or la pertinence d'une information est souvent proportionnelle à son niveau taxonomique. Par exemple la règle "client" achète "produit" est nettement moins informative que "homme adolescent" achète "iPod vidéo 4Giga". Les auteurs définissent comme sur-généralisée une règle d'association telle qu'il existe une règle plus taxonomiquement spécialisée ayant le même support. Il s'agit donc d'un élagage basé sur les relations taxonomiques de spécialisation/généralisation.

Les auteurs introduisent la notion de fermeture généralisée des relations afin de résoudre le problème de la sur-généralisation. Cette fermeture est définie sur une relation X comme

étant l'ensemble des relations de X et de toutes les généralisations de ces relations. Cette fermeture est dite fermée si X n'est pas sur-généralisée, c'est-à-dire qu'il n'existe pas de généralisation d'une de ses relations ayant le même support.

1.8.3 Algorithmes de construction de treillis de Galois

Un autre groupe d'algorithmes est basé sur les représentations concises et l'application de l'analyse formelle de concepts à la recherche de règles d'associations. Le premier de ceux-ci, proposé par Godin *et al.* (1995), considère l'ensemble de la base de transactions comme une relation binaire item-transaction (ou encore attribut-objet). Le treillis de Galois de cette relation, pouvant être illustré à l'aide d'un diagramme de Hasse, décrit une hiérarchie de concepts formés chacun d'un itemset fermé et de son TIDset associé. L'algorithme proposé par les auteurs permet la construction incrémentale du treillis, une transaction à la fois. On parle alors d'un parcours horizontal de la base de transactions. L'avantage principal de cette approche est qu'elle facilite la découverte des itemsets fermés. De plus, les études empiriques réalisées par les auteurs indiquent que le temps d'exécution de leur algorithme est proportionnel au nombre de concepts déjà présents dans le treillis.

L'algorithme Titanic (Stumme *et al.*, 2002) introduit la notion de treillis d'iceberg qui écarte les concepts dont le support est inférieur à un seuil minimal, et étend l'application de ces treillis aux systèmes de fermeture dont les relations peuvent être pondérées.

Parmi les algorithmes inspirés par celui de Godin *et al,* (2005), on peut citer l'algorithme Magalice (Rouane *et al.*, 2004) qui applique l'algorithme de base aux treillis d'iceberg . Cet algorithme permet également l'identification des générateurs minimaux des concepts du treillis, ce qui est d'une grande importance dans l'éventuelle recherche des règles d'association puisque, comme nous l'avons vu plus haut, les générateurs minimaux sont les prémisses minimales des règles informatives maximales.

L'algorithme Magalice-A (Nehmé *et al.*, 2005) qui est dérivé de l'algorithme Magalice, construit l'iceberg non pas une transaction à la fois, mais plutôt un item à la fois. On parle alors d'un parcours vertical de la base de transactions. La modification de ce dernier algorithme aux fins de recherche de règles d'association multi-niveaux est une de nos contributions dans le cadre de ce travail Une trace d'exécution de cet algorithme modifié est réalisée dans la section 2.2.3

1.9 Notre contribution

Le volet pratique de cette étude porte sur l'adaptation de l'algorithme Magalice-A aux règles multi-niveaux. Nous avons retenu cet algorithme car il combine les avantages de la recherche de concepts fréquents et de leurs générateurs minimaux en plus d'utiliser une représentation verticale de la base de transactions –donc plus facilement adaptable à l'organisation taxonomique des attributs. Nous proposons deux variations de cet algorithme, soit une première version (Magalice-AT) qui permet la prise en charge de données taxonomiques et une deuxième version (Magalice-AT+) qui offre une optimisation du traitement.

Aux fin de comparaison expérimentale, nous avons également implémenté l'algorithme Cumulate puisqu'il est l'algorithme de référence dans plusieurs travaux portant sur les règles d'association multi-niveaux (fp-tax, prutax, mms_cumulate). Notre implémentation de cet algorithme nous servira donc de base de comparaison expérimentale avec nos deux versions de l'algorithme Magalice-A.

CHAPITRE II

ALGORITHMES ÉTUDIÉS

Dans cette section, nous présentons les algorithmes étudiés dans ce travail, c'est-à-dire les algorithmes Cumulate, Magalice-AT et Magalice-AT+. L'idée motrice, le pseudo-code et une analyse de complexité de chacun de ces algorithmes sera présentée. Une trace commentée complètera la présentation.

2.1 Algorithme Cumulate

L'algorithme Cumulate est un algorithme de recherche d'itemsets multi-niveaux fréquents. Il a été proposé par Srikant et Agrawal (Srikant et Agrawal, 1995) et est basé sur l'algorithme Basic (lui-même étant une variante multi-niveaux de Apriori) des mêmes auteurs (Agrawal et Srikant, 1994). Comme l'algorithme Apriori, l'algorithme Cumulate procède par niveau en générant les k-itemsets à partir des k-1-itemsets. Chacun des niveaux requiert un balayage complet de la base de transactions (Godin, 2006), tout en prenant en compte une organisation taxonomique des attributs. Les intrants sont la valeur de support minimal et deux fichiers, l'un étant une énumération des transactions et l'autre une description de la taxonomie des attributs. L'extrant de cet algorithme est l'ensemble des itemsets fréquents correspondant au support minimal spécifié en intrant.

Puisque les règles d'association ne sont pas retournées par l'algorithme Cumulate, un post-traitement est donc nécessaire.

L'avantage principal de l'algorithme Cumulate est qu'il permet de traiter de très grands volumes de données, puisque la base de transactions ne réside pas en mémoire primaire.

Les inconvénients de l'algorithme Cumulate sont de trois ordres : premièrement, la génération des itemsets candidats est très onéreuse. Deuxièmement, les accès à la mémoire

secondaire sont très nombreux, avec des conséquences évidentes sur le temps d'exécution (Han et Kamber, 2001) et, troisièmement, un grand nombre des itemsets fréquents sont redondants (Zaki, *et al.*, 1997).

Premièrement, nous présenterons de façon générale l'intuition servant de base à l'algorithme. Deuxièmement, le traitement effectué par l'algorithme sera expliqué et illustré d'exemples et sa complexité algorithmique est analysée.

2.1.1 Description

L'algorithme Apriori décrit un parcours descendant par niveaux dans l'arborescence de l'ensemble puissance des attributs en évitant un maximum d'itemsets infréquents, les liens de cette arborescence correspondant à la relation « est un sous-ensemble de ». L'ensemble puissance d'un ensemble E est l'ensemble de tous les les sous-ensembles de E. Par exemple, si $E=\{a, b, c, d\}$, alors l'ensemble puissance de E sera égal à $\{\varnothing,\{a\}, \{b\}, \{c\}, \{d\}, \{ab\}, \{ac\}, \{ad\},\{bc\}, \{bd\}, \{abc\}, \{abd\},\{bcd\}, \{abcd\}\}$.

L'algorithme Cumulate ajoute à ce parcours un critère supplémentaire d'élagage du treillis, à savoir qu'un itemset candidat de taille 2 ne peut être composé d'un attribut et de son ancêtre taxonomique, puisqu'un tel itemset ne pourrait que générer une règle d'association redondante telle que, par exemple : *traitement de texte→logiciel* qui est redondante puisque tous les *traitements de texte* sont des *logiciels* (voir Figure 1.2). Autre différence avec l'algorithme Apriori: la fréquence des itemsets candidats de l'algorithme Cumulate est établie en les comparant avec les éléments de la base de transactions augmentée de leurs ancêtres taxonomiques. Le pseudo-code de l'algorithme Cumulate est présenté à la Figure 2.1, alors que celui de la méthode auxiliaire Apriori-gen est présenté à la Figure 2.2.

Les notations suivantes sont utilisées dans le pseudo-code :

- k : taille des itemsets considérés dans le niveau en cours

- C_k : ensemble des k-itemsets fréquents candidats. Chacun des éléments de C_k comporte un sous-élément .count qui indique la fréquence de l'élément
- L_k : ensemble des k-itemsets fréquents
- sauf indication contraire, un indice indique le rang d'un élément (ex : B_3 est le 3^e élément de l'ensemble B)
- sauf indication contraire, une lettre minuscule indique un élément de l'ensemble noté en majuscules (ex : a est un élément de A)

CUMULATE

intrants :

D : ensemble des transactions

T : ensemble des ancêtres taxonomiques de chaque attribut

α: support minimal

extrants :

L : ensemble des itemsets fréquents

variables locales :

A : tableau des attributs

A^* : tableau des ancêtres taxonomiques des attributs

k : longueur des itemsets en cours de traitement

C_k : ensemble des itemsets candidats de taille k

1.

2. **pour chaque** $d \in D$ **faire**

// lignes 2 à 4: initialisation des tableaux A et A^*

3. $A \leftarrow A \cup \{d_i \in d \,|\, d_i \notin A\}$

// tableau des attributs

4. $A^* \leftarrow A^* \cup \{d_i \in d \,|\, d_i \notin A^*\}$

// tableau des ancêtres taxonomiques

5. **pour chaque** $a^* \in A^*$ **faire**

// compilation de tous les ancêtres de chaque attribut

6. $a^* \leftarrow a^*_0 \cup \{t \in T \mid a^*_0 < t\}$

7. $L_1 \leftarrow \{a \in A \,|\, |d \in D \mid a \in d \cup \{a^* \in A^* \mid d_i = a^*_0\}| \geq \alpha\}$

// identification des itemsets fréquents de taille 1

Figure 2.1 : Pseudo-code de l'algorithme CUMULATE

CUMULATE (suite)

8. $k = 2$
9. **tant que** $|L_{k-1}| > 0$ **faire**
10. $C_k \leftarrow$ APRIORI-GEN(L_{k-1}, k) // générer les itemsets candidats de taille k
11. **si** $k = 2$ **alors**
12. $C_k \leftarrow C_k \setminus \{c \in C_k \mid \exists a^* \in A^*, c \subseteq a^*\}$
// optimisation par rapport à l'algorithme BASIC
// éliminer les candidats de taille 2 formés d'un élément
// et d'un de ses ancêtres taxonomiques
13. **pour chaque** $a^* \in A^*$ **faire**
14. $a^* \leftarrow a^* \setminus \{a^*_i \in a^* \mid a^*_i \notin \bigcup_j c_j \in C_k\}$
// optimisation par rapport à l'algorithme BASIC
// enlever du tableau des ancêtres taxonomiques
// les attributs non fréquents
15. **pour chaque** $d \in D$ **faire**
16. $d \leftarrow d \cup \{\{a^* \in A^* \mid d_i \subset a^*\} \setminus \{a^*_1 \mid \exists a^*_2, a^*_1 = a^*_2\}$
// ajouter à chaque transaction les ancêtres taxonomiques
//de chacun de ses attributs
17. **pour chaque** $c \in C_k$ **faire**
//augmenter la fréquence de tous les itemsets
//candidats présents dans la transaction d
18. **si** $c \subset d$ **alors** $c.count{+}{+}$
19. $L_k \leftarrow \{c \in C_k \mid c.count \geq \alpha\}$ // les itemsets candidats de fréquence supérieure
// à α et de taille k forment les itemsets fréquents de
// taille k
20. $k{+}{+}$ // augmenter la taille des candidats à considérer
21. **fin tantque**

Figure 2.1 (suite) : Pseudo-code de l'algorithme CUMULATE

APRIORI-GEN

intrants:

k : taille des itemsets candidats;

L: ensemble des itemsets fréquents de taille $k-1$

extrants:

C: ensemble des itemsets candidats de taille k

1. $C \leftarrow \{Lp \rhd\lhd Lq \mid \forall 0 < i < k-2, (p_i = q_i) \wedge (p_{k-1} < q_{k-1})\}$
   // faire la jointure des itemsets fréquents ne différant que par leur dernier élément
2. **pour chaque** $c \in C$ **faire**
3. **pour chaque** $s \in c$ **tel que** $|s| = k-1$ **faire**
4. **si** $s \notin L$ **alors** $C \leftarrow C \setminus \{c\}$
5. // élaguer les candidats dont un (ou plus) sous-ensemble n'est pas fréquent

Figure 2.2 : Pseudo-code de la méthode auxiliaire APRIORI-GEN

2.1.2 Trace commentée de l'algorithme Cumulate

Nous utiliserons comme exemple au cours de cette trace la base de transactions décrite *in extenso* dans la Figure 1.1 et représentée sous la forme d'un contexte binaire dans le tableau suivant. La taxonomie de la Figure 1.2 y est utilisée pour l'augmentation taxonomique des transactions.

Tableau 2.1 : Représentation sous la forme de contexte binaire de la base de transactions de la Figure 1.1 avant (A) et après (B) augmentation taxonomique.

A

objets	attributs a	b	c	d	e	f	g	h
1		x				x		
2			x					x
3		x	x	x				x
4							x	
5				x	x			
6				x				
7	x							x
8				x	x	x		
9		x						

B

objets	attributs a	b	c	d	e	f	g	h
1		x			x	x		
2		x	x					x
3		x	x	x				x
4					x	x	x	
5				x	x			
6				x				
7	x							x
8				x	x	x		
9		x						

2.1.2.1 Étape 1 – Chargement en mémoire primaire de la taxonomie des attributs (lignes 2 à 6)

La liste des ancêtres de chaque attribut est compilée et les informations sont conservées dans le tableau des ancêtres taxonomiques (tableau A^*, lignes 2 à 6 du pseudo-code). Dans l'exemple, il est noté que l'ancêtre taxonomique de l'attribut *c* est l'attribut *b*, que *e* est l'ancêtre taxonomique de *f* et que *e* et *f* sont ceux de *g*.

2.1.2.2 Étape 2 – Identification des attributs fréquents (ligne 7)

Chaque transaction est lue, puis augmentée par l'ajout des ancêtres taxonomiques de chacun de ses attributs. Une conséquence de cette augmentation taxonomique est qu'un attribut peut ainsi être ajouté à une transaction en tant qu'ancêtre d'un item déjà présent alors qu'il y figure déjà en tant qu'élément de la transaction non-augmentée.

Puisque nous visons à identifier la présence *d'au moins une instance* des attributs dans une transaction, il est donc nécessaire de s'assurer que les attributs d'une transaction augmentée n'y figurent qu'une seule fois chacun. Une fois cette opération réalisée, la fréquence de chaque attribut présent dans la transaction est incrémentée.

Une fois complétée la lecture des transactions, les attributs dont la fréquence correspond à un support supérieur au support minimal retenu constituent chacun un 1-itemset fréquent. L'ensemble de ces itemsets constitue l'ensemble L_1. Plus généralement, la variable k est utilisée pour indiquer la longueur des itemsets considérés. Les itemsets fréquents de longueur k sont contenus dans l'ensemble L_k et les itemsets candidats (c'est-à-dire potentiellement fréquents) sont contenus dans l'ensemble C_k.

Dans notre exemple, si nous utilisons un support minimal de 2, nous aurons :

L_1={*imprimante, imprimante laser, souris, logiciel, bureautique, écran*}, ce qui, en utilisant les abréviations décrites à la Figure 1 et utilisées dans la suite de cette trace, devient : L_1={*b, c, d,e,f, h*}.

2.1.2.2.3 Étape 3- Génération des itemsets *k*-candidats (ligne 10 et méthode auxiliaire APRIORI-GEN)

Les itemsets candidats de taille k (l'ensemble C_k) sont générés tel que décrit par Agrawal et Srikant (1994), soit par la jointure des itemsets fréquents de taille *k-1* tels que leurs *k-2* premiers éléments, en ordre lexicographique, soient identiques (l'ordre lexicographique est utilisé pour éviter de générer deux fois le même candidat), ce qui correspond à la ligne 3 de la méthode auxiliaire APRIORI-GEN. Comme tous les sous-ensembles d'un itemset fréquents sont eux-mêmes fréquents, les *k*-candidats peuvent être obtenus par l'union de deux itemsets fréquents de taille *k-1* ne différant que par leur dernier (donc $k\text{-}1^{ième}$) élément.

Le candidat est alors composé des *k-1* premiers éléments du premier itemset fréquent et du *k-1* ième élément du deuxième itemset fréquent. Plus formellement, l'opération est décrite par :

$$C_k = L_{k-1}\ p \mid L_{k-1}\ q$$

tels que:

$$(p[1] = q[1]) \wedge (p[2] = q[2]) \wedge \ldots \wedge (p[k\text{-}2] = q[k\text{-}2]) \wedge (p[k\text{-}1] < q[k\text{-}1]).$$

L'étape suivante (lignes 4 à 6 de la méthode auxiliaire APRIORI-GEN) consiste à s'assurer que tous les sous-ensembles de taille *k-1* de chaque itemset candidat sont eux-mêmes des itemsets fréquents. Par exemple, si les itemsets fréquents de taille k=2 étaient {*bc, be, bh*}, les itemsets candidats de taille k=3 seraient {*bce, bch, beh*}. L'itemset candidat *bch* serait alors éliminé puisque ses sous-ensembles de taille k=2 sont {*bc, bh, ch*} et que *ch* n'est pas un itemset fréquent.

Pour k=2, le cas est trivial et C_2 est le produit cartésien de L_1 par lui-même soit, dans notre exemple,

$$C_2 = \{bc, bd, be, bf, bh, cd, ce, cf, ch, de, df, dh, ef, eh, fh\}$$

L'explosion combinatoire lors de la génération des *k*-candidats est évidente.

Un cas plus intéressant (et plus restreint) serait le suivant : Supposons que k=4 et que L_3={*bcg, bgh, beg,bfh*}. Nous obtiendrions $C_4=L_3|L_3$={*bcgh, bceg,begh*}

L'itemset fréquent *bfh* ne serait pas retenu lors de la jointure car il ne pourrait pas servir à l'élaboration d'un itemset fréquent de taille 4. En effet, si un itemset fréquent de taille 4 commençant par *bfh* existait (noté *bfh ?*), tous ses sous-ensembles de taille 3 devraient également être fréquents, d'après le principe d'Apriori (Agrawal et Srikant, 1994). Or, ici, les itemserts fréquents de taille 3 sont { *bcg, bgh, beg,bfh* }, tandis que les sous-ensembles *bfh ?* de taille 3 sont { *bfh, bh?, fh ?, bf ?*}. Puisque au moins un de ces sous-ensembles ne fait pas partie de L_3, *bfh ?* ne peut donc pas faire partie de L_4. Il est finalement à noter que l'utilisation de l'ordre lexicographique empêche de considérer plus d'une fois le même itemset (par exemple, *bgf* et *fgb*) et donc d'éviter des traitements redondants.

2.1.2.2.4 Étape 4 – Élagage des itemsets *k*-candidats (lignes 11-14)

Cette étape constitue une optimisation par rapport à l'algorithme Basic.

Si k=2, les candidats composés d'un item et d'un de ses ancêtres taxonomiques sont éliminés afin d'éviter ultérieurement la génération de règles redondantes (vues à la section 1.2). Cette opération n'est plus nécessaire pour k>2, puisque tous les éléments de $C_{k|k>2}$ sont générés à partir de L_2 soit directement (pour k=3) ou indirectement (pour k>3). Par conséquent, dans un 3-itemset fréquent {*b,c,e*}, nous sommes certains d'avoir affaire à trois attributs différents. Dans notre exemple, *bc* et *ef* sont ainsi éliminés.

Les attributs ne faisant partie d'aucun itemset candidat sont retirés du tableau des ancêtres taxonomiques (lignes 13-14). Dans notre exemple, l'attribut *g* est infréquent et il est donc éliminé du tableau des ancêtres taxonomiques. Si *e* était infréquent, il serait retiré de la liste des ancêtres taxonomiques de *f* et de *g* (voir la Figure 1.2). Cette opération vise à restreindre l'augmentation taxonomique des transactions (étape 5) aux seuls attributs fréquents et, ainsi, à réduire le nombre de comparaisons à effectuer dans l'évaluation des fréquences des itemsets candidats.

2.1.2.2.5 Étape 5 – Détermination de la fréquence de chaque itemset *k*-candidat (lignes 15-18)

Chaque transaction est lue à partir de la base de transactions située en mémoire secondaire. Cette transaction est alors augmentée en lui ajoutant en mémoire primaire l'ensemble des ancêtres taxonomiques de chacun de ses attributs.

Plus formellement, nous avons :

$$t = t \cup \bigcup_{i=1}^{|t|} T[t[i]]$$

Par exemple, la transaction #2 de notre base de transactions ($\{c,h\}$) deviendrait $\{\boldsymbol{b},c,h\}$ (les ancêtres des attributs de la transaction d'origine sont en caractères gras).

Chaque itemset candidat inclus dans la transaction augmentée voit ensuite sa fréquence incrémentée. Par exemple, pour la transaction #2 et pour k=2, les itemsets candidats de taille 2 sont comparés à la transaction, soit: {*bd, be, bf, bh, cd, ce, cf, ch, de, df, dh, ef, eh, fh*}. La fréquence des itemsets candidats *bh* et *ch* serait alors augmentée de 1, alors que celle des itemsets candidats *bd, be, bf, cd, ce, cf, de, df, dh, ef, eh* et *fh* ne serait pas augmentée.

2.1.2.2.6 Étape 6 – Élimination des *k*-candidats infréquents (ligne 19)

Les itemsets candidats dont le support est supérieur ou égal au support minimal deviennent alors les itemsets fréquents de taille k (L_k) . Par exemple, pour k=2 et pour un support minimal de 20%, l'itemset candidat *dh* n'est pas retenu car on ne le retrouve que dans une transaction pour un support de 11%, alors que les itemsets candidats *bh, ch, de,* et *ef* seraient conservés puisqu'on les retrouve dans au moins 2 transactions, pour un support supérieur à 20 %.

2.1.2.2.7 Étape 7 – Vérification des conditions de fin d'exécution (lignes 20 et 21)

Si L_k n'est pas vide, k est incrémenté et le traitement reprend à l'étape 3.

Sinon, le traitement est terminé et l'ensemble des itemsets fréquents de la base de transaction est décrit par

$$\bigcup_{i=1}^{k} L_i$$

Les règles d'association peuvent ultimement être déduites des itemsets fréquents en examinant chaque sous-ensemble s de chaque itemset fréquent l. La règle $s \rightarrow (l - s)$ est retenue si sa confiance est supérieure ou égale au seuil de confiance minimale désiré. Par exemple, pour l'itemset fréquent *bch* (c'est-à-dire : « *imprimante*, *imprimante laser*, *écran*»), quelques-unes des règles possibles seraient *imprimante* → {*imprimante laser*, *écran*}, *imprimante laser* → {*imprimante*, *écran*} et *écran* → {*imprimante, imprimante laser*}.

Selon la définition de la confiance d'une règle vue à la section 1.2 et en utilisant les valeurs du Tableau 1, la confiance des la règles que nous venons d'énumérer seraient égales, respectivement, à 2/4=50%, 2/2=100% et 2/3=66%. La règle *imprimante laser* → {*imprimante*, *écran*} est une règle exacte, puisque sa confiance est de 100%. Si le seuil de confiance minimale (minconf) était de 60 %, la règle *imprimante* → {*imprimante laser*, *écran*} serait rejetée et les deux autres seraient conservées.

2.1.3 Complexité algorithmique

Comme nous l'avons vu plus haut, l'algorithme Cumulate est dérivé de l'algorithme Apriori. Gunopulos *et al.* (2003) ont démontré qu'un algorithme trouvant les itemsets fréquents en ordre de taille décroissante (tel qu'Apriori) ne peut pas avoir une complexité

polynomiale. Or, tout algorithme qui *ajoute* des traitements à un algorithme non polynomial est lui-même non polynomial. Il en découle donc que l'algorithme Cumulate a une complexité algorithmique non polynomiale.

Les auteurs remarquent également que le nombre de candidats générés pour une taille k d'itemset candidat est borné par 2^k. Puisque le nombre total d'accès à la base de transactions via la mémoire secondaire dépend uniquement du nombre d'itérations (k), le temps de traitement total de l'algorithme Apriori restera donc relativement faible, pour autant que k reste petit, ce qui suppose un support minimal élevé ou encore des données faiblement corrélées (Pasquier *et al.*, 1999). Gunopulos *et al.* (Gunopulos *et al.*, 2003) démontrent que, dans ces conditions, Apriori (et, par extension, de Cumulate) reste efficace.

2.2 Algorithme MAGALICE-AT

L'algorithme Magalice-AT est une adaptation de l'algorithme Magalice-A pour la recherche de règles d'association multi-niveaux (« T » fait référence à l'utilisation d'une taxonomie).

L'algorithme Magalice-A, proposé dans Nehmé *et al.*, (2005) effectue la construction de treillis d'icebergs de concepts fermés fréquents et en identifie les générateurs. Cet algorithme peut donc servir à la production de représentations concises. La construction du treillis d'iceberg est effectuée de manière incrémentale, un attribut à la fois, en effectuant un parcours vertical de l'arborescence.

L'algorithme Magalice-AT est traité dans cette section en trois étapes : premièrement, nous présentons de façon générale l'intuition servant de base à l'algorithme. Deuxièmement, le traitement effectué par l'algorithme est expliqué et illustré d'exemples et finalement, sa complexité algorithmique est analysée.

2.2.1 Description de l'algorithme MAGALICE-AT

Selon ce que nous avons vu plus haut, l'approche des algorithmes de la « famille » d'Apriori est fondée sur la construction et le parcours de l'arborescence de l'ensemble puissance des attributs (c-à-d générer les itemsets candidats et déterminer leur fréquence), l'élagage de certains éléments non prometteurs (c-à-d l'élagage des candidats dont un sous-ensemble n'est pas fréquent). Cette approche a pour inconvénient que les itemsets fréquents trouvés (et subséquemment utilisés dans la recherche de règles d'association) ne sont pas nécessairement fermés, ce qui entraîne une redondance considérable dans les traitements, puisque la recherche des règles d'associations entraînera l'examen de tous les sous-ensembles des itemsets fréquents, donc plusieurs balayages de la base de transactions. Ce problème sera d'autant plus important que l'itemset fréquent sera de grande taille.

L'approche incrémentale de construction de treillis est fondée, elle, sur l'observation que l'ajout d'un nouvel attribut a à un treillis de concepts fermés $\mathcal{L}(O,A,I)$ muni des relations $f:O\rightarrow A$ et $g:A\rightarrow O$ aura pour effet que le treillis $\mathcal{L}^{+}(O,A^{+},I)$ résultant sera composé de concepts inchangés, de concepts modifiés par l'ajout de a à leur intension, et de nouveaux concepts. La modification de concepts existants est possible car, si $C(X,Y)$ est un concept fermé de $\mathcal{L}$ et a un attribut à ajouter à $\mathcal{L}$ alors $X \cap g(a)$ est fermé et fait partie de $\mathcal{L}^{+}$ (Rouane, 2006) .

L'algorithme Magalice-A construit un treillis de concepts fermés avec leurs générateurs incrémentalement par ajout d'attributs. Initialement, le treillis est égal à $\mathcal{L}(O,\{\varnothing\},I)$ et, une fois le traitement terminé, le treillis devient égal à $\mathcal{L}(O,A,I)$. Tous les éléments de A sont donc ajoutés un à un au treillis $\mathcal{L}$. Tous les concepts du treillis résultant sont des concepts fermés et la cardinalité de leur extension correspond à leur support.

Bien que l'utilisation de concepts fermés simplifie considérablement la découverte des règles d'association par rapport à l'algorithme Cumulate, une lacune subsiste encore : En effet, un treillis de concepts fermés comporte *tous* les concepts fermés, quel que soit leur support, alors que l'algorithme Cumulate ne trouve que les itemsets ayant une fréquence supérieure ou égale à un seuil $0<\alpha\leq 1$ spécifié. Ce n'est cependant pas le cas avec un iceberg de concepts fermés et Nehmé, *et al.*, 2005) ont proposé une version de l'algorithme Magalice-A construisant un treillis d'iceberg . C'est cette version que nous utilisons dans l'algorithme Magalice-AT.

Nehmé *et al.* (2005) ont en effet démontré que tous les générateurs minimaux du treillis d'origine restent des générateurs minimaux dans le treillis modifié. Il en découle que les générateurs minimaux des concepts inchangés restent eux-mêmes inchangés, alors que ceux des concepts modifiés sont susceptibles d'être modifiés à leur tour. Bien entendu, les générateurs minimaux des nouveaux concepts doivent également être calculés.

L'ensemble des générateurs minimaux d'un concept modifié est égal l'ensemble de ses générateurs minimaux dans le treillis d'origine, auquel on ajoute le produit du nouvel

attribut par l'ensemble des générateurs minimaux du concept minimal de la classe d'équivalence du concept (Nehmé, *et al.*, 2005). Or, cette modification d'un concept n'entraîne aucun changement dans sa classe d'équivalence. En conséquence, si le concept d'origine est noté c_1 et le concept modifié est noté c_2, nous avons, (d'après Nehmé, *et al.*, 2005) :

$$gen_2(c_2) = gen_1(c_1) \cup min\left(\bigcup_{\hat{c} \in [c]_{\mathcal{R}} \text{ and } \hat{c} \neq c} gen_1(\hat{c}) \right) \times \{a\}$$

Quand à l'ensemble des générateurs minimaux d'un nouveau concept, il est égal au produit du nouvel attribut par l'ensemble des générateurs minimaux de taille minimale de la classe d'équivalence de son géniteur dans le treillis d'origine. En conséquence, si l'on considère l'application $\gamma : A_2 \rightarrow O_1$ qui associe à un ensemble d'attributs d'un concept modifié l'ensemble des objets associés à ce concept dans le treillis non modifié, nous avons (d'après Nehmé, *et al.*, 2005) :

$$gen_2(c) = min\left(\bigcup_{\hat{c} \in [\gamma(c)]_{\mathcal{R}}} gen_1(\hat{c}) \right) \times \{a\}$$

Comme les générateurs minimaux sont les prémisses minimales de règles d'association, l'extraction de ces règles est donc beaucoup plus rapide et efficace que dans le cas de l'algorithme Cumulate.

L'algorithme MAGALICE-AT utilise l'algorithme INCA-GEN de génération d'icebergs pour effectuer le traitement d'attributs faisant partie d'une taxonomie, comme nous l'avons vu plus haut avec l'algorithme CUMULATE. MAGALICE-AT procède d'abord à un pré-traitement des transactions au cours duquel les ancêtres taxonomiques de tous les attributs présents dans une transaction lui sont ajoutés. Si un ancêtre taxonomique est commun à plus d'un attribut présent dans une transaction, il ne lui est ajouté qu'une seule fois, comme nous l'avons vu plus haut pour l'algorithme Cumulate. L'iceberg est alors construit à partir du contexte binaire augmenté taxonomiquement.

Les notations suivantes sont utilisées dans le pseudo-code :

$\mathcal{R}(c)$: sélecteur de la classe d'équivalence du concept c. $\mathcal{R}(c)$ est l'extension du concept maximal de la classe d'équivalence du concept c

$\bar{\mathcal{L}}^{\alpha}$: treillis d'iceberg de support minimal α

$\bar{C}^{\alpha}$: contexte binaire associé à un treillis d'iceberg de support minimal α

MAGALICE-AT

intrants :

$\bar{\mathcal{L}}^{\alpha} = \langle \bar{C}^{\alpha}, \leq \rangle$: treillis d'iceberg défini par son son contexte ($\bar{C}^{\alpha}$), son support minimal α et sa relation $\leq$

Tx : ensemble de couples (attribut, ensemble d'attributs) décrivant l'arborescence taxonomique

extrant :

$\bar{\mathcal{L}}^{\alpha} = \langle \bar{C}^{\alpha}, \leq \rangle$: treillis d'iceberg intrant mis à jour

variables locales :

attr : attribut

```
1. AJOUTER-ANCETRES(Tx, C̄^α)          //effectuer l'augmentation taxonomique du contexte
2. pour chaque attr de C̄^α faire
3.     INCA-GEN(L̄^α, attr)             // ajouter les attributs un à un au treillis d'iceberg
```

Figure 2.3 : Pseudo-code de l'algorithme MAGALICE-AT

AJOUTER - ANCETRES

intrants :

Tx : ensemble de couples (attribut, ensemble d'attributs) décrivant l'arborescence taxonomique

C : contexte. On considère qu'un contexte est formé d'objets auxquels sont associés un ensemble d'attributs (noté *attribs*)

extrants :

C : contexte augmenté taxonomiquement

variables locales :

ancetres : tableau des ancêtres taxonomiques de chaque attribut

1. **pour chaque** *tx* **de** *Tx* **faire**
2. $ancetres[tx] \leftarrow \left\{ \bigcup_{i=1}^{n} t_i \in Tx \middle| t_i \leq tx \right\}$

   // *ancetres*[*tx*] contient l'ensemble des ancêtres taxonomiques de *tx*
3. **pour chaque** *object* **de** *C* **faire**
4. **pour chaque** $at \in attribs$ **associé à** *object* **faire**

   //ajouter à chaque objet l'ensemble des ancêtres

   //taxonomiques de ses attributs
5. $object \leftarrow object \cup ancetres[at]$
6. **fin pour chaque** *at*
7. **fin pour chaque** *object*

Figure 2.4: Pseudo-code de l'algorithme AJOUTER-ANCETRES

INCA - GEN

intrants :

$\overline{\mathcal{L}}^{\alpha} = \langle \overline{C}^{\alpha}, \leq \rangle$: treillis d'iceberg défini par son son contexte ($\overline{C}^{\alpha}$), son support minimal α et sa relation $\leq$

a : attribut à ajouter au treillis d'iceberg

extrants :

$\overline{\mathcal{L}}^{\alpha} = \langle \overline{C}^{\alpha}, \leq \rangle$: treillis d'iceberg intrant mis à jour

variables locales :

Classes : ensemble des classes d'équivalence

Modifiés : ensemble des concepts modifiés

1. *Classes* ← CALCULER-CLASSES($\overline{C}^{\alpha}$,a) // calculer les classes d'équivalence // du contexte $\overline{C}^{\alpha}$ en fonction de l'attribut a
2. *Modifiés* ← $\varnothing$ // initialiser l'ensemble des modifiés.
3. **pour chaque** θ **de** *Classes* **en ordre croissant d'extension faire** // examiner toutes les classes d'équivalence // calculées à l'étape 1
4. $c \leftarrow \theta.concept\_min$ // c est le concept minimal de la // classe d'équivalence considérée
5. **si** $|\mathcal{R}(c)| = |c.extension|$ **alors** // si le sélecteur de la classe d'équivalence de c // et l'extension de c ont même cardinalité,
6. $Modifiés \leftarrow Modifiés \cup \{c\}$ // alors c est un concept modifié
7. $c.intention \leftarrow c.intention \cup \{a\}$ // et le nouvel attribut est ajouté à l'intention de c
8. **sinon** // sinon, c est géniteur d'un nouveau concept $\hat{c}$ // dont l'extension est le sélecteur de la classe // d'équivalence de c et l'intention est // l'intersection de l'intention de c et de a
9. $\hat{c} \leftarrow nouveauConcept(\mathcal{R}(c), c.intention \cup \{a\})$
10. $\overline{\mathcal{L}}^{\alpha} \leftarrow \overline{\mathcal{L}}^{\alpha} \cup \{\hat{c}\}$ // le nouveau concept $\hat{c}$ est ajouté au treillis d'iceberg
11. **pour tous** $\overline{c}$ **de** θ **tels que** $\overline{c} \prec c$ **faire** // examiner tous les concepts de la classe d'équivalence // faisant partie de la couverture inférieure de c
12. $créer\_arc(\hat{c}, \overline{c})$ //créer un arc entre les concepts $\hat{c}$ et $\overline{c}$
13. **si** $\overline{c} \in Modifiés$
14. $supprimer\_arc(c, \overline{c})$ // si $\overline{c}$ est un concept modifé, supprimer l'arc qui le relie à c
15. **fin pour**
16. $\hat{c}.gén\_min \leftarrow \varnothing$ // initialisation de l'ensemble des générateurs minimaux de $\hat{c}$
17. $\theta.concept\_min \leftarrow \hat{c}$ // le concept minimal de la classe d'équivalence // considérée est remplacé par $\hat{c}$
18. **fin sinon**

(... page suivante)

Figure 2.5: Pseudo-code de l'algorithme INCA-GEN (d'après Nehmé, et al., 2005)

INCA - GEN (suite)

19. $c \leftarrow \theta.concept\_min$ // c prend la valeur du concept minimal de la
// classe d'équivalence considérée
20. $c.gén\_min \leftarrow c.gén\_min \cup \theta.gén\_min$ X $\{a\}$
// on ajoute aux générateurs de c le produit cartésien de
// ses générateurs minimaux et de $\{a\}$

Figure 2.5 (suite) : Pseudo-code de l'algorithme INCA-GEN (d'après Nehmé, et al., 2005)

CALCULER - CLASSES

intrants :

$\overline{C}^{\alpha}$: contexte faisant partie de la définition du treillis d'iceberg de la méthode appelante

a : attribut à ajouter au treillis d'iceberg de la méthode appelante.

extrants :

Classes : ensemble de classes d'équivalence dont les éléments sont des triplets (*ensemble d'objets*, *ensemble de générateurs minimaux*, *concept minimal*)

variables locales :

E : ensemble d'objets

θ : classe d'équivalence

1. **pour chaque** c **de** $\overline{C}^{\alpha}$ **tel que** c **soit fréquent faire**
2. $E \leftarrow c.extension \cap a'$ //E est l'intersection de l'extension du concept c // et des objets associés à l'attribut a
3. $\theta \leftarrow Classes(E)$ //on recherche un éventuelle classe d'équivalence θ // dont l'ensemble d'objets associés serait E.
4. **si** $\theta = NULL$ **alors** //si θ n'existe pas, il faut la créer
5. $\theta \leftarrow nouvelle\_classe\_d'équivalence()$
6. $\theta.concept\_minimal \leftarrow c$ // c est le concept minimal de cette classe
7. $\theta.objets \leftarrow E$ //E est l'ensemble des objets associés à cette classe
8. $Classes \leftarrow Classes \cup \{\theta\}$ // ajouter à l'ensemble des classes d'équivalence //répertoriées
9. **fin si**
10. **si** $\theta.concept\_minimal > c$ **alors** //mise à jour du concept minimal de la classe d'équivalence
11. $\theta.concept\_minimal \leftarrow c$
12. **fin si**
13. $\theta.générateurs\_minimaux \leftarrow Min(\theta.générateurs\_minimaux \cup Gén(c))$ //l'ensemble des générateurs minimaux de la classe θ est égal aux //éléments minimaux de l'union des générateurs minimaux de θ // et des générateurs de c. Si l'intention de c n'a qu'un seul attribut, $Gén(c)=c$
14. **fin pour**

Figure 2.6: Pseudo-code de la méthode CALCULER-CLASSES

2.2.2 Trace commentée de l'algorithme MAGALICE-AT

Dans cette section, nous expliquerons l'exécution de l'algorithme Magalice-AT en expliquant l'évolution de l'iceberg obtenu à partir du contexte binaire de la section 2.1.2 (Tableau 2.1) lors de l'ajout de l'attribut *h*. Aux fins d'illustration, l'iceberg avant et après l'ajout de cet attribut est représenté graphiquement dans les Figures 7 et 8. Les attributs et les ensembles d'attributs sont notés sous forme condensée : *ab* correspond à *a,b*. Les abbréviations de la Figure 1.2 sont utilisées afin de ne pas surcharger le treillis. Les noeuds du treillis correspondent à une association entre un itemset et un ensemble d'attributs. Par exemple, le noeud #5 (I={d}, E={3,5,6,8}, G=[d]) signifie que l'attribut *d* –souris, d'après la Figure 1.2- se retrouve dans les objets 3, 5, 6 et 8 et que le générateur minimal de ce noeud est l'ensemble [*d*].

Les concepts *c* du treillis comportent chacun une intention (notée *c.intention*) et une extension (notée *c.extension*) et sont reliés entre eux par des arcs.

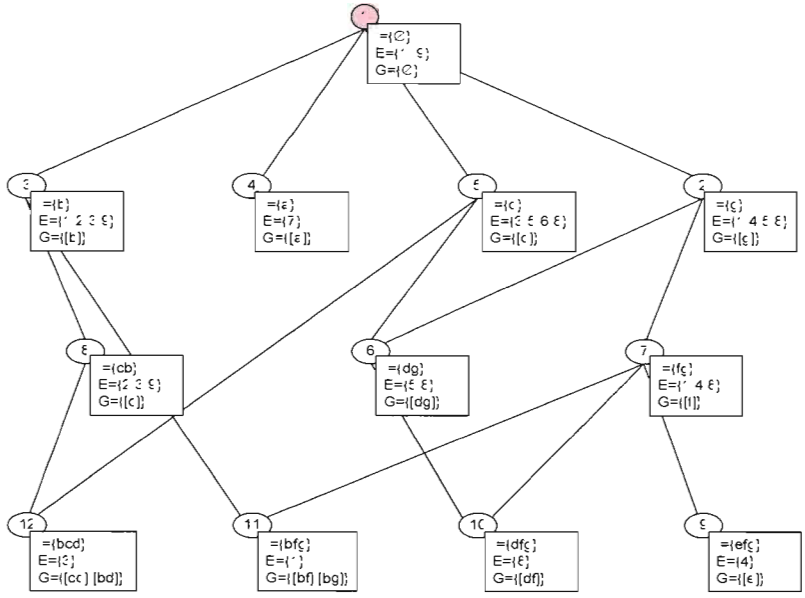

Figure 2.7: Iceberg construit à partir du contexte binaire du Tableau 2.1 pour α=0,99 avant l'ajout de l'attribut *h*. I=intention, E=extension, G=générateurs minimaux. Le concept supremum est en rouge

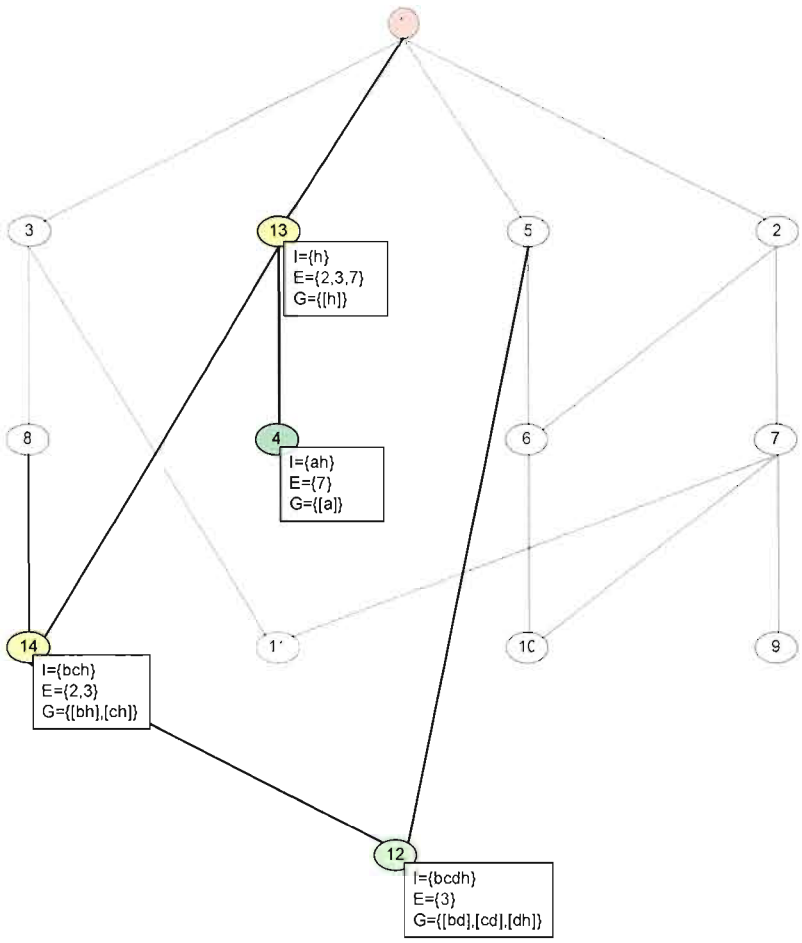

Figure 2.8: Iceberg construit à partir du contexte binaire du Tableau 2.1 pour α=0,99 après l'ajout de l'attribut *h*. I=intention, E=extension, G=générateurs minimaux. Le concept supremum est en rouge. Les concepts ajoutés sont en jaune et les concepts modifiés sont en vert.

2.2.2.1 Étape 1 – Augmentation taxonomique du contexte binaire (ligne 1)

Comme nous l'avons mentionné plus haut, la première étape de l'algorithme Magalice-AT consiste en l'augmentation de la banque de transactions par l'ajout à chaque transaction de l'ensemble des ancêtres taxonomiques distincts de ses éléments. (méthode AJOUTER-ANCETRES). Le format du fichier descripteur de la taxonomie est identique à celui utilisé par l'algorithme Cumulate.

La liste des ancêtres de chaque attribut est compilée et les informations sont conservées dans le tableau *ancetres* (AJOUTER-ANCETRES, ligne 2) et, dans le contexte C, tous les objets déjà en relation avec un attribut sont également mis en relation avec les ancêtres de cet attribut (AJOUTER-ANCETRES, lignes 3 à 7). La nature binaire du contexte fait en sorte que, si deux attributs d'une même transaction possèdent un ancêtre commun, cet ancêtre ne figurera qu'une seule fois dans la transaction augmentée. Par exemple, l'attribut e (*logiciel*) est ajouté à la transaction (l'objet) #1, selon la relation $f \rightarrow e$ puisqu'elle contient l'attribut f (*logiciel bureautique*) et que e n'y figure pas déjà.

2.2.2.2 Étape 2 – Ajout d'un attribut à l'iceberg (ligne 3)

Les attributs sont ajoutés un à un à l'iceberg à l'aide de la méthode INCA-GEN.

2.2.3 Trace commentée de la méthode INCA-GEN

Dans cette méthode auxiliaire, l'ensemble des concepts (fréquents et infréquents) de l'iceberg est parcouru par ordre de support décroissant, ce qui assure un parcours descendant du treillis.

2.2.3.1 Étape 1 – Calcul des classes d'équivalence du contexte augmenté par l'ajout du nouvel attribut (ligne 1)

Cette étape est prise en charge par la méthode CALCULER-CLASSES (Nehmé, *et al.*, 2005). Tous les concepts fréquents sont examinés et l'on tente de trouver une classe d'équivalence telle que son sélecteur soit égal à l'intersection de l'extension du concept considéré et de l'ensemble des objets associés au nouvel attribut (ensemble E). Si il n'existe pas de telle classe d'équivalence, une nouvelle classe d'équivalence est crée avec comme concept minimal le concept considéré et E comme sélecteur (méthode CALCULER-CLASSE, lignes 4 à 9).

Si le concept minimal de la classe d'équivalence est supérieur au concept considéré, celui-ci devient le nouveau concept minimal de la classe d'équivalence (méthode CALCULER-CLASSE, lignes 10 à 12).

Les générateurs minimaux de la classe d'équivalence sont les éléments minimaux de l'union des générateurs minimaux du concept considéré et de ceux de la classe d'équivalence (méthode CALCULER-CLASSE, ligne 13).

2.2.3.2 Étape 3 – Établissement de l'état du concept minimal de la classe d'équivalence (inchangé, modifié ou générateur) (lignes 5 à 10)

Si l'extension du concept minimal de la classe d'équivalence considérée (le concept c) est de même taille que le sélecteur de sa classe d'équivalence, le concept c est un concept modifié (lignes 5 à 8). La modification apportée consistera en l'ajout du nouvel attribut à l'intention du concept c. C'est le cas du concept #4, puisque $\mathcal{R}(C_4(\{1,5,8\},\{b\})) = \{1,5,8\}$, qui est l'extension du concept #4.

Dans le cas contraire, le concept c est le géniteur d'un nouveau concept. L'intention de ce nouveau concept sera égale à l'union de l'intention de c et de l'ensemble formé du nouvel

attribut a, tandis que son extension sera égale au sélecteur de la classe d'équivalence considérée (ligne 9). Par exemple, le concept #2 est géniteur d'un nouveau concept (en l'occurrence, le concept #19), puisque $\mathcal{R}(c_2)$ est inférieur à l'extension du concept ({6,7}>{6}) . Le nouveau concept #19 ayant pour géniteur le concept #2 aura pour extension {6} et pour intension celle du concept #2 ({a}) à laquelle on rajoute le nouvel attribut, ce qui donne {ah}.

En résumé, nous aurons donc :

$c(X,Y)$ modifié $\Rightarrow$ $c(X,Y) \Rightarrow c(X, Y \cup \{a\})$

$c(X,Y)$ générateur $\Rightarrow$ $c^{+}(\mathcal{R}(c), Y \cup \{a\})$

2.2.3.3 Étape 4 – Mise à jour des arcs (lignes 11 à 15)

Ce traitement est nécessaire pour que les relations d'ordre et donc de spécialisation/généralisation entre les concepts soient conservées dans l'iceberg modifié.

Ce traitement consiste en un parcours des concepts minimaux de la classe d'équivalence de c (ligne 11). Un arc est ensuite créé entre tous les concepts considérés et c (ligne 12). Si le concept considéré est un concept modifié, l'arc entre ce concept et c est alors supprimé (lignes 13 et 14). Par exemple, l'arc entre le concept inchangé #7 et le concept modifié #5 a été supprimé (la relation $C_7 > C_5$ est conservée grâce à la transitivité à la relation $C_7 > C_{20} > C_5$ et n'a donc pas besoin d'être explicitée par un arc entre les concepts #5 et #7). Finalement, l'ensemble des générateurs minimaux du nouveau concept est initialisé (ligne 16) et le nouveau concept devient le concept minimal de sa classse d'équivalence (ligne 17).

2.2.3.4 Étape 5 – Mise à jour des générateurs minimaux (lignes 19 et 20)

Lors de cette étape, nous nous assurons que le concept *c* est le concept minimal de sa classe d'équivalence (ligne 19). Cette précaution est nécessaire pour le cas où *c* aurait été géniteur d'un nouveau concept qui serait alors devenu minimal dans la classe d'équivalence.

L'ensemble des générateurs minimaux devient le produit cartésien du nouvel attribut par l'ensemble des générateurs minimaux du concept minimal de la classe d'équivalence du concept (calculé à l'étape 1) (ligne 20).

Par exemple, dans le cas d'un concept modifié, les générateurs minimaux du concept modifié #8 sont [*cd*] (son générateur minimal dans le treillis d'origine) et [*dh*], puisque le concept minimal de la classe d'équivalence du concept #8 dans le treillis d'origine était le concept #9 de générateur minimal [*d*], tandis que dans le cas d'un concept géniteur, le nouveau concept #19 a pour géniteur le concept #2 dont le générateur minimal est [*d*]; en conséquence, le générateur minimal du concept #19 est [*dh*].

2.2.4 Complexité algorithmique

Dans cette section, nous décrivons la complexité algorithmique des deux méthodes auxiliaires, soit AJOUTER-ANCETRES et INCA-GEN, puis celle de l'algorithme principal MAGALICE-AT.

2.2.4.1 Complexité de la méthode AJOUTER-ANCETRES

Soient :

- m le nombre d'attributs distincts de la base de transactions
- n le nombre de transactions (objets)

Puisque le fichier descripteur de la taxonomie contient une entrée et une seule pour chaque attribut (son ancêtre taxonomique immédiat), la complexité de la méthode est de $O(m + 2n)$.

De plus, nous pouvons supposer que l'utilisation de l'antémémoire lors de la lecture des transactions diminuerait significativement l'impact du plus grand nombre d'opérations de lecture/écriture en mémoire secondaire.

2.2.4.2 Complexité de l'algorithme INCA-GEN

La description de l'algorithme INCA-GEN et de sa méthode auxiliaire CALCULER-CLASSES, telle que faite par ses auteurs (Nehmé, *et al.*, 2005), ne comporte pas d'analyse de complexité ; cependant, cet algorithme est une modification de l'algorithme MAGALICE-A de Rouane (2006) – et qui est également un des auteurs de INCA-GEN) qui consiste en l'ajout d'une mise à jour de la liste des générateurs minimaux. Or, si l'on considère

l le nombre de concepts présents dans l'iceberg(après ajout du nouvel attribut)

, cette mise à jour se fait en $O(|l|)$.

De plus, si l'on considère également

m le nombre d'attributs distincts de la base de transactions

n le nombre de transactions (objets)

, la complexité de la méthode MAGALICE-A est de $O(ml . (m + n))$, ce qui est clairement majorant par rapport à $O(|l|)$. En conséquence, la complexité de l'algorithme INCA-GEN peut être établie comme étant de $O(ml . (m + n))$.

2.2.4.4 Complexité de l'algorithme MAGALICE-AT

Puisque la complexité de la méthode INCA-GEN est majorante par rapport à celle de la méthode ADD-ANCESTORS, la complexité de l'algorithme MAGALICE-AT est égale à celle de la méthode INCA-GEN, soit $O(\ ml\ .\ (m + n)\ )$.

2.3 Algorithme Magalice-AT+

L'algorithme Magalice-AT+ est une optimisation de l'algorithme Magalice-AT (d'où le « + »).

Puisque les attributs infréquents sont exclus des icebergs et qu'un *1-itemset* infréquent ne peut pas être un sous-ensemble d'un *k-itemset* fréquent (d'après la propriété d'antimonotonicité utilisée par Apriori (Agrawal et Srikant, 1994 ; Pasquier, *et al.*, 1999)), nous proposons d'éliminer par un pré-traitement les attributs infréquents du jeu de données. Il s'agit d'une approche déjà employée, par exemple par Güvenir *et al.* (Güvenir *et al.*, 2001) pour réduire le nombre d'itemsets candidats générés dans leur modification de l'algorithme Apriori. Une réduction de la taille du jeu de données a le potentiel de diminuer le temps de traitement, puisque l'analyse de la complexité de l'algorithme Magalice-AT a révélé plus haut qu'un des facteurs influençant la durée du traitement est le nombre d'attributs présents dans le jeu de données.

2.3.1 Description

L'algorithme Magalice-AT+ procède de façon similaire à celle de l'algorithme Magalice-AT, soit : un pré-traitement du contexte binaire suivi de l'ajout un par un de tous les attributs à l'iceberg à l'aide de la méthode INCA-GEN décrite plus haut.

Les notations suivantes sont utilisées dans le pseudo-code :

$\mathcal{R}(c)$: sélecteur de la classe d'équivalence du concept c. $\mathcal{R}(c)$ est l'extension du concept maximal de la classe d'équivalence du concept c

$\bar{\mathcal{L}}^{\alpha}$: treillis d'iceberg de support minimal α

$\bar{C}^{\alpha}$: contexte binaire associé à un treillis d'iceberg de support minimal α

MAGALICE-AT+

intrants :

$\bar{\mathcal{L}}^{\alpha} = \langle \bar{C}^{\alpha}, \leq \rangle$: treillis d'iceberg défini par son son contexte ($\bar{C}^{\alpha}$), son support minimal α et sa relation $\leq$

Tx : ensemble de couples (attribut, ensemble d'attributs) décrivant l'arborescence taxonomique

extrant :

$\bar{\mathcal{L}}^{\alpha} = \langle \bar{C}^{\alpha}, \leq \rangle$: treillis d'iceberg intrant mis à jour

variables locales :

attr : attribut

1. AJOUTER-ANCETRES-FREQ($Tx, \bar{C}^{\alpha}$) //effectuer l'augmentation taxonomique du contexte
2. **pour chaque** *attr* **de** $\bar{C}^{\alpha}$ **faire**
3. INCA-GEN($\bar{\mathcal{L}}^{\alpha}$, *attr*) // ajouter les attributs un à un au treillis d'iceberg

Figure 2.9 : Pseudo-code de l'algorithme MAGALICE-AT+

AJOUTER - ANCETRES - FREQ

intrants :

Tx : descripteur de l'arborescence taxonomique

C : contexte. On considère qu'un contexte est formé d'objets (noté *object*) auxquels sont associés un ensemble d'attributs (noté *attribs*)

α : fréquence minimale

extrants :

C : contexte augmenté taxonomiquement

variables locales :

ancetres : tableau des ancêtres taxonomiques de chaque attribut

comptes : tableau de la fréquence de chaque attribut (initialement à 0)

attr _ fréq : ensemble d'attributs fréquents

1. **pour chaque** *tx* **de** *Tx* **faire**
2. $ancetres[tx] \leftarrow \left\{ \bigcup_{i=1}^{n} t_i \in Tx \middle| t_i \leq tx, t_i \notin ancetres[tx] \right\}$

   // *ancetres*[*tx*] contient l'ensemble des ancêtres taxonomiques de *tx*
3. **pour chaque** *object* **de** *C* **faire**
4. **pour chaque** $at \in attribs$ **associé à** *object* **faire**

   //ajouter à chaque objet l'ensemble des ancêtres

   //taxonomiques de ses attributs
5. $object \leftarrow object \cup \left\{ \bigcup_{i=1}^{n} anc_i \in ancetres[at] \right\}$
6. **fin pour chaque** *at*
7. **fin pour chaque** *object*
8. **pour chaque** *object* **de** *C* **faire**
9. **pour chaque** $at \in attribs$ **associé à** *object* **faire**

   //incrémenter la fréquence de chacun

   //des attributs associés à l'objet
10. $comptes[at] \leftarrow comptes[at] + 1$
11. **fin pour chaque** *at*
12. **fin pour chaque** *object*

Figure 2.10: Pseudo-code de l'algorithme AJOUTER-ANCETRES-FREQ

AJOUTER - ANCETRES - FREQ (suite)

13. **pour chaque** *object* **de** *C* **faire**
14. *attr _ fréq* ← ∅ // initialiser l'ens. des attributs fréquents associés // à cet objet
15. **pour chaque** *at* ∈ *attribs* **associé à** *object* **faire**
16. **si** *comptes*[*at*] ≥ α **alors** //ajouter à *attr _ fréq* les attributs fréquents
attr _ fréq ← *attr _ fréq* ∪ {*at*}
17. **fin si**
18. **fin pour chaque** *at*
19. *attribs* ← *attr _ fréq* // remplacer l'ensemble des attributs // associé à l'objet par l'ens. des attributs fréquents
20. **fin pour chaque** *object*

Figure 2.10 (suite) : Pseudo-code de l'algorithme AJOUTER-ANCETRES-FREQ

2.3.2 Trace commentée de la méthode MAGALICE-AT+

Puisque la méthode MAGALICE-AT+ ne diffère de la méthode MAGALICE-AT que par le pré-traitement où la méthode AJOUTER-ANCETRES est remplacée par la méthode AJOUTER-ANCETRES-FREQ, la trace commentée sera donc réduite à la seule trace commentée de la méthode AJOUTER-ANCETRES-FREQ.

2.3.2.1 Trace commentée de la méthode AJOUTER-ANCETRES-FREQ

La méthode AJOUTER-ANCETRES-FREQ vise à augmenter la base de transactions (représentée par le contexte binaire) par l'ajout à chacun des ancêtres taxonomiques des attributs la constituant, puis à comprimer ces transactions en y retranchant les attributs infréquents.

2.3.2.2 Description

La liste des ancêtres de chaque attribut est compilée et les informations sont conservées dans le tableau *ancetres* (AJOUTER-ANCETRES-FREQ, ligne 2) et, dans le contexte C, tout les objets déjà en relation avec un attribut sont également mis en relation avec les ancêtres de cet attribut (AJOUTER-ANCETRES-FREQ, lignes 3 à 6). La nature binaire du contexte fait en sorte que, si deux attributs d'une même transaction possèdent un ancêtre commun, cet ancêtre ne figurera qu'une seule fois dans la transaction augmentée. Par exemple, l'attribut e (*logiciel*) est ajouté à la transaction (l'objet) #1, selon la relation $f \rightarrow e$ puisqu'elle contient l'attribut f (*logiciel bureautique*) et que e n'y figure pas déjà.

La base de transactions est ensuite parcourue une deuxième fois et la fréquence de chaque attribut distinct est alors calculée (lignes 8 à 12). Lors d'un troisième parcours, les attributs infréquents sont alors retranchés des transactions (lignes 13 à 20).

Par exemple, si le support minimal employé correspondait à trois occurrences, le contexte binaire utilisé par l'algorithme INCA-GEN serait tel que représenté dans le Tableau 2. Les attributs *a* et *d* ne comptant pas au moins trois occurrences sont éliminés de la base de transactions.

Tableau 2.2 : Représentation sous la forme de contexte binaire de la base de transactions après augmentation taxonomique et élimination des objets des attributs ayant moins de trois occurrences

		attributs							
		a	**b**	**c**	**d**	**e**	**f**	**g**	**h**
	0			X				X	
	1		X	X					X
	2					X	X	X	
o	**3**		X	X					X
b	**4**		X	X					X
j	**5**		X						X
e	**6**								X
t	**7**			X					
s	**8**		X			X	X	X	X
	9					X			
	10			X					X
	11			X			X		

2.3.3 Complexité algorithmique

Dans cette section, nous décrivons la complexité algorithmique de la méthode auxiliaire AJOUTER-ANCETRES-FREQ, puis celle de l'algorithme principal MAGALICE-AT+. Le calcul de la complexité algorithmique de la méthode auxiliaire INCA-GEN est omis ici puisque déjà vu précédemment à la section 2.2.4.2

2.3.3.1 Complexité de la méthode AJOUTER-ANCETRES-FREQ

Soient :

m le nombre d'attributs distincts de la base de transactions

n le nombre de transactions (objets)

Par rapport à la méthode AJOUTER-ANCETRES, l'élagage des attributs infréquents requiert un parcours supplémentaire de la base de transactions en lecture, puis un autre en écriture, ce qui nous donne une compexité de $O(m + 4n)$, ce qui est asymptotiquement équivalent à celle de la méthode AJOUTER-ANCETRES.

De plus, nous pouvons encore une fois supposer que l'utilisation de l'antémémoire lors de la lecture des transactions diminuerait significativement l'impact du plus grand nombre d'opérations de lecture/écriture en mémoire secondaire.

2.3.3.2 Complexité de l'algorithme MAGALICE-AT+

Puisque la complexité de la méthode INCA-GEN est majorante par rapport à celle de la méthode AJOUTER-ANCETRES-FREQ, la complexité de l'algorithme MAGALICE-AT est égale à celle de la méthode INCA-GEN, soit $O(\ ml\ .\ (m + n)\ )$.

CHAPITRE III

EXPÉRIMENTATION

Nous présentons dans cette section dans un premier temps l'implémentation que nous avons réalisée des algorithmes Cumulate, Magalice-AT et Magalice-AT+, ainsi que les environnements (matériel et logiciel) dans lesquels les ces implémentations ont été faites.

Dans un deuxième temps, nous présentons et analysons les résultats de l'évaluation expérimentale de ces implémentations.

3.1 Environnement expérimental

Nous décrivons dans cette section l'environnement expérimental (matériel et logiciel), ainsi la génération des jeux de données utilisés dans l'évaluation des implémentations des algorithmes.

3.1.1 Matériel

L'expérimentation a été faite sur un micro-ordinateur Compaq Presario 3240CA doté d'un microprocesseur AMD Athlon64 3200+ cadencé à 2GHz et doté de 1,25Go de mémoire vive.

3.1.2 Logiciel

Le système d'exploitation utilisé est SuSe Linux Enterprise server 9.3 (version 64 bit) équipé d'une machine virtuelle Java version 5 (version 64 bit). L'utilisation d'une version 64 bit du système d'exploitation et de la machine virtuelle nous a offert des possibilités

d'allocation de mémoire plus importantes qu'avec les versions 32 bit (>2Go), ce qui a été utile lors de l'implémentation des algorithmes Magalice-AT et –AT+ où l'ensemble des données doit résider en mémoire primaire. De plus, l'allocation de mémoire sur une plate-forme Windows se fait par blocs qui ne sont pas nécessairement contigus, contrairement à ce qui est requis par la machine virtuelle Java et à ce qui se fait sous Linux, Linux nous a donc garanti l'absence de permutation entre la mémoire primaire et la mémoire secondaire en cours d'exécution (ce qui aurait introduit des aberrations dans la mesure des temps d'exécution), contrairement à Windows.

Les implémentations des algorithmes Cumulate, Magalice-AT et Magalice-AT+ ont été faites en Java (jdk 1.5.0_05) à l'aide de l'environnement de programmation JBuilder 2006 de Borland (version « enterprise »).

L'algorithme Cumulate impliquant de nombreux accès au disque, nous avons éliminé l'effet de cache d'entrée/sortie en mesurant indépendamment le temps de lecture de la base de transactions par le programme et en ajoutant cette valeur, multipliée par le nombre d'itérations, au temps de traitement obtenus.

L'algorithme IncA-Gen utilisé par les algorithmes Magalice-AT et –AT+ l'a été dans le cadre de la plate forme Galicia 3 (Galicia, 2006) sous le même environnement (système d'exploitation, machine virtuelle Java) que les autres algorithmes implémentés.

Le générateur de données synthétiques a été compilé avec le compilateur C++ g++ (version 3.3.3) sous SuSe Enterprise server 9.3.

3.1.3 Génération des jeux de données

Nous avons utilisé dans cette étude des jeux de données obtenus à l'aide du générateur de données synthétiques de Srikant et Agrawal (Srikant et Agrawal, 1995). Ce générateur a été utilisé dans de nombreuses études (ex : Pasquier, *et al.*, 1999, Zaki, *et al.*, 1997, Ben Yahia, Hamrouni et Mephu Nguifo, 2006).

Ce générateur permet de vérifier l'influence de nombreux paramètres sur la performance d'algorithmes :

- le nombre de transactions (objets)
- le nombre d'attributs
- le nombre de racines (attributs n'ayant aucun parent taxonomique)
- la hauteur de l'arborescence taxonomique
- la taille moyenne des transactions (nombre moyen d'attributs par objet)
- la taille maximale des itemsets potentiellement fréquents
- le nombre maximal d'itemsets potentiellement fréquents (« potentiellement » car aucun support minimal n'est spécifié)
- le nombre moyen de descendants taxonomiques de chaque attribut . Il s'agit là du nombre de fils immédiats de l'attribut dans l'arborescence taxonomique
- le facteur de distribution taxonomique du choix des attributs (probabilité qu'un attribut du niveau *i* soit choisi divisée par la probabilité qu'un attribut de niveau *i*+1 soit choisi)

Le générateur de transactions est basé sur l'observation que, dans la réalité, les consommateurs ont tendance à acheter des items en tant qu'ensembles d'items et non pas en tant qu'items individuels (Agrawal et Srikant, 1994). Une transaction est donc principalement composée d'un ou plus de ces ensembles. Un exemple serait "*cahier*"+"*stylo*"+"*règle*"+"*liquide correcteur*"+"*crayon*"+"*gomme à effacer*". Ces ensembles d'items fréquemment groupés correspondent aux itemsets maximaux fréquents du générateur de jeux de données.

La relation taxonomique entre les items est modélisée par un ensemble d'arborescences. Le nombre d'arborescences correspond au paramètre "nombre de racines". La construction de chaque arborescence se fait de la racine vers les feuilles.

Le nombre d'enfants d'un noeud intermédiaire est fixé à l'aide d'une distribution de Poisson dont la valeur moyenne est égale au paramètre "nombre de descendants par noeud". La construction de l'arborescence se fait en attribuant les enfants aux racines, puis aux enfants de niveau 2, et ainsi de suite jusqu'à ce que tous les attributs aient été insérés dans l'arborescence taxonomique. Une des conséquences de cette méthodologie est que toutes les feuilles d'une arborescence ne sont pas nécessairement au même niveau. De plus, si le nombre de niveaux n'a pas été spécifié comme paramètre lors de l'exécution du générateur de jeux d'essai, le réel de niveaux dépendra du nombre d'attributs et du nombre de descendants par noeud.

Chaque noeud comporte un poids qui correspond à la probabilité qu'il soit choisi lors de la génération d'un itemset potentiellement fréquent (IPF). Les poids sont attribués aux nœuds de telle sorte que le poids d'un nœud interne soit égal à la somme des poids de tous ses descendants, divisée par le paramètre "facteur de distribution taxonomique du choix des attributs" (FDT). Si la valeur de ce paramètre est grande, les items auront tendance à être choisis parmis les feuilles des arborescences, tandis que si cette valeur est faible, les items auront tendance à être choisis parmi les nœuds plus proches des racines.

Les itemsets potentiellement fréquents (IPF) sont générés de la façon suivante: la taille de l'IPF est déterminée à partir d'une distribution de Poisson dont la moyenne est égale au paramètre "taille moyenne des itemsets potentiellement fréquents". Puisque des itemsets de grande taille ont souvent des items en commun, une partie des items de l'IIPF précédent (déminée par le paramètre "facteur de corrélation") est copiée dans le nouvel IPF. Cette opération entraîne qu'il va exister des sous-ensemble d'IPFs dont la fréquence sera supérieure à celle de l'IPF dont ils sont dérivés. Le reste des items de l'IPF sont choisis au hasard, comme le sont la totalité des items du premier IPF généré.

Tous les IPF n'ont pas la même probabilité de faire partie d'une transaction; chacun d'entre eux se voit attributé un poids obtenu à partir d'une distribution exponentielle de moyenne 1 (les poids sont normalisés de telle sorte que leur somme soit 1). Le poids d'un IPF correspond à la probabilité qu'il soit choisi lors de la composition d'une transaction.

Lors de l'ajout d'un IPF à une transaction, celui-ci est "spécialisé": chaque item x qui correspond à un nœud interne ou à une racine taxonomique est remplacé par une feuille dont il est l'ancêtre. Ce choix se fait par un parcours descendant du sous-arbre ayant pour racine l'item x. Dans ce parcours, la probabilité qu'un chemin suivi sera proportionnelle à la somme des poids des nœuds de chaque sous-arborescence.

Chaque IPF comporte finalement un indice de corruption (C) qui permet de simuler le fait que tous les items d'un itemset fréquent ne sont pas toujours achetés lors d'une même transaction. Cet indice a une valeur comprise entre 0 et 1 et est obtenu à partir d'une distribution normale de moyenne 0.5 et de variance 0.1. Lors de l'ajout d'un IPF à une transaction, chaque item de cet IPF a une probabilité C de ne pas être ajouté à la transaction.

Les transactions du jeu d'essai sont construites de la façon suivante: la taille (en items) de la transaction est déterminée à partir d'une distribution de Poisson dont la valeur moyenne est égale au paramètre "taille moyenne des transactions". Les IPF sont choisis pseudo-aléatoirement (voir plus haut) et ajoutés à la transaction jusqu'à ce que la taille fixée soit atteinte. Si l'ajout du dernier IPF ferait passer la taille de la transaction au-dessus de la limite fixée, cet IPF sera ajouté quand même à la transaction dans 50% des cas et, autrement, ajouté à la transaction suivante.

Les valeurs par défaut des paramètres du générateur sont donnés dans le tableau suivant.

Tableau 3.3 : Valeurs par défaut des paramètres du générateur de données synthétiques

paramètre	**valeur par défaut**
nombre de transactions	1,000,000
nombre moyen d'items par transaction	10
nombre d'items distincts	100,000
nombre de racines	250
nb moyen de descendants taxonomiques par attribut	5
facteur de distribution taxonomique	1
nombre d'itemsets potentiellement fréquents	1,000
taille moyenne des IPF maximaux	4

3.2 Résultats expérimentaux

Les jeux d'essai utilisés dans notre expérimentation ont été générés en employant les valeurs par défaut (voir Tableau 1) sauf lorsqu'indiqué. Comme le nombre de permutations possibles entre les différents paramètres est considérable, nous avons concentré notre étude sur les paramètres étudiés en détail par Srikant et Agrawal (1995) dans leur description de l'algorithme Cumulate. Certaines modifications ont cependant été nécessaires, notamment en ce qui concerne le nombre d'attributs et d'objets ; en effet, les expérimentations de Srikant et Agrawal (1995) ont été faites en C, alors que les nôtres l'ont été en Java où l'espace mémoire adressable est comparativement réduit. L'implémentation à notre disposition de l'algorithme IncA-gen (utilisée par notre implémentation des algorithmes Magalice-AT et Magalice-AT+) conserve l'ensemble des données en mémoire primaire et s'est avérée incapable de traiter les jeux d'essai produits en utilisant les valeurs par défaut.

Compte tenu des différences dans les environnements logiciels, les tendances des résultats obtenus pour l'algorithme Cumulate sont les mêmes que celles décrites par Srikant et Agrawal (1995).

3.2.1 Taille maximale des itemsets fréquents

Contrairement aux algorithmes Magalice-AT et Magalice-AT+ dont la performance dépend strictement du nombre d'objets, d'attributs et du nombre de concepts du treillis, l'algorithme Cumulate est celui dont la performance a la plus grande chance d'être influencée par la taille maximale des itemsets fréquents. En effet, le nombre d'itérations réalisées lors de son exécution est égal à la taille maximale des itemsets fréquents découverts.

Dû au fait que, lors du processus de génération du jeu de données, plus d'un itemset potentiellement fréquent (voir 3.1.3) peut être inclus dans une transaction, il est possible de

retrouver après traitement par nos algorithmes des itemsets fréquents de taille supérieure à celle spécifiée.

3.2.1.1 Résultats

Le jeu de données utilisé emploie les paramètres par défaut modifiés comme suit : 100 attributs, 10% des attributs sont des racines, le nombre de niveaux de l'arborescence taxonomique est de 3 (déterminé automatiquement par le générateur en fonction du nombre d'objets et du nombre de descendants par objet) et le support minimal est de 3%.

Le résultat de l'exécution de l'implémentation des trois algorithmes sur ce jeu de données est présenté dans la figure suivante.

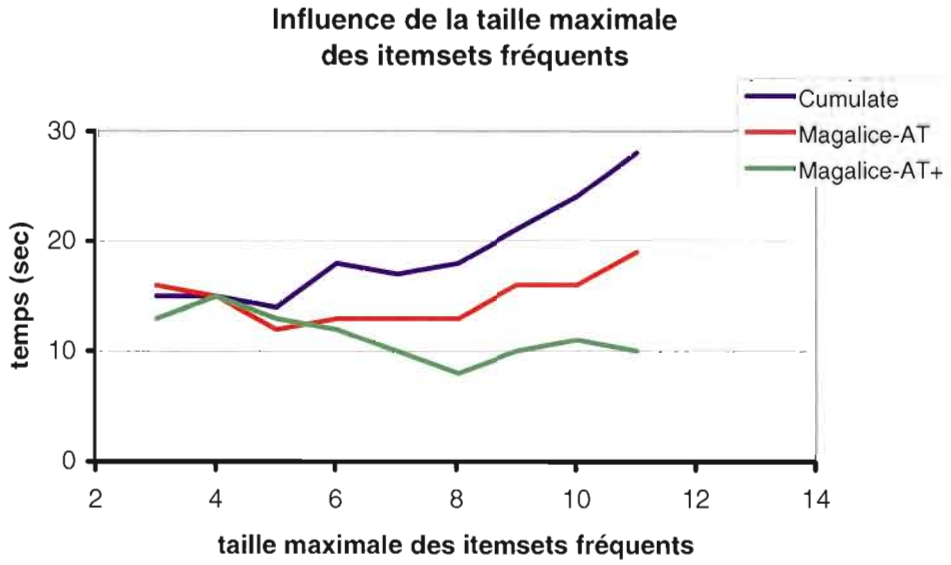

Figure 3.11 : Influence de la taille maximale des itemsets fréquents sur le temps d'exécution des algorithmes.

3.2.1.2 Analyse des résultats

La performance des algorithmes Magalice-AT et –AT+ reste similaire , ce qui reflète leur indépendance vis-à-vis ce paramètre ; en effet, il n'influence ni le nombre d'attributs fréquents, ni le nombre de concepts (fréquents ou non) ni, évidemment, le nombre d'objets ou d'attributs.

L'augmentation du temps de traitement observée avec l'algorithme Cumulate s'explique par le fait que la complexité du traitement dans les algorithmes par niveau a comme borne 2^k, où k correspond à la taille maximale des itemsets fréquents (Gunopulos, *et al.*, 2003). Les auteurs en concluent également que, si k est très petit, la performance de ces algorithmes reste bonne, tel que l'indique la meilleure performance de Cumulate que celle de Magalice-AT, pour $k < 4$.

3.2.2 Nombre d'objets

Le nombre d'objets est généralement un facteur important de modification de performance des algorithmes de recherche de règles d'association lors des mises à l'échelle. L'algorithme Cumulate diffère des algorithmes Magalice-AT et Magalice-AT+ sur ce point en ce que la base de transactions est accédée de manière répétitive en mémoire secondaire, ce qui implique que le temps d'exécution de l'algorithme Cumulate devrait augmenter linéairement par rapport à cet aspect du traitement.

En ce qui concerne les algorithmes Magalice-AT et Magalice-AT+, l'analyse de complexité de ces algorithmes indique, nous l'avons vu plus haut, que le nombre d'objets influencerait linéairement la performance des algorithmes Magalice-AT et Magalice-AT+.

3.2.2.1 Résultats

Le jeu de données utilisé emploie les paramètres par défaut modifiés comme suit : 100 attributs, 10% des attributs sont des racines, la taille maximale des itemsets fréquents est de 4, le nombre de niveaux de l'arborescence taxonomique est de 3 (déterminé automatiquement par le générateur en fonction du nombre d'objets et du nombre de descendants par objet) et le support minimal est de 3%

Le résultat de l'exécution de l'implémentation des trois algorithmes sur ce jeu de données est présenté dans la figure suivante.

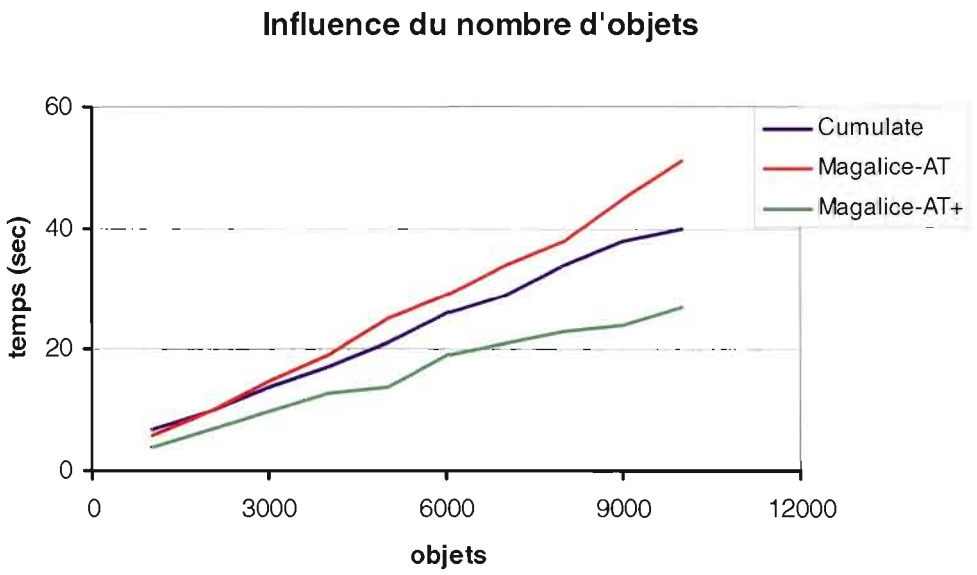

Figure 3.12: Influence du nombre d'objets sur le temps d'exécution des algorithmes.

3.2.2.2 Analyse des résultats

Nous constatons que le temps d'exécution de l'algorithme Cumulate est, pour des jeux de données de moins de 2000 objets, inférieur à celui des algorithmes Magalice-AT et Magalice-AT+. Cette meilleure performance peut s'expliquer par les « coûts fixes » de mise en place des structures de données de la méthode INCA-GEN, alors que l'algorithme Cumulate encourt des coûts d'entrée/sortie qui sont directement proportionnels au nombre d'objets.

La meilleure performance de l'algorithme Magalice-AT+ peut s'expliquer, quant à elle, par le fait que l'élagage préalable des 1-itemsets infréquents diminue le nombre d'attributs à traiter, et, donc, le nombre d'itérations de la méthode INCA-GEN.

3.2.3 Support minimal

Le support minimal d'un itemset représente le rapport du nombre de transactions divisé par le nombre total de transactions. Le nombre d'itemsets fréquents est donc directement proportionnel à la valeur du support minimal.

Dans le cas de l'algorithme Cumulate, une diminution de la valeur de ce paramètre entraîne, lors de chaque itération, le choix d'un plus grand nombre de k-itemsets candidats. Le contraire est vrai dans le cas d'une augmentation de la valeur du support minimal.

Dans le cas des algorithmes Magalice-AT et Magalice-AT+, un faible support minimal augmente le nombre de concepts fréquents, donc une diminution du nombre de concepts exclus du traitement. En conséquence, une faible valeur de support minimal devrait augmenter le temps de traitement.

Puisqu'il exclut *a priori* les attributs infréquents, l'algorithme Magalice-AT+ assure que le nombre d'attributs à traiter sera plus petit que pour l'algorithme Magalice-AT. Puisque l'analyse de complexité algorithmique de ces algorithmes nous indique que le temps de

traitement est fonction linéaire du nombre de concepts, le temps de traitement de l'algorithme Magalice-AT+ devrait lui aussi être plus court que celui de Magalice-AT.

3.2.3.1 Résultats

Le jeu de données utilisé emploie les paramètres par défaut modifiés comme suit : 3,000 objets, 100 attributs, 10% des attributs sont des racines, le nombre de niveaux de l'arborescence taxonomique est de 3 (déterminé automatiquement par le générateur en fonction du nombre d'objets et du nombre de descendants par objet) et la taille maximale des itemsets fréquents est de 4.

Le résultat de l'exécution de l'implémentation des trois algorithmes sur ce jeu de données est présenté dans la figure suivante.

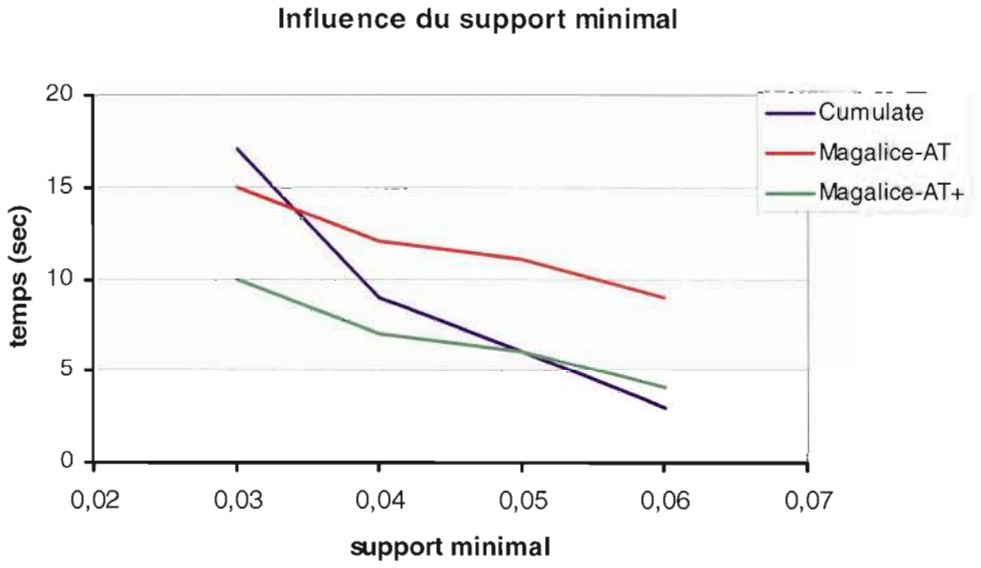

Figure 3.13: Influence du support minimal sur le temps d'exécution des algorithmes.

3.2.3.2 Analyse des résultats

L'augmentation du temps d'exécution des algorithmes Magalice-AT et Magalice-AT+ s'explique par le fait que l'augmentation de la valeur du support minimal diminue le nombre de concepts fréquents. Or, dans ces algorithmes, une part importante du traitement (la méthode CALCULER-CLASSES) est réservée aux concepts fréquents. Il en découle qu'une diminution du nombre de concepts fréquents entraîne effectivement une accélération du traitement, et inversement. Encore une fois, la meilleure performance de Magalice-AT+ est dûe à l'élimination préalable des attributs non fréquents, donc une diminution du nombre total d'attributs à traiter.

En ce qui concerne l'algorithme Cumulate, un faible support minimal entraîne une augmentation du nombre des 1-itemsets candidats, augmentation qui se répercute effectivement de façon exponentielle sur le nombre des itemsets candidats de taille supérieure et, conséquemment, sur le temps d'exécution.

3.2.4 Nombre moyen de descendants taxonomiques par attribut

L'augmentation du nombre moyen de descendants taxonomiques par attribut a pour effet d' « aplatir » l'arborescence taxonomique en augmentant la proportion de feuilles et de nœuds intermédiaires de bas niveau, au détriment des racines et des nœuds intermédiaires de haut niveau.

Puisque, de par l'effet de généralisation de la taxonomie, les items fréquents ont tendance à se retrouver dans les niveaux taxonomiques supérieurs (Srikant et Agrawal, 1995), une augmentation du nombre de descendants taxonomiques par attribut aura pour effet de diminuer le nombre relatif des attributs de niveau taxonomiquement supérieur mais d'augmenter le nombre de transactions comportant ces attributs. En conséquence, les transactions augmentées taxonomiquement auront donc tendance à comporter moins

d'attributs lorsque le nombre moyen de descendants taxonomiques par attribut augmentera (et inversement dans le cas contraire).

3.2.4.1 Résultats

Le jeu de données utilisé emploie les paramètres par défaut modifiés comme suit : 3,000 objets, 100 attributs, 10% des attributs sont des racines, la taille maximale des itemsets fréquents est de 4, le nombre de niveaux de l'arborescence taxonomique est de 3 (déterminé automatiquement par le générateur en fonction du nombre d'objets et du nombre de descendants par objet) et le support minimal est de 3%

Le résultat de l'exécution de l'implémentation des trois algorithmes sur ce jeu de données est présenté dans la figure suivante.

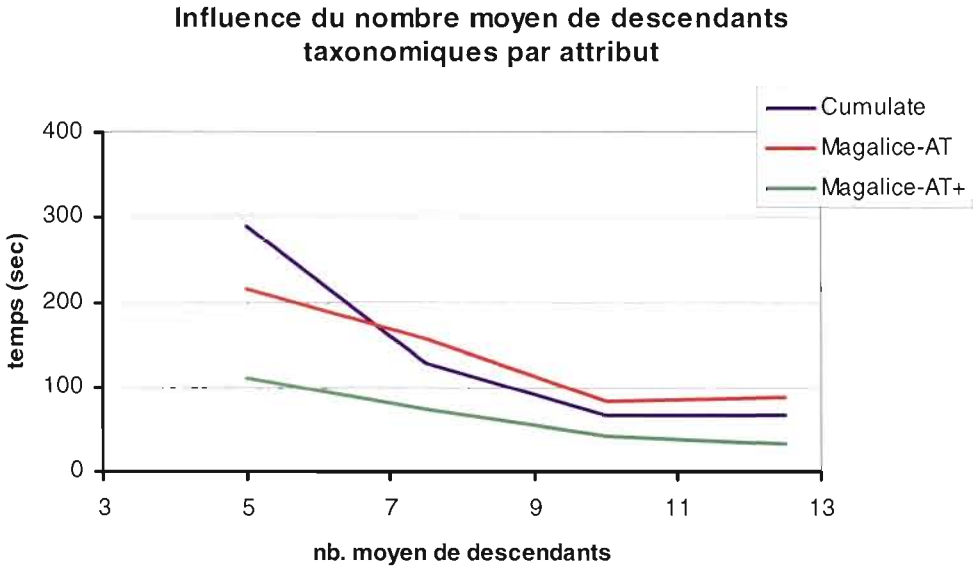

Figure 3.14 : Influence du nombre moyen de descendants taxonomiques par attribut sur le temps d'exécution des algorithmes.

3.2.4.2 Analyse des résultats

Nous constatons à l'examen des résultats que la diminution du temps d'exécution de notre implémentation de l'algorithme Cumulate est initialement beaucoup plus rapide que celle des deux autres algorithmes. D'autre part, l'évolution des temps d'exécution des trois algorithmes est similaire à partir d'une moyenne de 10 descendants taxonomiques par attribut.

Ces résultats s'expliquent d'une part par l'effet de l'augmentation du nombre moyen de descendants taxonomiques par attribut sur l'augmentation taxonomique des transactions pour les algorithmes Magalice-AT et –AT+, puisqu'une diminution de la taille moyenne des transactions taxonomiquement augmentées entraîne une diminution de la densité du contexte binaire et, conséquement, une diminution du temps d'exécution (Valtchev *et al.*, 2002).

D'autre part, la diminution du nombre d'items fréquents distincts associée à l'augmentation du nombre moyen de descendants taxonomiques par attribut entraîne pour l'algorithme Cumulate une diminution exponentielle du nombre d'itemsets candidats générés, diminution qui se reflète dans la diminution du temps d'exécution observée.

3.2.5 Facteur de distribution taxonomique

Le facteur de distribution taxonomique est défini comme la probabilité qu'un item provienne du niveau taxonomique i divisée par la probabilité qu'il provienne du niveau taxonomique i+1.

Une augmentation de la valeur de ce paramètre favorise, lors de la génération des itemsets fréquents, le choix d'attributs proches des feuilles de l'arborescence taxonomique, alors qu'une diminution favorise le choix d'attributs proches des racines de l'arborescence taxonomique.

Ces changements dans la distribution du choix des attributs ont pour conséquence qu'une diminution de la proportion des attributs « feuilles » aura pour effet d'augmenter la proportion d'attributs fréquents de niveau taxonomique plus élevé. Comme les attributs de niveau taxonomique plus élevé (donc plus proches des racines) sont évidemment moins nombreux que les attributs de niveau taxonomique plus bas (donc plus proches des feuilles). Une augmentation de la valeur du facteur de distribution taxonomique aura donc tendance à augmenter le nombre d'attributs fréquents.

3.2.5.1 Résultats

Le jeu de données utilisé emploie les paramètres par défaut modifiés comme suit : 100 attributs, 10% des attributs sont des racines, la taille maximale des itemsets fréquents est 4, le nombre de niveaux de l'arborescence taxonomique est de 3 (déterminé automatiquement par le générateur en fonction du nombre d'objets et du nombre de descendants par objet) et le support minimal est de 3%

Le résultat de l'exécution de l'implémentation des trois algorithmes sur ce jeu de données est présenté dans la figure suivante.

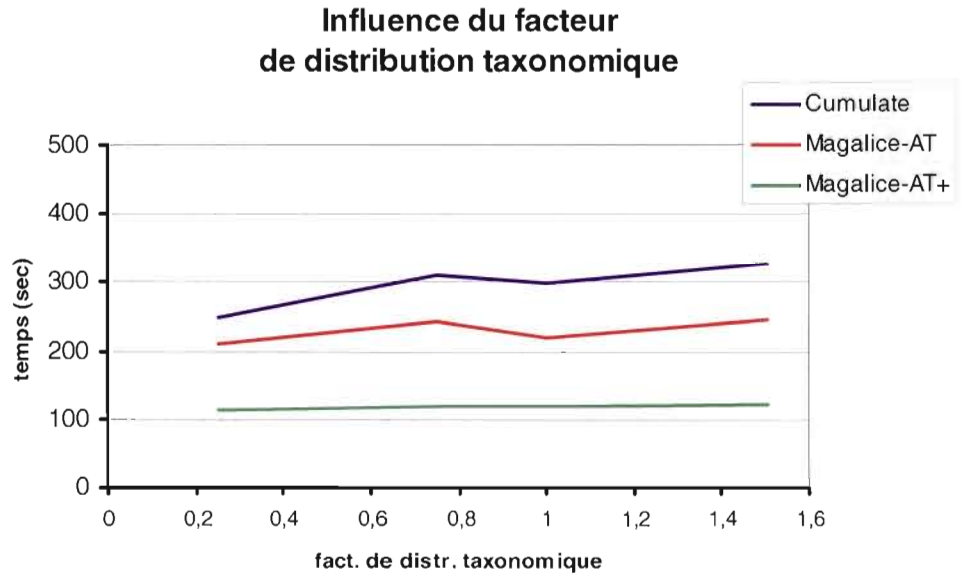

Figure 3.15 : Influence du facteur de distribution taxonomique sur le temps d'exécution des algorithmes.

3.2.5.2 Analyse des résultats

Dans le cas de l'implémentation de l'algorithme Cumulate, la plus grande proportion d'items fréquents dans les nœuds intermédiaires a pour conséquence qu'un plus grand nombre d'items infréquents seront des feuilles. Puisque le nombre de nœuds de niveau i est généralement (et c'est le cas pour notre jeu d'essai) beaucoup plus grand que le nombre de noeuds de niveau i-1, le nombre d'itemsets candidats produits à partir des itemsets fréquents sera moins nombreux dans le cas où le facteur de distribution taxonomique sera plus faible.

Dans le cas des algorithmes Magalice-AT et Magalice-AT+, nous n'observons qu'une légère augmentation des temps d'exécution en fonction des valeurs de ce paramètre, ce qui est normal puisque, nous l'avons vu plus haut, le nombre d'attributs fréquents augmente légèrement en fonction du facteur de distribution taxonomique. Le temps d'exécution généralement plus faible observé pour l'algorithme Magalice-AT+ est expliquable par le

plus faible nombre d'attributs utilisés dans la construction du treillis, du fait de l'élagage préalable des 1-itemsets infréquents.

Finalement, nous constatons que l'effet des variations de ce paramètre sur les temps d'exécution est plus faible que ce que nous avons observé pour d'autres paramètres, puisque, contrairement aux autres paramètres étudiés, celui-ci ne détermine qu'une tendance dans la distribution des attributs fréquents.

3.2.6 Hauteur de l'arborescence taxonomique

Une augmentation de la hauteur de l'arborescence taxonomique entraîne une augmentation du nombre moyen d'ancêtres taxonomiques par attribut. En conséquence, plus le nombre de niveaux sera grand, plus l'augmentation taxonomique moyenne des transactions sera importante. De plus, comme les items fréquents ont tendance à se retrouver parmi les noeuds supérieurs de l'arborescence taxonomique, une augmentation de la hauteur de cette arborescence aura tendance à augmenter le nombre d'itemsets fréquents.

3.2.6.1 Résultats

Le jeu de données utilisé emploie les paramètres par défaut modifiés comme suit : 100 objets, 100 attributs, 10% des attributs sont des racines, la taille maximale des itemsets fréquents de 4 et le support minimal est de 5%.

Cette diminution du nombre d'objets est dûe au fait que l'augmentation du nombre de niveaux taxonomiques a augmenté considérablement les besoins en mémoire des implémentations des algorithmes Magalice-AT et –AT+ sur la plate-forme Galicia, au point d'en empêcher l'exécution. Seule une réduction considérable du nombre d'objets du jeu d'essais a permis de remédier à ce problème.

L'augmentation du support minimal a été nécessaire pour garder les temps d'exécution de notre implémentation de l'algorithme Cumulate à l'intérieur des limites raisonnables.

Lors de la génération des jeux d'essais, le nombre de niveaux de la taxonomie a été spécifié au moyen du paramètre *nlevels* au lieu d'être calculé par le générateur.

Le résultat de l'exécution de l'implémentation des trois algorithmes sur ce jeu de données est présenté dans la figure suivante.

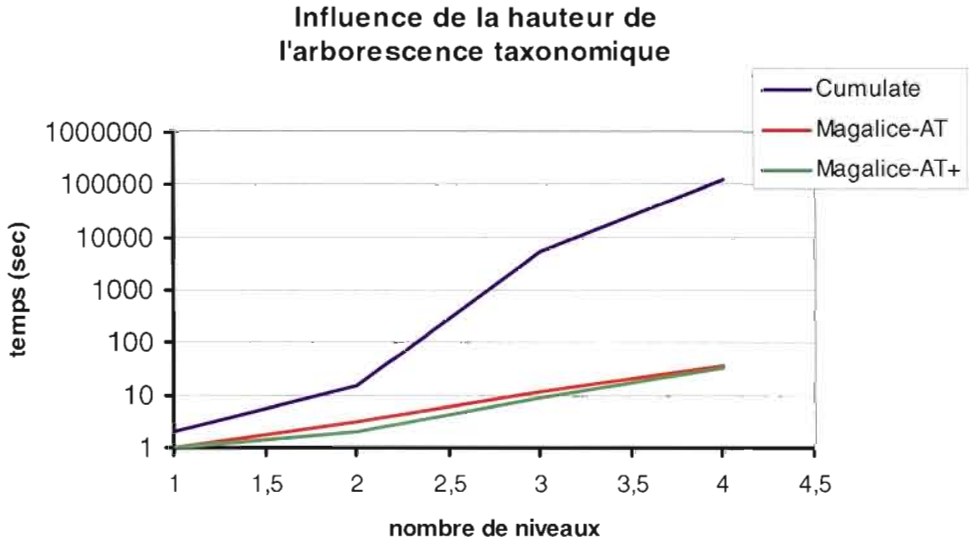

Figure 3.16 : Influence de la hauteur de l'arborescence taxonomique sur le temps d'exécution des algorithmes.

3.2.6.2 Analyse des résultats

Comme nous l'avons vu plus haut, les items fréquents ont tendance à se retrouver dans les niveaux taxonomiques supérieurs. La multiplication des niveaux taxonomiques entraîne à la fois la multiplication des items fréquents et, conséquemment, des candidats, ce qui amène une augmentation du temps de traitement dans le cas de l'algorithme Cumulate. Vu le processus de génération de candidats de cet algorithme, l'augmentation du nombre de

candidats est exponentielle dans le cas d'une augmentation linéaire du nombre de niveaux de l'arborescence taxonomique.

Comme les algorithmes Magalice-AT et -AT+ n'utilisent pas la génération de candidats, l'augmentation de leur temps de traitement est dûe à la simple augmentation du nombre d'items fréquents, laquelle entraîne l'augmentation du nombre de concepts fréquents, donc une augmentation du temps de traitement. Le comportement similaire des algorithmes Magalice-AT et -AT+ est explicable par le fait que les items infréquents ne sont évidemment pas affectés par l'augmentation du nombre de niveaux dans la taxonomie.

CHAPITRE IV

CONCLUSION

La fouille de données, aussi appelée prospection de données, est définie comme le traitement d'une grande quantité d'informations afin d'en extraire des connaissances non triviales et utiles. Cette analyse permet de dégager de la masse d'informations des tendances, des regroupements de données et de formuler des hypothèses. Une difficulté réside dans le fait que, pour qu'une association entre un groupe d'items soit détectable, les items doivent être retrouvés dans un ensemble de transactions en nombre suffisant, ce qui n'est trop souvent pas le cas. L'exploitation des liens de spécialisation/généralisation entre les attributs (exprimés par une arborescence taxonomique) permet la mise en évidence de règles d'association qui resteraient autrement ignorées.

Les algorithmes de recherche de règles d'association multi-niveaux décrits à ce jour sont basés sur l'approche par niveaux ou sur l'utilisation de structures de données appropriées et génèrent un nombre important de règles redondantes. Nous avons proposé dans ce travail deux adaptations multi-niveaux d'un algorithme de recherche de règles d'association basé sur l'analyse formelle de concepts et qui restreint la génération de règles aux seules règles informatives maximales. Cette approche est prometteuse pour améliorer la recherche de règles d'association multi-niveaux car les algorithmes basés sur l'analyse formelle de concepts sont généralement beaucoup plus performants que les algorithmes traditionnels, tant par leur temps d'exécution que par la qualité de l'information produite.

Nous avons implémenté nos algorithmes dans le cadre d'une plate-forme de construction de treillis, alors que les pré-traitements l'ont été dans des applications autonomes. Afin d'obtenir une base de comparaison, nous avons également implémenté un algorithme de recherche de règles d'association multi-niveaux déjà utilisé comme comparatif dans d'autres études. Ces implémentations nous ont permis de constater expérimentalement que les

algorithmes basés sur l'analyse formelle de concepts sont généralement beaucoup plus performants que l'algorithme de référence.

Notre travail a démontré le potentiel que représentent les algorithmes de règles d'associations basés sur l'analyse formelle de concepts dans le cadre de la génération de bases de règles multi-niveaux. Plusieurs approches prometteuses d'optimisation et d'exploitation de nos résultats restent cependant ouvertes dans ce domaine.

BIBLIOGRAPHIE

Agrawal, R. , et R. Srikant. 1994. «Fast Algorithms for Mining Association Rules». In *20th Int'l Conference on Very Large Databases*.

Ben Yahia, S., T. Hamrouni et E. Mephu Nguifo. 2006. «Frequent closed itemset based algorithms: a thorough structural and analytical survey». *SIGKDD Explorations*, vol. 8, no 1, p. 93-104.

Berardi, M., M. Lapi, P. Leo et Loglisci C. 2005. «Mining generalized association rules on biomedical literature ». In *18th international conference on Innovations in Applied Artificial Intelligence*.

Brin, S., R. Motwani, J.D. Ullman et S. Tsur. 1997. «Dynamic itemset counting and implication rules for maket basket data.». In *SIGMOD* 1997).

Burdick, D., M. Calimlim et J. Gehrke. 2001. «MAFIA: A maximal frequent itemset algorithm for transactional databases». In *17th International Conference on Data Engineering*: IEEE Computer Society.

Cortes, C., et V. Vapnik. 1995. «Support-Vector Networks». *Machine Learning*, vol. 20, p. 273-297.

Godin, R., et R. Missaoui. 1994. «An incremental concept formation approach for learning from databases». *Theoretical computer science*, vol. 133, no 2, p. 387-419.

Godin, R. 2006. *Systèmes de gestion de bases de données par l'exemple*, 2e édition. Longueuil, QC, Canada: Loze-Dion.

Gunopulos, D., R. Khardon, H. Mannila, S. Saluja, H. Toivonen et S.S. Ram. 2003. «Discovering all most specific sentences». *ACM Transactions on Database Systems*, vol. 28, no 2, p. 140-174.

Güvenir, S. , Özel Ayse et H. Altay. 2001. «An Algorithm for Mining Association Rules Using Perfect Hashing and Database Pruning». In *TAINN'2001*.

Han, J., J. Pei et Y. Yin. 2000. «Mining frequent apattern without candidate generation». In *ACM SIGMOD Conference on Management of Data* 2000).

Han, J., et M. Kamber. 2001. *Data mining: concepts and techniques*. San Francisco, CA: Morgan Kaufmann Publishers, 550 p.

Hipp, J., A. Myka, R. Wirth et U. Güntzer. 1998. «A new algorithm for faster mining of generalized association rules». In *PKDD '98*: Springer.

Jiang, N., et L. Gruenwald. 2006. «Research issues in data stream association rule mining». *SIGMOD Record*, vol. 35, no 1, p. 14-19.

Jiang, T., et A-H. Tan. 2006. «Mining RDF Metadata for Generalized Association Rules: Knowledge Discovery in the Semantic Web Era». In *WWW 2006*.

Liu, B., W. Hsu et Y. Ma. 1999. «Mining association rules with multiple minimum supports.». In *Int. Conf. Knowledge Discovery and Data Mining*.

Luxenberger, M. 1991. «Implications partielles dans un contexte.». *Mathématiques et Sciences Humaines*, vol. 29, no 113, p. 35-55.

Nehmé, K., P. Valtchev, M. H. Rouane et R. Godin. 2005. «On computing the minimal generator family for concept lattices and icebergs». In *ICFCA 2005*.

Pasquier, N., Y. Bastide, R. Taouil et L. Lakhal. 1999. «Efficient mining of association rules using closed itemsets lattices». *Information Systems*, vol. 24, no 1, p. 25-46.

Pasquier, N. 2000. «Extraction de bases pour les règles d'association à partir des itemsets fermés fréquents». In *INFORSID* 2000).

Pei, J., Han J. et R. Mao. 2000. «CLOSET: An efficient algorithm for mining frequent closed itemsets». In *ACM SIGMOD Worksop on Research Issues in Data Mining and Knowledge Discovery*: ACM.

Pramudiono, I., et M. Kitsuregawa. 2004. «FP-tax: Tree structure based generalized association rule mining». In *DKMD*.

Rouane, M. H., K Nehme, P. Valtchev et R. Godin. 2004. «On-line maintenance of iceberg concept lattices». In *ICCS 2004*.

Rouane, M. H. 2006. «Étude de l'analyse formelle dans les données relationnelles - Application à la restructuration des modèles structuraux UML». Thèse de doctorat, Montréal, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal.

Savasere, A., E. Omiecinski et S. Navathe.1995.«An efficient algorithm for mining association rules in large databases». Rapport technique no., Atlanta, GA, College of computing, Georgia Institute of Technology, 24 p.

Shenoy, P., J.R. Haritsa et S. Sudarshan. 2000. «Turbo-charging vertical mining of large databases». In *SIGMOD 2000*: ACM.

Srikant, R., et R. Agrawal. 1995. «Mining generalized association rules». In *21st Int'l Conference on Very Large Databases.*

Stumme, G., R. Taouil, Y. Bastide, N. Pasquier et L. Lakhal. 2002. «Computing iceberg concept lattices with TITANIC». *Data & Knowledge Engineering*, vol. 42, no 2, p. 189-222.

Taouil, R., N. Pasquier, Y. Bastide et L. Lakhal. 2000. «Mining bases for association rules using closed sets». In *ICDE.*

Tseng, M., et W. Lin. 2001. «Mining Generalized Association Rules with Multiple Minimum Supports.». In *Third International Conference on Data Mining and Knowledge Discovery.*

Valtchev, P., R. Missaoui, R. Godin et M. Meridji. 2002. «Generating Frequent Itemsets Incrementally: Two Novel Approaches Based On Galois Lattice Theory». *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 14, no 2-3, p. 115-142.

Zaki, M.J., S. Parthasarathy, M. Ogihara et W. Li.1997.«New algorithms for fast discovery of association rules». Rapport technique no. 651, The University of Rochester, Computer Science Dept., Rochester, NY 24 p.

Zaki, M.J., et C. Hsiao. 2002. «ChARM: An efficient algorithm for closed itemset mining». In *2nd SIAM International Conference on Data Mining.*