CIGA+ : un algorithme de calcul d'un ensemble concis de motifs fermés fréquents

Rokia Missaoui

Département d'informatique et d'ingénierie, Université du Québec en Outaouais, C.P. 1250, succursale B, Gatineau, Québec, Canada, J8X 3X7

rokia.missaoui@uqo.ca

Ganaël Jatteau

Département d'informatique et d'ingénierie, Université du Québec en Outaouais, C.P. 1250, succursale B, Gatineau, Québec, Canada, J8X 3X7

jatg01@uqo.ca

Sachant que le résultat d'un algorithme de fouille de données *data mining* peut être très grand même pour des ensembles réduits de données, l'objectif de cet article est de proposer une approche qui permet de réduire ce résultat et donc le temps de calcul en approximant l'ensemble des motifs fermés fréquents (MFF). Plus précisément, nous proposons CIGA+ *Closed Itemset Generation and Approximation*, un algorithme qui construit et exploite un graphe de dépendances pour en extraire un ensemble de MFF dont le degré d'approximation (éventuellement nul) dépend de la valeur affectée a deux paramètres d'entrée : la fréquence de co-occurrences de deux items individuels et la tolérance. Les expérimentations montrent le potentiel de notre approche pour générer un ensemble pertinent de MFF. De plus, la comparaison de CIGA+ avec un autre algorithme de génération d'un ensemble approximatif de MFF montre la capacité de CIGA+ a extraire rapidement un ensemble de MFF même a partir de bases de données volumineuses et denses.

Fouille de données, règles d'association, approximation, motifs fermés fréquents.

Since the output of a data mining task can be very large even for a reasonably small data set, the objective of the present paper is to describe an approach which reduces the data mining output and hence the execution time by approximating the set of frequent closed itemsets. More precisely, an algorithm called CIGA+ (Closed Itemset Generation and Approximation) is proposed and aims at partial or complete generation of frequent closed itemsets (*FCIs*) based on the construction and exploration of a dependency graph. The degree of approximation (eventually null) depends upon the value assigned to two parameter thresholds : cooccurrence frequency between two individual items and tolerance. Experimental analysis of our approach illustrates its cost-effectiveness and its potential for efficient association rule mining. Moreover, a comparative study with an existing and efficient algorithm for mining *FCIs* shows that CIGA+ has good performances even for large and dense data sets.

Association rule mining, frequent closed itemsets, approximation.

1 Introduction

L'extraction des règles d'association est un important thème de recherche qui a attiré l'attention et l'intérêt de nombreux chercheurs en fouille de données. Dans plusieurs travaux inspirés de l'algorithme Apriori (Agrawal et Strikant, 1994), la génération des règles se fait en deux étapes : la détection des motifs fermés fréquents, puis la génération des règles d'association ayant une confiance $\geq$ *minconf* (confiance minimale tolérée). La deuxième étape est relativement simple alors que le première présente de grands défis et peut être très coûteuse puisque le nombre de motifs obtenus peut être exponentiel par rapport au nombre total d'items. La recherche de motifs fermés fréquents (MFF) est reconnue comme étant une solution satisfaisante pour réduire le nombre de motifs fermés et donc de règles d'association générées a partir de ces motifs (Pasquier, Bastide et al., 1999, Pei, Han et al., 2000,Wang, Han et al., 2003, Zaki, 2000, Zaki et Hsiao, 2002). Toutefois, le nombre de MFF reste parfois très élevé même pour des bases de données de petite taille.
Dans cette étude, nous cherchons a améliorer le processus d'extraction des motifs fermés fréquents en rendant l'approche plus paramétrable que les méthodes classiques et en offrant des possibilités d'approximation du résultat. L'idée est d'utiliser un graphe de dépendances comme support a la génération des MFF ainsi que des techniques d'élagage fondées sur trois seuils : le support, la co-occurrence et la tolérance. Les deux derniers paramètres peuvent être choisis de manière a obtenir soit la totalité soit une partie des MFF. Dans le deuxième cas, le nombre de MFF peut être largement réduit sachant que le plus souvent, les MFF les moins pertinents se trouvent écartés. Notre but est de limiter la prospection des MFF aux candidats qui auront le plus de chance d'apparaître dans de grands motifs tout en éliminant les couples d'items faiblement liés entre eux.
L'article est organisé comme suit : la section 2 fournit un rappel sur les concepts et un brève étude de la littérature sur l'extraction des règles d'association. Notre solution est décrite dans les sections 3 et 4 où sont décrits la construction du graphe de dépendances et l'exploration de ce même graphe pour générer un ensemble de MFF. Des résultats empiriques fondés sur la comparaison de CIGA+ avec BAMBOO (Wang, Han et al., 2003) sont présentés dans la section 5. Enfin, une conclusion est fournie en section 6.

2 Survol de la littérature

La génération des règles d'association a fait l'objet d'un nombre considérable de travaux de recherche dans le domaine de la fouille de données. Dans cette section, nous rappelons d'abord quelques notions élémentaires, puis nous présentons une revue de la littérature sur la production des règles d'association.

2.1 Notions élémentaires

Soit $I=\{i_1,i_2,\ldots,i_m\}$ un ensemble de m items distincts. Une transaction T contient un ensemble d'items contenus dans I, et possède un identifiant unique *TID*. Un sous-ensemble Y de I où $k=|Y|$ est un motif de taille k (*k-itemset*). Une base de transactions $\mathcal{D}$ est l'ensemble de transactions effectuées sur un sous-ensemble d'items de I. $X \subset \mathcal{D}$ est un ensemble de transactions (appelé *tidset*) tandis que la fraction de transactions dans $\mathcal{D}$ contenant Y est appelée support de Y et notée $supp(Y)$. Ainsi, on dit qu'un motif est fréquent lorsque son support dépasse un support donné appelé *minsupp*. A un *tidset* X donné correspond un motif dont la valeur est $f(X)$. Cela correspond à l'ensemble des items communs a toutes les transactions de X. De la même manière, $g(Y)$ représente l'ensemble des transactions qui contiennent tous les items de l'ensemble Y.

Un motif *Y* est dit fermé lorsqu'il n'existe aucun motif *Z* de taille supérieure a *Y* mais avec le même support que *Y*. En d'autres termes, tout motif possède le même support que sa fermeture. De fa3on similaire, un *tidset* est fermé lorsque aucun autre *tidset* de taille supérieure possède le même support. Une autre fa3on d'écrire la condition de fermeture revient a dire que *Y* (resp. *X*) est fermé si et seulement si $Y=f(g(Y))$ (resp. $X=g(f(X))$).
Depuis l'apparition de l'algorithme Apriori, plusieurs algorithmes d'extraction des règles d'association ont vu le jour. Pour la plupart d'entre eux, le but est d'améliorer l'efficacité de la méthode initiale (Hipp, Guntzer et al., 2000) tandis que le principal problème reste le vaste ensemble de MFF qui est $Y_1 \cap Y_2 = \phi$ difficilement manipulable a l'étape subséquente de production des règles. Pour réduire la taille de cet ensemble, plusieurs études ont déja été conduites sur les motifs fermés (Zaki et Hsiao, 2002, Pasquier, Bastide et al., 1999, Bastide, Taouil et al., 2000, Pei, Han et al., 2000) et les motifs maximaux (MFM) [1] (Bayard, 1998, Burdick, Calimlim et al., 2001). Bien qu'il puisse y avoir un nombre exponentiellement plus grand de MFF que de MFM, les premiers ont l'avantage de n'engendrer aucune perte d'information.
Une règle d'association *r* est une implication de la forme $Y_1 \Rightarrow \square Y_2$, où et sont des sous-ensembles de *I*, et Le support de la règle *r* est égal à $\mathrm{supp}(Y_1 \cup Y_2)$ alors que la confiance représente le rapport $\mathrm{supp}(Y_1 \cup Y_2) / \mathrm{supp}(Y_1)$
Avec Apriori (Agrawal et Strikant, 1994), le calcul des règles d'associations à partir de l'ensemble des MFF se fait de la façon suivante. Pour chaque motif fréquent *Y*, on génère les divers sous-ensembles non vides de *Y*. Ensuite, pour chaque sous-ensemble de *Y*, une règle de la forme $Y_1 \Rightarrow Y - Y_2$ est retenue si le rapport $\mathrm{supp}(Y) / \mathrm{supp}(Y_1)$ est au moins égal à *minconf*. Dans le cadre des MFF, certaines études se sont intéressées à la génération des ensembles non redondants de règles (Lakhal, Pasquier et al.,1999, Luxemburger, 1991, Valtchev, Missaoui et al., 2003) où est un générateur, c.-a-.d., un sous-ensemble minimal de *Y* tel que sa fermeture est égale à *Y* (Pfaltz et Taylor, 2002).

2.2 Calcul des motifs fermés fréquents

ACLOSE, CLOSE (Pasquier, Bastide et al., 1999), CHARM (Zaki et Hsiao, 2002) et CLOSET+ (Wang, Han et al., 2003) font partie des premiers algorithmes de calcul de MFF. TITANIC (Stumme, Taouil et al., 2002) hérite de l'approche fondée sur la notion de générateur présente dans ACLOSE, mais propose des simplifications intéressantes. CLOSET et son amélioration récente CLOSET+ génèrent tous les deux les MFF comme les branches maximales d'un arbre appelé *FP-tree*, une structure basée sur un arbre préfixe (ou arbre *trie*) augmentée par une liste transversale de pointeurs. BAMBOO (Wang et Karypis, 2004) est une version améliorée de CLOSET+ dans la mesure où il produit un résultat plus concis et propose des performances plus intéressantes. BAMBOO exploite la contrainte de support de longueur décroissante en plus d'un certain nombre d'élagages et d'optimisations pour améliorer l'efficacité du calcul des MFF. La contrainte de support décroissant consiste a fournir un support sous forme d'une fonction décroissante qui varie selon la taille du motif de manière a réduire le nombre de motifs de petite taille. Les expérimentations conduites dans (Wang et Karypis, 2004) montrent que BAMBOO est généralement plus performant que trois algorithmes connus de génération des MFF pour les divers types de bases de données. Toutefois, la fonction de support décroissant n'est pas établie par les auteurs d'une manière théorique mais davantage par des expérimentations et pour une base de données très spécifique.

[1] Les motifs fréquents maximaux sont les ensembles dont n'importe quel surensemble est non fréquent.

Notre algorithme, appelé CIGA+, est similaire a *BAMBOO* dans la mesure où il génère un sous-ensemble concis de MFF et propose des temps d'exécution intéressants. A la différence de BAMBOO qui fonde son élagage sur la fonction de support décroissant, CIGA+ utilise les co-occurrences d'items pour déterminer rapidement les motifs de grande taille et s'inspire des propriétés des treillis de concepts en analyse formelle de concepts (Ganter et Wille, 1999).

2.3 Définition du problème

Dans ce qui suit, nous décrivons CIGA+, un algorithme qui utilise trois paramètres de seuil pour produire un ensemble complet ou approximatif des motifs fermés fréquents (MFF) en explorant un graphe de dépendances dans lequel les noeuds sont des items et les arêtes représentent des liens (fréquences de cooccurrence) entre deux items. Les seuils utilisés sont : le support minimal (*minsupp*), la confiance minimale entre deux items (*mincooc*), et la tolérance minimale (*mintol*). Le premier paramètre est déja connu par la communauté du *data mining*, le second paramètre *mincooc* permet d'écarter des liens faibles entre deux items individuels en se basant sur leur fréquence de co-occurrence, et le dernier paramètre permet d'éliminer d'éventuels liens qui ne sont pas nécessaires entre deux items. Par conséquent, le graphe de dépendances est simplifié pour en faciliter son parcours et la production d'un ensemble plus réduit de MFF. Le paramétrage du seuil *mintol* dépend du degré d'approximation souhaité par l'utilisateur. Un seuil de 100% permet d'obtenir l'ensemble complet des MFF tandis qu'une valeur non nulle génère une approximation plus ou moins importante de cet ensemble.
CIGA+ comporte deux étapes : la construction du graphe de dépendances (section 3) puis la génération des MFF (section 4). Les seuils de cooccurrence et de tolérance sont utilisés lors de la première étape tandis que le support est utilisé dans la seconde.
A titre d'exemple, nous allons considérer la petite base de transactions de la table (tab1). Elle comporte huit transactions et neuf items *{a,b,c,d,e,f,g,h,i}* qui décrivent des paniers de consommateurs (achat de produits).
Les items sont triés selon l'ordre décroissant de leur support car les items les plus fréquents apparaissent plus souvent dans le résultat et donc sont traités en premier. Le tri des items est aussi une condition nécessaire pour appliquer l'élagage décrit dans la section 3.

Tid	(a) Beurre	(b) Eau minérale	(c) Baguette	(d) Biscuits	(e) Foie gras	(f) Roquefort	(g) Saumon fumé	(h) Champagne	(i) Caviar
1	X	X					X		
2	X	X					X	X	
3	X	X	X				X	X	
4	X		X				X	X	X
5	X	X		X		X			
6	X	X	X	X		X			
7	X		X	X	X				
8	X		X	X		X			

TAB. 1: Base de transactions.

3 Prétraitement

Cette section décrit l'étape de prétraitement qui consiste à construire un graphe de dépendances a partir d'une matrice de cooccurrences. Le graphe servira ensuite à la génération des MFF. Le graphe peut être partiel ou complet (i.e tous les MFF sont générés) selon les valeurs qui sont attribuées à *mincooc* et *mintol*. La signification de ces paramètres est expliquée dans cette section.

3.1 Matrice de cooccurrence

La construction du graphe de dépendances débute par la construction d'une matrice de cooccurrences (voir table 2) pour les items fréquents. Il s'agit d'une matrice triangulaire dont les lignes et les colonnes sont ordonnées par ordre décroissant du support des items. La valeur d'une cellule $cooc[i,j]$ (avec $j \geq i$) représente le nombre de fois que l'item apparaît avec a_i et $cooc[i,i]$ représente le support (absolu) de l'item. Par exemple, $cooc[a,d]=4$.

	a	b	c	d	g	f	h	e	i
a	8	5	5	4	4	3	3	1	1
b		5	2	2	3	2	2	0	0
c			5	3	2	2	2	1	1
d				4	0	3	0	1	0
g					4	0	3	0	1
f						3	0	0	0
h							3	0	1
e								1	0
i									1

TAB. 2: Matrice de cooccurrence.

Un ensemble de mesures sur des règles élémentaires (c.a.d. règles qui mettent en jeu deux items individuels) peut être directement extrait de la matrice de cooccurrence :

- Le support relatif d'une règle est $a_i \Rightarrow a_j$. Le paramètre *minsupp* est sa valeur minimale tolérée.
- La confiance d'une règle est $a_i \Rightarrow a_j$. Sa plus petite valeur tolérée est appelée $\frac{cooc[i,j]}{cooc[i,i]}$ *mincooc*.
- La tolérance d'un lien (a_j,a_i),où $supp(a_j) <= supp(a_i)$ est équivalente a la confiance d'une règle. Sa plus petite valeur acceptée est *mintol*.

Par exemple, *conf*($f \Rightarrow d$)=100%, ce qui signifie que l'item *f* apparaît toujours en présence de l'item *d*.

3.2 Graphe de dépendances

On définit un graphe de dépendances *G*=<*N*,*T*> comme étant un graphe orienté et acyclique dont les noeuds dans *N* correspondent aux items fréquents. Il représente les dépendances qui ont lieu entre deux items selon leur fréquence de cooccurrence. Ainsi, une arête dans *T* allant du noeud a_i au noeud a_j, et notée (a_i,a_j), indique que *conf*($a_i \Rightarrow a_j$) >= mincooc et $supp(a_i) >= supp(a_j)$.

3.3 Élagage

Comme pour la plupart des algorithmes de type Apriori, un des élagages utilisés dans CIGA+ consiste a ignorer les items non fréquents (c.-a-.d., les items qui ont un support inférieur a *minsupp*). Les autres élagages permettent d'écarter des MFF que l'on juge non pertinents. Les paramètres sont exploités de la manière suivante: utilisation de *minsupp* pour écarter les items non fréquents, utilisation de *mincooc* pour écarter les liens faibles entre deux items donnés dans le graphe de dépendances, et utilisation de *mintol* pour éliminer les arêtes inutiles entre deux items donnés.

Propriété 1 Lorsque $g(a_j) \subseteq g(a_i)$ (c.-a-d. $conf(a_j \Rightarrow a_i) = 100\%$), alors l'arête (a_h,a_j) du graphe de dépendances est toujours inutile quelque soit tel que $supp(a_k) \geq supp(a_i)$.

La propriété 1 (figure 1) signifie que lorsqu'un item apparaît systématiquement dans une transaction avec a_i, alors l'arête (a_h,a_j) dans le graphe de dépendances est redondante pour tout noeud a_h qui précède a_i. Ceci est toujours vrai dans la mesure où tout MFF contenant a_j

va nécessairement inclure a_i, et par conséquent un chemin qui court-circuite a_i en allant de a_h à a_i conduit toujours vers un motif non fermé. Il est important de noter que a_i n'est pas $supp(a_k) \geq supp(a_i)$ nécessairement le parent direct de a_j, et que la présence de (a_h,a_j) dans le graphe de dépendances, par définition, implique .

Dans notre exemple, l'item *Roquefort* apparaît toujours avec *biscuits* puisque *conf(Roquefort* *biscuits) = 100%*. Ainsi, tout *MFF* qui contient *Roquefort* et tout autre item ayant un support supérieur ou égal au support de *biscuits* (e.g., *eau minérale*) va nécessairement contenir l'item *biscuits*. En d'autres termes, la fermeture de {*eau minérale, Roquefort*} contiendra toujours l'item *biscuits*. L'arête *(eau minérale, Roquefort)* est alors inutile.

La propriété 1 est adaptable pour approximer (simplifier) le graphe de dépendances. La propriété suivante en est une généralisation.

Propriété 2 *Si* $conf(a_j \Rightarrow a_i) \geq mintol$*, alors l'arête* ($a_h$,$a_j$) *est rarement utile pour tout noeud* a_h *tel que* $supp(a_k) \geq supp(a_i)$.

La propriété 2 permet d'écarter l'arête *(*a_h,a_j*)* chaque fois que l'item a_j apparaît avec a_i (en se basant sur la valeur de *mintol*) puisque tout motif qui contient a_j aura de fortes chances de contenir a_i.

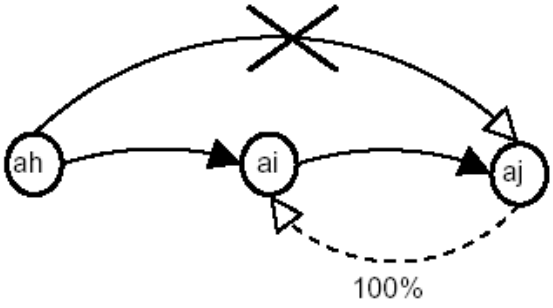

FIG. 1: Illustration de l'élagage appliqué au graphe avec *mintol* =100%.

Un cas particulier de l'élagage basé sur *mintol* intervient lorsque représente l'état initial. Dans un tel cas, cela implique que le noeud ne sera pas connecté au noeud initial dès que la condition $conf(a_j \Rightarrow a_i) \geq mintol$ aura lieu.

Le graphe de dépendances de la figure 2 a été généré en utilisant *mintol*=100 %. On peut remarquer qu'il n'y a pas systématiquement d'arêtes allant du noeud initial vers chaque noeud du graphe. Par exemple, on observe que *Roquefort* apparaît tout le temps en présence de l'item *biscuits* et par conséquent, une arête allant de {} vers *Roquefort* est non justifiée puisqu'il existe un chemin aboutissant a *Roquefort* en passant par *biscuits*, et que *Roquefort* ne peut pas constituer a lui seul un MFF. De même, les arêtes (*eau minérale, Roquefort*), (*beurre, Roquefort*) et (*baguette, Roquefort*) sont inappropriées.

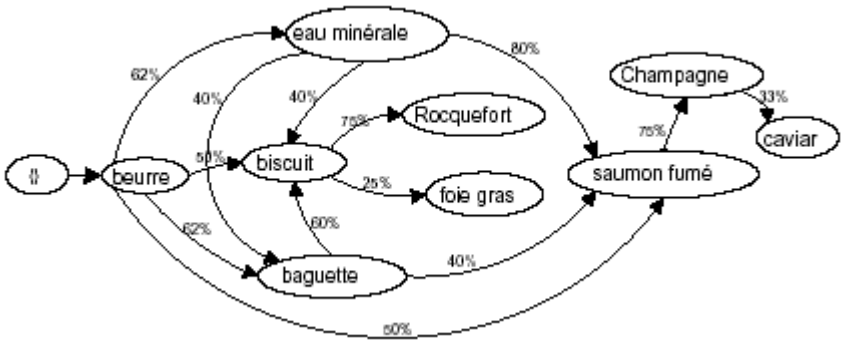

FIG. 2: Graphe de dépendances complet avec *mincooc*=0 et un seuil de tolérance=100%.

3.4 Algorithme

Dans cette sous-section, nous présentons l'algorithme qui construit le graphe de dépendances. L'algorithme 1 construit le graphe de dépendances à partir de la matrice de cooccurrences. Il possède trois paramètres en entrée : la matrice de cooccurrence *cooc*[*n*,*n*] où *n* est le nombre d'items fréquents, *mincooc* et *mintol*. *N* représente l'ensemble de noeuds dans le graphe de dépendances. {} est le noeud initial et *(*a_i,a_j*)* représente l'arête allant du noeud au noeud .

La construction du graphe de dépendances s'effectue de la fa3on suivante. Comme l'indiquent les deux boucles *pour*, les items sont traités par ordre décroissant de leur support, c.a.d. en considérant en premier les items les moins fréquents puis en intégrant progressivement ceux qui sont les plus fréquents. Deux items distincts et (identifiés respectivement par la ligne *i* et la colonne *j* de la matrice de cooccurrence) sont comparés pour déterminer si l'arête *(*a_i,a_j*)* doit être tracée. Sauf dans le cas où la propriété 1 (ou la propriété 2) est vérifiée, il existe une arête allant de a_i à a_j tant que la condition supp(a_i) >= supp(a_j)est vraie. La ligne 5 tient compte de la confiance (fréquence de cooccurrence) tandis que les lignes 8 et 9 exploitent le seuil de tolérance (figure 1). En d'autres termes, la ligne 5 vérifie si et sont connectés dans la base de transactions tandis que la ligne 8 applique l'élagage mentionné précédemment (sous-section 3.3). A la ligne 14, les points d'entrée sont générés. Par défaut, aucun ("item") noeud du graphe de dépendances n'est connecté au noeud initial. Dès qu'un item n'apparaît "presque jamais" en présence d'un autre, alors on choisit de le relier au noeud initial.

Algorithme 1: CONSTRUIRE_GRAPHE

Entrées : $cooc[n,n]$, $mincooc$, $mintol$
Sorties : ($G =< \mathcal{N}, \mathcal{T} >$)
Créer les noeuds N_1 jusqu'à N_n;
pour j *allant de* n *jusqu'à* 1 **faire**
  $placer\_lien$ = vrai;
  pour i *allant de* $j-1$ *jusqu'à* 1 **faire**
    si $\frac{cooc[i,j]}{cooc[i,i]} \geq mincooc$ **alors**
      $\mathcal{T} \leftarrow \mathcal{T} \cup (a_i, a_j)$;
    fin
    si $\frac{cooc[i,j]}{cooc[j,j]} \geq mintol$ **alors**
      $placer\_lien$ = faux;
      sortir;
    fin
  fin
  si $placer\_lien$ **alors**
    $\mathcal{T} \leftarrow \mathcal{T} \cup (\{\}, a_i)$;
  fin
fin

Comme mentionné ci-dessus, la confiance (exprimée comme la fréquence de cooccurrence entre deux items) est un moyen de réduire la taille du graphe de dépendances. Le graphe partiel (figure 3) est présenté (par opposition au graphe complet de la figure 2) où le seuil de confiance *mincooc* vaut 50 %. Par exemple, l'ensemble d'items {*beurre, eau minérale, baguette*} n'apparaîtra pas dans l'ensemble des MFF puisque le chemin correspondant dans le graphe n'existe pas du fait que *conf(eau minérale □ baguette)=40%*.

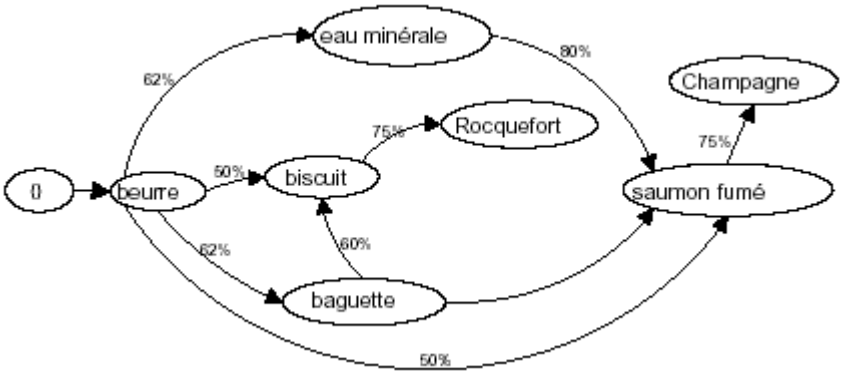

FIG. 3: Graphe de dépendances avec *mincooc*=50% et *mintol*=100%.

Preuve de complétude : Nous allons montrer que le graphe de dépendances *G* est une base complète pour générer tous les MFF lorsque aucune approximation n'est utilisée (c.-a-d. *mincooc*=0% et *mintol*=100%). Tout d'abord, nous supposons qu'aucun élagage n'est appliqué en se basant sur la propriété 2. Dans ce cas, lorsque a_i et a_j apparaissent dans une même transaction *T*, alors, il y a toujours une arête allant de vers avec supp(a_i) $\text{supp}(a_i) \geq \text{supp}(a_j)$. Si maintenant on tient compte de l'élagage décrit par la propriété 1, on cherche alors a montrer que l'arête (a_h,a_j) écartée ne contribue pas à la construction d'un MFF noté *Z* et représentant le chemin allant de l'état initial au noeud a_j. Ainsi, lorsque $g(a_j) \subseteq g(a_i)$, alors a_i n'aura qu'un seul prédécesseur a_i et le graphe *G* ne conduira pas a un MFF puisque $Z \supset \{a_h \cup a_j\}$ et $Z \cap a_i = \phi$. Ayant les propriétés $g(a_j) \subseteq g(a_i)$ et $g(Z) \subseteq g(a_j)$, alors $g(Z) \subseteq g(a_i)$ ce qui signifie que supp(Z) = supp($Z \cup a_i$) et donc *Z* n'est pas un MFF.

4 Génération des MFF

L'idée derrière CIGA+ est la suivante. Le support de a plus de chances d'être élevé que le support de si l'item le moins fréquent dans *Y* est tel que $conf(a_i \Rightarrow a_j) > conf(a_i \Rightarrow a_k)$. Cela permet de se diriger rapidement vers les motifs de plus grande taille.
Par exemple, considérons le motif *Y*={*beurre, baguette*} (voir figure 2) où *baguette* est l'item dans *Y* qui possède le plus petit support. L'extension de *Y* est *g(Y)={3,4,6,7,8}*. Donc, le candidat le plus prometteur pour augmenter *Y* est *biscuits* plutôt que *saumon fumé* puisque *conf(baguette* □ *biscuits)=3/5* tandis que *conf(baguette* □ *saumon fumé)=2/5*. Ainsi, *g*(*Y*∪*biscuits*)={6,7,8} et *g*(*Y* ∪ saumon fumé)={3,4}.
Dans cette section, nous décrivons tout d'abord l'algorithme CIGA+ en indiquant comment les *tidsets* fermés et les motifs sont calculés puis nous illustrons l'exécution de CIGA+ par un exemple.

4.1 Algorithme

L'algorithme CIGA+ (voir l'algorithme 2) possède quatre paramètres : la matrice de cooccurrences *cooc*[*n,n*], *minsupp*, *mincooc* et *mintol*. La variable globale *L* est utilisée dans la procédure CIGA+_SUB pour stocker les *MFF* calculés. CIGA+ procède en deux étapes. Tout d'abord, le prétraitement des données basé sur la construction du graphe de dépendances est effectué. Ensuite, un appel de la procédure récursive que nous appelons CIGA+_SUB est effectué pour chaque noeud directement lié au noeud initial. Tant que le support du chemin courant (c.-a-d. l'ensemble d'items situés sur le chemin) est plus grand que *minsupp* ou bien

qu'un noeud terminal n'est pas atteint, l'algorithme 3 continue à parcourir le graphe de dépendances en explorant les successeurs du noeud courant.

Algorithme 2: CIGA+

Entrées : $cooc[n,n]$,$minsup$, $mincooc$, $mintol$
Sorties : L : Ensemble des *MFF*
$\mathcal{G} \leftarrow$ CONSTRUIRE_GRAPHE ($cooc[n,n]$,$mincooc$,$mintol$);
global $L \leftarrow \emptyset$;
pour chaque *noeud a dans T({},a)* **faire**
  CIGA+_SUB(a, g(a),$minsup$);
fin

Dans l'algorithme 3, la variable *a* représente l'item associé au dernier noeud du chemin courant. *X* est le *tidset* associé au motif constitué du chemin courant dans le graphe. Le traitement dans l'algorithme 3 inclut : le calcul de *X=g(Y)* et du support de *Y* (c.-a-.d. la taille de *X*), une exploration récursive du graphe de dépendances, et (le calcul des *MFF* (voir ligne 10). La première étape est la plus coûteuse. C'est pourquoi, il est important d'utiliser une structure de données adaptée pour stocker et retrouver les *g*(*a*) ainsi qu'une manière efficace de calculer les intersections entre *g*(*a*) et *X*. Nous avons utilisé un index pour les items individuels afin d'obtenir rapidement la valeur du *tidset* résultant.
Le calcul de *f*(*X*) intervient lorsque le support du motif courant est inférieur a *minsupp* ou bien lorsqu'on atteint un noeud terminal (voir ligne 3 de l'algorithme 3). Chaque fois que l'on identifie une égalité entre le support de *Y* et $Y \cup a$ au cours du parcours, alors le calcul de la fermeture *g*(*Y*) de *Y* n'est pas effectué puisque *Y* n'est pas fermé. Le calcul a la ligne 10 intervient lorsque aucun successeur du noeud courant *a* ne conserve le même support que celui du motif *Y=f*(*X*) qui correspond au chemin se terminant par *a*. Ce type de calcul intervient plus ou moins souvent selon les seuils retenus pendant la construction du graphe de dépendances.

Algorithme 3: CIGA+_SUB

Entrées : a, X, $minsupp$
$X \leftarrow g(a) \cap X$;
$candidat \leftarrow vrai$;
si $taille(X) > (minsup \times |\mathcal{D}|)$ *et* $taille(X) > 1$ **alors**
  pour chaque *succ de a* **faire**
    si *CIGA+_SUB(succ,X,minsupp)*=$taille(X)$ **alors**
      $candidat = faux$
    fin
  fin
  si $candidat$ **alors**
    $L \leftarrow L \cup f(X)$;
  fin
fin
retourner *taille(X)*

4.2 Calcul des *tidsets* et motifs fermés

La génération des MFF dans CIGA+ consiste a explorer le graphe de dépendances de faɔon récursive. Un chemin allant du noeud initial jusqu'au noeud courant représente la séquence d'items qui correspond au motif *Y*. Pour chaque chemin dans le graphe, le *tidset* correspondant [2] associé a la séquence d'items courante est calculé.

[2]On rappelle que *X=g*(*Y*) et *Y=f*(*X*).

Propriété 3 Soit X un tidset, Y un motif et $a_i \notin$ Y. Alors, X =g(Y $\cup a_i$) est équivalent à $g(Y) \cap g(a_i)$, et X est un tidset fermé.

La propriété 3 permet de mettre à jour X en se basant sur la valeur de g(Y) pour y inclure un nouvel item à partir d'une simple intersection. Cette propriété est utile pour générer rapidement le tidset associé au chemin courant dans le graphe de dépendances. De plus, nous savons que X est un tidset fermé puisqu'il est construit à partir d'une intersection entre deux tidsets fermés (Ganter et Wille, 1999). Par exemple, g(beurre, cracker) = {5,6,7,8}. Lorsque le noeud Roquefort est atteint (figure 3), le calcul de g(beurre, cracker, Roquefort) est effectué en faisant l'intersection des deux tidsets : {5,6,7,8} et {5,6,8}.

L'algorithme 3 parcourt le graphe en utilisant la propriété 3 pour générer les tidsets fermés X. Une fois que le tidset X relatif au chemin courant a été calculé et que sa taille est bien supérieure ou égale au support absolu, le motif correspondant Y=g(X) peut être calculé. Cependant, un tel calcul risque d'être coûteux et parfois inutile. C'est la raison pour laquelle nous évitons de le faire systématiquement. Comme indiqué auparavant, un chemin dans le graphe de dépendances est exploré progressivement jusqu'à ce que le support de son motif associé devienne inférieur à minsupp ou bien qu'un noeud terminal soit atteint. Pendant ce parcours, à chaque fois que l'ajout d'un noeud au chemin courant ne réduit pas le support, on en déduit que le motif courant n'est pas fermé et on évite de calculer sa fermeture (ligne 5). Lorsque la condition de la ligne 3 n'a pas lieu, alors le motif devient un candidat intéressant et sa fermeture est calculée à la ligne 10.

4.3 Coût et structures de données

Le traitement complet de la génération des MFF comprend trois étapes principales: la transformation des données, (la construction du graphe de dépendances et l'extraction des MFF. La transformation des données et la construction du graphe font partie du prétraitement car leur coût est négligeable par rapport au calcul des MFF. La complexité de l'algorithme de construction du graphe de dépendances (algorithme 1) est $O(m^2)$ où *m* représente le nombre d'items. La transformation des données, quant a elle, permet de réarranger la base pour obtenir de meilleures performances lors du calcul des MFF. L'opération la plus commune pendant ce calcul est l'intersection entre deux ensembles de transactions $g(I)$ et $g(a_i)$ où a_i est un item. Aussi, la base de données est réarrangée pour stocker tous les $g(a_i)$ sous forme binaire.

4.4 Exécution de l'algorithme

La figure 4 illustre l'exécution de l'algorithme CIGA+ a partir des données de la table 1 et du graphe de dépendances de la figure 2. Les paramètres *minsupp*, *mincooc*, et *mintol* ont comme valeur 0%, 0% et 100% respectivement de sorte que l'algorithme génère l'ensemble complet des MFF.

Lorsque l'algorithme commence l'exploration du graphe de dépendances, chaque descendant direct du noeud initial est tout d'abord pris en compte (voir ligne 4 de l'algorithme 3). Dans la figure 4, on voit que seul le noeud *a* est directement accessible depuis la racine. $X=g(a)$ est alors égal a {1,2,...,8}. Une fois que le noeud *a* est traité, ses successeurs *b*, *c*, et *d* sont alors pris en considération. Supposons que le chemin [a,c,g,h,i] est suivi. L'exploration du noeud *h* mène au calcul de $X=X_{a,c,g} \cap g(h) = \{3,4\}$.

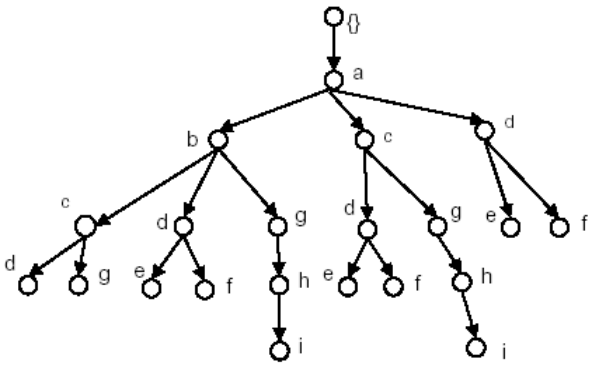

FIG. 4: Simulation d'exécution de CIGA+.

Lorsque le noeud *i* est atteint, *X* devient égal a {4}. Comme *i* est un noeud terminal, on calcule *f(X)=f({4})={a,c,g,h,i}* et ensuite on obtient un MFF. Sachant que le support reste constant en passant de la séquence *[a,c,g,h]* à *[a,c,g]*, il n'est pas nécessaire d'effectuer un calcul de fermeture au niveau du noeud *g*.
La table 3 représente l'ensemble complet des motifs fermés *MF* générés a partir de la table 1.

MF	support	*MF*	support
abcdefghi	0	abg	3/8
acghi	1/8	abc	3/8
abcgh	1/8	agh	3/8
abcdf	1/8	adf	3/8
acde	1/8	ag	4/8
abgh	2/8	ac	5/8
acdf	2/8	ab	5/8
acgh	2/8	ad	4/8
abdf	2/8	a	1

TAB. 3: Liste des motifs fermés

5 Étude expérimentale

Il est difficile de faire une étude comparative de CIGA+ avec d'autres algorithmes puisqu'il n'existe aucune procédure qui utilise le même type d'approximation que la notre. Toutefois, BAMBOO est un algorithme efficace de génération des MFF basé sur *CLOSET+* qui permet aussi de faire de l'approximation mais sous d'autres formes et hypothèses. Comme les deux algorithmes partagent un objectif similaire, nous en faisons une étude comparative dans cette section. Cependant, nous écartons l'usage du support décroissant dans BAMBOO et optons pour un support fixe puisque aucune méthode rigoureuse n'est fournie par les auteurs pour générer la fonction du support décroissant pour une base donnée.
Les expérimentations ont été conduites sous GNU Linux en utilisant un Pentium IV à 3 Ghz et avec 1024 Mo de mémoire. La comparaison des deux algorithmes s'est faite selon leur temps d'exécution et le nombre de MFF générés en considérant trois types de bases de

transactions connues dans la communauté de fouille de données : *Chess*, *Pumsb* et *Mushroom*.
Comme le montrent les figures ci-après, CIGA+ affiche des temps d'exécution plus faibles car il génère moins de MFF. Mais dans tous les cas, le taux d'accroissement des courbes que nous obtenons est fortement similaire pour les deux algorithmes, ce qui montre que les performances sont identiques si on se rapporte à la même échelle. Toutefois, CIGA+ est relativement plus stable et offre une performance relative plus avantageuse lorsque le volume de données devient grand ou lorsque le support minimal devient très faible. Par exemple, avec *minsupp* = 45%, BAMBOO génère une sortie dix fois plus volumineuse que celle produite par CIGA+, et nécessite 51 fois plus de temps a s'exécuter avec la base Pumsb en utilisant *mintol*= 95 % et *mincooc* = 10 %.

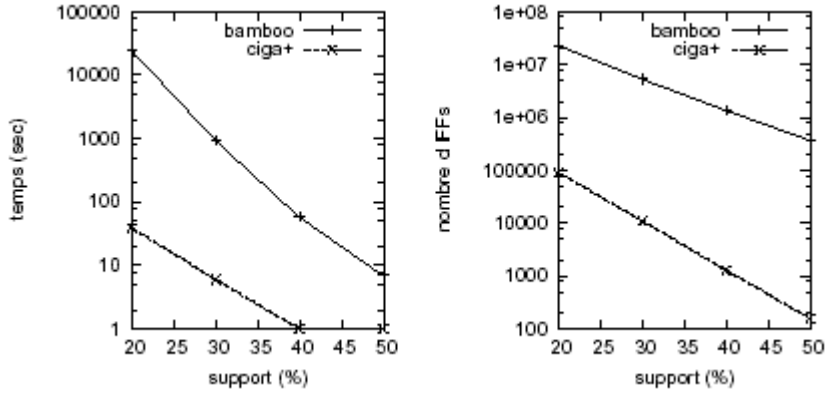

FIG. 5: Influence du support sur les temps d'exécution et le nombre de **MFF** pour la base de données Chess.

La figure 5 montre les performances de CIGA+ par rapport a BAMBOO lorsque *mincooc* = 10 % et *mintol* = 95 %. La valeur non nulle du premier paramètre permet d'écarter certains MFF peu pertinents conduisant par la suite a un ensemble plus faible de règles d'associations. Avec *minsupp* = 20 %, CIGA+ génère une sortie qui est 254 fois plus petite que celle de BAMBOO et s'exécute 620 fois plus rapidement que BAMBOO. La courbe de la figure 6 confirme nos conclusions pour la base de données dense *Pumsb*.

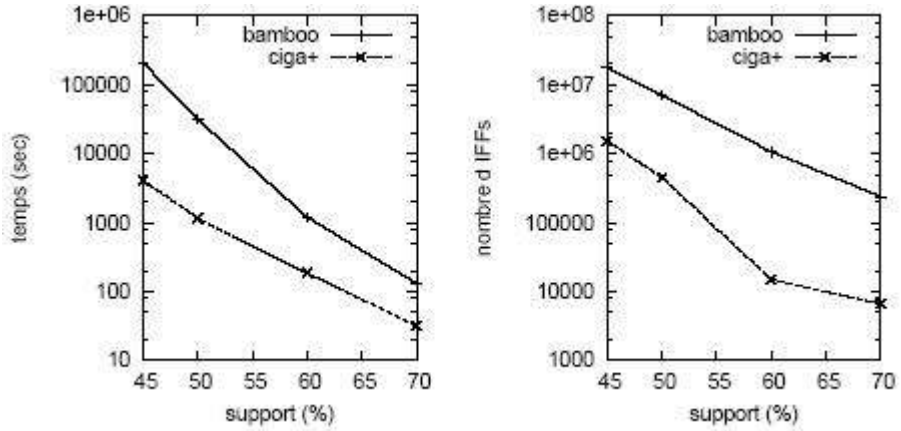

FIG. 6: Influence du support sur les temps d'exécution et le nombre de **MFF** pour la base de données *Pumsb*.

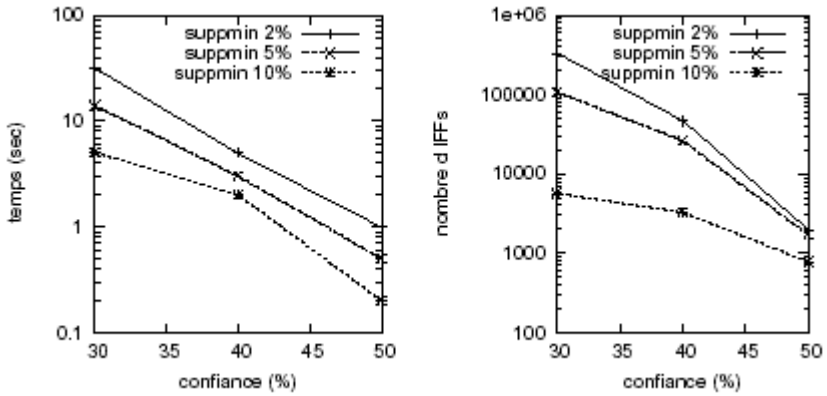

FIG. 7: Influence du paramètre *mincooc* sur la base de données *Mushroom*.

Nous n'avons pas illustré l'influence du seuil de tolérance dans les graphiques. En effet, le comportement de CIGA+ face au paramètre *mintol* dépend fortement des données qui sont utilisées. Pour des données denses, l'élagage a tendance à être plus important au fur et à mesure que ce seuil diminue.

6 Conclusion et travaux futurs

Nous avons présenté un algorithme appelé CIGA+ qui permet de calculer un ensemble concis de MFF en se focalisant sur ceux qui sont les plus prometteurs. CIGA+ réduit l'ensemble des MFF grâce à l'utilisation de deux nouveaux paramètres : la confiance entre deux items élémentaires et le seuil de tolérance. Le premier paramètre a pour but d'écarter les liens faibles entre deux items individuels. Le seuil de tolérance *mintol* permet de simplifier le graphe de dépendances en identifiant les paires d'items qui apparaissent souvent ensemble. Les arêtes peu pertinentes du graphe de dépendances sont ainsi écartées, ce qui améliore le parcours du graphe et réduit l'ensemble de MFF qu'il contient. Le paramétrage de *mintol* dépend du degré d'approximation souhaité. Une valeur de 100 % permet d'obtenir l'ensemble complet de MFF tandis qu'une valeur faible conduit vers une forte approximation de cet ensemble. En exploitant ces deux paramètres, l'utilisateur est en mesure d'avoir le contrôle sur le temps d'exécution de l'algorithme ainsi que sur la taille du résultat et le nombre de règles d'association qu'il souhaite obtenir.

Notre prochaine étape consistera à enrichir le cadre de CIGA+ en lui permettant de calculer les générateurs ainsi que les inclusions (ordre partiel) entre les MFF (Ganter et Wille, 1999) générés afin d'exploiter nos travaux antérieurs sur la génération de règles d'associations et la construction du treillis de concepts. Le but étant de faire de CIGA+ une procédure complète pour la génération des règles d'associations. Le cadre proposé pourra aussi être adapté pour permettre une construction efficace du treillis de concepts et de l'iceberg de concepts (Stumme, Taouil et al., 2002).

Remerciements

Nous remercions Le Conseil de Recherches en Sciences Naturelles et en Génie du Canada (CRSNG) et le Fonds Québécois de la Recherche sur la Nature et les Technologies (FQRNT) pour les subventions accordées. Nous remercions également Professeur Jianyong Wang de nous avoir fourni l'exécutable de l'algorithme BAMBOO.

(Agrawal et Srikant, 1994) Agrawal, R., Srikant, R. (1994). *Fast algorithms for mining association rules.* In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94), Santiago, Chile, pages 487-499.
(Bastide, Taouil et al., 2000) Bastide, Y., Taouil R., Pasquier N., Stumme G., Lakhal L. (2000). *Mining Frequent Patterns with Counting Inference.* SIGKDD Explorations, ACM Computer, 2(2), pages 66-75.
(Bayard, 1998) Bayard, R.J., (1998). *Efficiently Mining Long Patterns from Databases*. In Proc. of the ACM SIGMOD Conference on Management of Data, pages 85-93.
(Boros, Gurvich et al., 2002) Boros, E., Gurvich, V., Khachiyan, L., Makino K., (2002). *On the Complexity of Generating Maximal Frequent and Minimal Infrequent Sets*. Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science, pages 133-141.
(Burdick, Calimlim et al., 2001) Burdick, D., Calimlim, M., Gehrke, J., (2001). *MAFIA : a maximal frequent itemset algorithm for transactional databases.* In Proceedings of the 17th IEEE ICDE Conference (ICDE'01), Heidelberg, Germany, pages 443-452.
(Ganter et Wille, 1999) Ganter, B., Wille, R., (1999). *Formal Concept Analysis : Mathematical Foundations*. Springer, Berlin-Heidelberg.
(Godin et Missaoui, 1994) Godin, R., Missaoui, R., (1994), *An incremental concept formation approach for learning from databases*. Theoretical Computer Science, volume 133, pages 387-419.
(Han, Pei et al., 2000) Han, J., Pei, J., Yin, Y., (2000). *Mining Frequent Patterns without Candidate Generation*. Proc. 2000 ACMSIGMOD Int. Conf. on Management of Data (SIGMOD'00), Dallas, TX.
(Hipp, Guntzer et al., 2000) Hipp, J., Guntzer, U., Nakhaeizadeh, G., (2000). *Algorithms for association rule mining - a gen-eral survey and comparison*. SIGKDD Explorations, 2(1), pages 58-64.
(Lakhal, Pasquier et al., 1999) Lakhal, L., Pasquier, N., Bastide, Y., Taouil, R., (1999). *Efficient mining of association rules using closed itemset lattices*. Information Systems, Volume 24 , pages 25-46.
(Luxemburger, 1991) Luxemburger, M., (1991), *Implications partielles dans un contexte*. Mathmatiques et Sciences Humaines, 29(113), pages 35-55.
(Pasquier, Bastide et al., 1999) Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L., (1999). *Efficient Mining of Association Rules using Closed Itemset Lattices*. Information Systems, Elsevier Science, 24(1), pages 25-46 .
(Pasquier, 2000) Pasquier, N., (2000). *Data mining : algorithmes d'extraction et de rduction des rgles d'association dans les bases de donnes*. Thse de doctorat, Universit de Clermont-Ferrand II.
(Pei, Han et al., 2000) Pei, J., Han, J., Mao, R., (2000). *Closet : An efficient algorithm for mining frequent closed itemsets*. In SIGMOD Int'l Workshop on Data Mining and Knowledge Discovery, pages 21-30.
(Pfaltz et Taylor, 2002) Pfaltz, J., Taylor, C., (2002). *Scientific discovery through iterative transformations of concept lattices*. In proceedings of the 1st International Workshop on Discrete Mathematics and Data Mining, Washington (DC), pages 65-74.
(Stumme, Taouil et al., 2002) Stumme, G., Taouil, R., Bastide, Y, Pasquier, N., Lakhal, L., (2002). *Computing Iceberg Concept Lattices with Titanic*. Data and Knowledge Engineering, 42(2), pages 189-222.
(Valtchev, Missaoui et al., 2003) Valtchev, P., Missaoui, R., Hacene, M. R., Godin, R., (2003). *Incremental maintenance of association rule bases*. In Proceedings of the 2nd Workshop on Discrete Mathematics and Data Mining, San Francisco.
(Valtchev, Missaoui et al., 2004) Valtchev, P., Missaoui, R., Godin, R., (2004). *Formal Concept Analysis for Knowledge and Data Discovery : New Challenges, Proceedings of the Second International Conference on Formal Concept Analysis.* ICFCA, Sydney, Australia, pages 352-371.
(Wang, Han et al., 2003) Wang, J., Han, J., Pei, J., (2003). *CLOSET+ : searching for the best strategies for mining frequent closed itemsets*. Ninth ACM SIGKDD international conference on Knowledge discovery and data mining.
(Wang et Karypis, 2004) Wang, J., Karypis, G., (2004). *BAMBOO : Accelerating Closed Itemset Mining by Deeply Pushing the Length-Decreasing Support Constraint*. In Proceedings of the SDM'04.
(Zaki, 2000) Zaki, M.J., (2000). *Generating non-redundant association rules*. In Proceedings of the KDD'00, pages 34-43.
(Zaki et Hsiao, 2002) Zaki, M.J., Hsiao, C.J., (2002). *CHARM : An Efficient Algorithm for Closed Itemset Mining*, In Proceeding of the 2nd SIAM International Conference on Data Mining (ICDM'02), Arlington.