CHAPITRE 3

LES ALGORITHMES DE GENERATION DES REGLES D'ASSOCIATION

3.1 Introduction :

Dans le domaine du data mining la recherche des **règles d'association** est une méthode populaire étudiée d'une manière approfondie dont le but est de découvrir des relations ayant un intérêt pour le statisticien entre deux ou plusieurs variables stockées dans de très importantes bases de données. Piatetsky-Shapiro[1]présentent des règles d'association extrêmement fortes découvertes dans des bases de données en utilisant différentes mesures d'intérêt.

En se basant sur le concept de relations fortes, Rakesh Agrawal et son équipe présente des règles d'association dont le but est de découvrir des similitudes entre des produits dans des données saisies sur une grande échelle dans les systèmes informatiques des points de ventes des chaînes de supermarchés. Par exemple, une règle découverte dans les données de ventes dans un supermarché pourrait indiquer qu'un client achetant des oignons et des pommes de terre simultanément, serait susceptible d'acheter un hamburger. Une telle information peut être utilisée comme base pour prendre des décisions marketing telles que par exemple des promotions ou des emplacements bien choisis pour les produits associés. En plus des exemples ci-dessus concernant le panier de la ménagère, les règles d'association sont employées aujourd'hui dans plusieurs domaines incluant celui de la fouille du web, de la détection d'intrusion et de la bio-informatique [5].

Le problème de génération des règles d'association peut être scindée en deux sous problèmes:

1. Le premier sous problème consiste à déterminer l'ensemble des itemsets fréquents dans la base de transaction T.
2. Le deuxième sous problème consiste à générer les règles d'association à partir de l'ensemble des itemesets fréquents.

Larecherche des règles d'association est basé sur plusieurs algorithmes, ces algorithmes sont classés selon deux (02) grandes approches algorithmiques pour l'extraction des régles d'association :

- **Approche classique basée sur l'extraction d'itemsets fréquents.**
- **Approche d'expansion des règles (nouvelle approche) [5].**

3.2 Génération des Itemset fréquents :

En peut déduire plusieurs algorithmes d'extraction des itemsets fréquents, Ces derniers sontutilisés pour générer les règles d'association, parmi ces algorithmes, on peut citer les suivantes :

- Algorithme Apriori (Agrawal &Srikant, 1994) (voir le CHAPITRE II)
- Algorithme FP-GROWTH (Han et al., 2004)
- Algorithme ECLAT (Zaki, 2000)
- Algorithme FIN (Deng et al, 2014)
- Algorithme PrePost+ (Deng et Lv, 2015)

3.2.1 Algorithme FP-GROWTH:

Cette méthode s'appelle FP-growth (Frequent Pattern growth). Elle consiste d'abord à compresser la base de données en une structure compacte appelée FP-tree (Frequent Pattern tree), puis à diviser la base de données ainsi compressée en sous projections de la base de données appelées bases conditionnelles.

Chacune de ces projections est associée à un item fréquent. L'extraction des itemsets fréquents se fera sur chacune des projections séparément.

L'algorithme FP-growth apporte ainsi une solution au problème de la fouille de motifs fréquents dans une grande base de données transactionnelle. En stockant l'ensemble des éléments fréquents de la base de transactions dans une structure compacte, on supprimer la nécessité de devoir scanner de façon répétée la base de transactions. De plus, en triant les éléments dans la structure compacte, on accélère la recherche des motifs [6].

Structure d'un FP-tree :

Deux éléments essentiels constituent la structure d'un FP-tree. Ainsi les deux éléments qui composent cette structure sont :

- Une structure sous forme d'un arbre avec une racine étiquetée *'null'*.
- Un index (une table des pointeurs des items fréquents).

L'arbre quant à lui est composé, comme indiqué, d'une racine 'null' et d'un ensemble de noeuds.

Un noeud de l'arbre est composé par :

- Le nom de l'item (nom-item). Il s'agit de l'item qui représente le noeud.
- Le nombre d'occurrence (count) de transaction où figure la portion de chemin jusqu'à ce noeud.

- Un lien vers le noeud suivant dans l'arbre (node-link). Il s'agit d'un lien inter-noeud vers les autres occurrences du même élément (ayant le même nom-item) figurant dans d'autres séquences de transactions. Cette valeur prend la valeur ***null***s'il n'y a pas un tel noeud [6].

L'Index est une table d'en-tête qui contient la liste des items fréquents et qui pointe sur la première occurrence de chaque élément. Chaque entrée dans cette table contient :

- Le nom de l'élément (nom-item).
- Le pointeur tête de la séquence des noeuds ayant ce même nom-item.

L'avantage de cette représentation des données est qu'il suffit de suivre les liens inter-noeuds pour connaître toutes les associations fréquentes où figure l'élément fréquent.

Construction d'un FP-Tree :

La construction du FP-tree nécessite deux parcours de la base des transactions (T) et se fait de la manière suivante :

1. On effectue un premier parcours de la base T pour déterminer les items fréquents en fonction du support minimum fourni. Ces items seront triés par la suite par ordre décroissant de support dans une liste (L). Les items ainsi triés seront traités dans cet ordre.
2. Un second parcours de T est alors effectué. Chaque transaction est triée selon l'ordre des items dans L. Le noeud racine de l'arbre {null} est d'abord crée. Durant ce même parcours, une branche sera crée pour chaque transaction, mais les transactions ayant un même préfixe partageront le même début d'une branche de l'arbre, ainsi deux transactions identiques seront représentées par une seule et même branche.

La raison pour laquelle les items sont traités du plus fréquent au moins fréquent est que les items fréquents seront proches de la racine et seront mieux partagés par les transactions. [6].

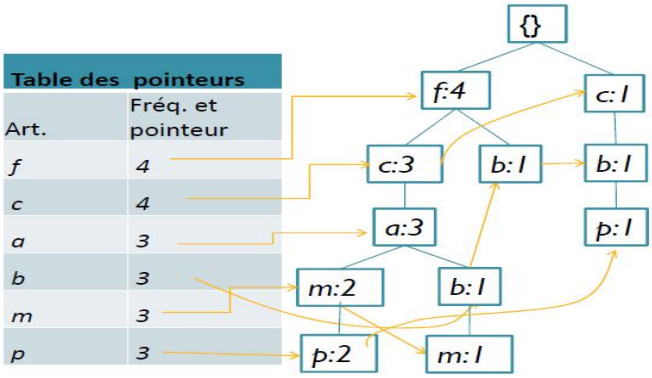


Figure 3.1 : Exemple d'un FP-tree construit à partir d'une base de transactions [6]

Avantages et inconvénients :

L'avantage majeur de l'algorithme est qu'il ne fait que deux balayages de la base des transactions. Le premier balayage étant pour trouver les k-itemset puis trier le résultat pour construire la liste des items fréquents dans T en ordre de fréquence. Le deuxième balayage ayant pour but la construction de la structure FP-tree. Aussi il peut être qualifié de :

- Complet :
    - puisqu'il contient toutes les informations sur les éléments fréquents
    - pour chaque élément fréquent, on affecte un chemin dans l'arbre.
- Concis :
    - puisqu'il ne contient que les éléments fréquents. les items sont classés en ordre de fréquence décroissante.
    - Plus un item est fréquent, plus il sera utilisé par les itemsets fréquent. Par conséquent son noeud sera utilisé souvent.

Néanmoins, malgré sa structure compacte, cela ne garantit pas, dans le cas ou la base de transactions est trop volumineuse, que toute la structure du FP-tree tiendra en mémoire centrale. De plus la construction de la strcuture FP-tree peut s'avérer longue et pourrait consommer beaucoup de ressources système [6].

Etapes de la construction du FP-tree :

Un *Frequent Pattern tree* est composé d'un noeud racine null associé à un ensemble de noeuds. Ces noeuds sont eux mêmes composés par :

- Le nom de l'élément.
- Le nombre d'occurence dans la base de transactions.
- La portion du chemin (composée de tous les éléments rencontrés de la racine jusqu'au noeud concerné).
- Un lien inter-noeud vers les autres occurencesdu même éléments figurant dans d'autres séquences de transactions.

L'avantage de cette structure de données est, qu'en suivant les liens inter-noeuds, on peut facilement connaître toutes les associations fréquentes où figure l'élément fréquent.

La construction d'une structure FP-tree passe par 6 étapes principales. Les étapes 1 à 5 prépare la structure et insère les éléments qui doivent s'y trouver. La sixième étape consiste à valider les informations insérées dans les étapes précédentes :

1. Calculer le support minimum
2. Parcours de la base des transactions pour trouver la somme totale des différentes occurences.

3. Définir la priorité des éléments, puis trier les items en fonction de leur priorité.
4. Création du noeud racine.
5. Insertion des noeuds enfants.
6. Validation.

Etape 1:

Considérons la base de transactions suivante. Supposons que le support minimum est défini à **50%**.

TID	Items
1	f, a, c, d, g, i, m, p
2	a, b, c, f, l, m, o
3	b, f, h, j, o
4	b, c, k, s, p
5	a, f, c, e, l, p, m, n

Tableau3.1 : Base de Transaction [6]

Dans notre cas, pour obtenir un support de 50%, il faut le calculer ainsi :

Support minimum = (**50**/100 * **5**) = **2.5**

- **50** étant le pourcentage minimum demandé,
- **5** étant le nombre total de transactions dans la base.

Si la valeur obtenue n'est pas entière, elle sera arrondie. par exemple :

Ceiling (**50**/100 * **5**) = **3**

Le résultat obtenu, **3** dans notre cas, constitue le support minimum et par conséquent tous les items da la base de transactions ayant un suport inférieur à 3 occurences minimum sera ignoré.Cette étape étant terminée, passons à l'étape suivante [6].

Etape 2 :

Dans cette étape nous allons parcourir la base de transactions afin de calculer les fréquences des éléments qui s'y trouvent. Par la suite, une fois les différentes fréquences obtenues, seuls les éléments dont la fréquence est supérieure au support minimum défini dans l'étape 1 seront retenus, les autres seront ignorés. Dans notre cas, le tableau suivant représente les items retenus ainsi que leurs nombre d'occurence respectif.

Item	Fréquence
f	4
a	3
c	4
b	3
m	3
p	3

Tableau3.2 : Occurences des items[6]

La table ainsi obtenue constitue la *table des entêtes* (Headers Table) appellée aussi la *table des pointeurs*. Maintenant que nous avons determiné nos différents items, il faut les *prioritiser*. Cette tâche est réalisée dans la prochaine étape de la construction [6].

Etape 3 :

Cette étape consiste à ordonner les différents éléments en fonction de leur poids. Il s'agit de les trier en fonction de leur nombre d'occurences. Ce tri s'effectue en ordre décroissant, l'élément ayant comptabilisé le plus grand nombre d'occurences est placé en tête et l'élément ayant comptabilisé le moins d'occurences est placé en queue. Ce traitement sera effectué pour chacune des lignes de transactions contenues dans la base des transactions.

TID	Items	Items Fréquents Ordonnés
1	f, a, c, d, g, i, m, p	f, c, a, m, p
2	a, b, c, f, l, m, o	f, c, a, b, m
3	b, f, h, j, o	f, b
4	b, c, k, s, p	c, b, p
5	a, f, c, e, l, p, m, n	f, c, a, m, p

Tableau3.3 : Items Fréquents Ordonnés [6]

Dans notre cas, l'élément ***f*** ayant un support de ***4*** est placé en tête, l'élément ***p*** quant à lui a un support de ***3*** est par conséquent il se retrouve en dernière position.

A la fin de cette étape, on peut considérer que tout est prêt pour commencer la construction de la structure FP-tree avec la création et l'insértion des différents noeuds [6].

Etape 4 :

A partir du résultat obtenu lors de l'étape précédente, nous commençons la construction de la structure FP-tree. Tout d'abord l'élément 'Racine' de l'arbre est créé. Cet élément racine ne contiendra aucun élément. Il contiendra uniquement des liens vers ses éléments enfants.

On commence par parcourir chaque élément de la transaction (voir illustration plus loin). Puis pour chaque élément de la transaction on vérifie l'éxistance d'un noeud correspondant, s'il n'existe pas, le noeud est créé, dans le cas contraire le nombre d'occurence est incrémenté. Puis pour chaque élément créé on va établir un lien depuis la table des entêtes vers l'élément inséré dans l'arbre.

A ce stade l'arbre est encore vide, donc la procédure de vérification de l'exsitance d'un noeud en particulier indiquera qu'il n'existe pas et par conséquent il faudra le créer.

La première transaction est composée des éléments (f, c, a, m, p) triés par ordre décroissant en fonction de leur poids. L'élément ***f*** étant le premier de la liste, un noeud correspondant est inséré à partir de l'élément racine de l'arbre. Le noeud ***f*** contient le compte de ***1*** car c'est la première fois que l'on insère cet élément. Un lien est établi entre l'élément racine de l'arbre et l'élément ***f*** puis un autre lien est établi à partir de la table des entêtes. On porsuit de même avec les éléments suivants (c, a, m, p). Ainsi on obtient la structure illustrée par cette figure.

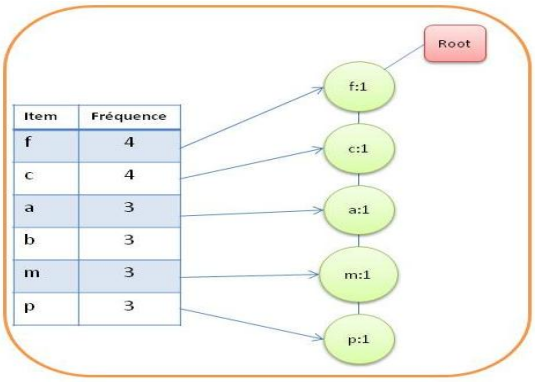


Figure 3.2 : Construction FP-tree à partir de la racine

Etape 5 :

La construction se poursuit avec la deuxième transaction qui est composée des éléments (f, c, a, b, m). Cette fois-ci l'arbre contient des éléments et par conséquent pour chaque élément trouvé son nombre d'occurences est incrémenté de 1.

La transaction contient l'élément ***f***, l'arbre aussi. Par conséquent le nombre d'occurence de ***f*** passe à ***2*** (1 + 1). Il en est de même pour les élémens ***c***, et ***a***. Nous arrivons à l'élément ***b***. Il n'existe pas d'éléments ***b*** correspondant, donc un nouveau noeud est créé à partir de notre position actuelle, c.à.d le noeud ***a*** et un nouveau lien est créé à partir de ***a*** vers ***b*** puis un lien à partir de la table des entêtes vers le nouveau élément inséré. Concenrnat l'élément ***m*** restant,

étant donné qu'il n'existe pas de noeud correspondant à partir de notre position (noeud***b***) un nouveau noeud est créé et initialisé avec une valeur de ***1***, puis en plus du lien créé à partir de l'élément ***b***, un lien est créé à partir du noeud m déjà existant. Ceci est illustré par la figure ci-dessous.

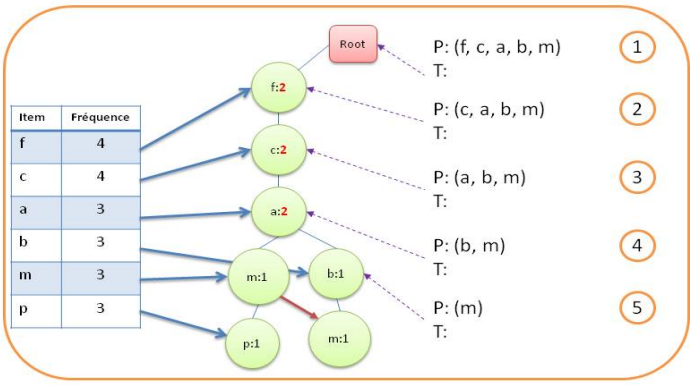


Figure 3.3 : Construction FP-tree à partir de la 2ème transaction

A la fin du traitement de toutes les transactions de la base la structure finale de Fp-tree est illustrée par la figure suivante.

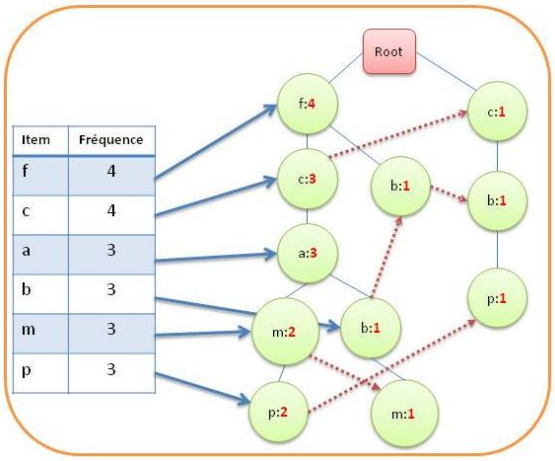


Figure 3.4 : Etat final de la structure FP-tree

Etape 6 :

Cette dernière étape consiste à valider les informations de l'arbre. Comment savoir si elles sont correctes ? La réponse à cette question est très simple, il suffit de comparer les informations obtenues à partir des différents noeuds de l'arbre avec les informations de la table des entêtes.Pour cela, il faut compter, et additionner si besoin, toutes les occurences d'un éléments dans l'arbre et comparer le résultat obtenus à celui stocké dans la table des entêtes.Ainsi après avoir compter tous les éléments de la structure, on obtient le résultat suivant :(f:4, c:4, a:3, b:3, m:3, p:3) ce qui est conforme à la tables des entêtes.

Fonctionnement :

Pour expliquer comment sont identifiés les itemsets fréquents je vais me baser sur le même exemple de l'article précédent. A la fin de la deuxième étape de la construction du FP-tree nous avons pu définir la table des entêtes (*Headers Table*). Cette table, tout comme le FP-tree obtenu à la fin de la phase de construction, vont nous servir à identifer les itemssets fréquents.

Rappelons que, dans notre exemple le support minimum à été défini à **3**, et que la table des enêtes a été triée en fonction du poids (nombre d'occurences) de chaque élément. De même, seuls les éléments ayant un support **> 3** ont été retenus lors de la préparation de la table des entêtes [6].

Lors de la construction du FP-tree, des liens ont été créés entre les noeuds et la table des entêtes afin de faciliter la navigation. Par conséquent tous les noeudsa_i peuvent être atteints en suivant la chaine des pointeurs des champs noeud-lien en commençant par celui de la table des pointeurs correspondant à l'item fréquent a_i.

Egalement, tous les motifs contenant ai doivent être un sous-ensemble d'un chemin partant d la racine et contenant a_i[6].

Identifier les motifs fréquents contenant p :

Pour identifer les motifs fréquents contenant l'élément *p*, la table des entêtes est parcourue de bas en haut ; commencant par l'élément *p* de la table et se terminant par l'élément *c* (le dernier élément *f* est ignoré).

A partir de la table des pointeurs, on parcourt le FP-tree en suivants les pointeurs pour l'item fréquent p. Puis, nous créeons tous les chemin à partir de la racine et se terminant par *p* pour former la *base des motifs p-conditionnés*. Ceci constitue l'ensemble des préfixes de *p*.

L'élément **p** peut être atteint en suivant deux chemins à partir de la racine : (élément occurence)

- (f : 4, c : 3, a : 3, m : 2, p : 2)
- (c : 1, b : 1, p : 1)

Les transactions contenant 'p' doivent avoir le nombre d'occurence de ***p***, par conséquent nous obtenons :

- (f : 2, c : 2, a : 2, m : 2, p : 2)
- (c : 1, b : 1, p : 1)

Etant donné que 'p' fait parti de ces transactions, nous pouvons l'ignorer et nos chemins, appellés *Motif de Base Conditionné* (*Conditional Pattern Base, CPB*) devient :

- (f : 2, c : 2, a : 2, m : 2)
- (c : 1, b : 1)

Un *Chemin de Base Conditionné* contient toutes les transactions dans lesquelles un élément occure. Pour trouver tous les motifs fréquents contenant un élément (en l'occurence 'p')n nous avons besoin de trouver tous les motifs fréquents dans le CPB puis leur ajouter l'élément. Ceci peut être fait avec la construction d'un nouveau FP-tree pour le CPB [6].

Pour terminer l'identification des motifs fréaquents contenant ***p***, on va, pour chaque motif dans la base des motifs ***p-conditionnés***, puis évaluer le nombre de chaque item et construire le FP-tree correspondant à partir de cette base des motifs.

(f : 2, c : 2, a : 2, m : 2), (c : 1, b : 1) => (c : 3)

Seul l'élément ***c*** est fréquent ca c'est le seul élément qui un nombre d'occurences supérieur au support minimum. Là aussi, il faut éliminer tous les éléments dont le support est inférieur au support minimum et les prioritiser comme décrit précédemment. La valeur de l'occurence du motif fréquent à identifier est extraite de la valeur du support obtenu par le sous-arbre. Par conséquent le motif fréquent contenant p est:**(p:3, cp:3)**[6]

Identifier les motifs fréquents contenant m :

Pour identifier les motifs fréquents contenant l'élément m, il suffit de suivre la même procédure de l'identification de l'élément p.

L'élément ***m*** peut être atteint en suivant deux chemins à partir de la racine, l'identification des chemins **m-conditionnés** nous donne :

- (f : 4, c : 3, a : 3, m : 2, p : 2) => (f : 2, c : 2, a : 2)
- (f : 4, c : 3, a : 3, b : 1, m : 1) => (f : 1, c : 1, a : 1, b : 1)

Il faut noter que seuls les préfixes (tous ce qui est avant 'm') sont pris en considération. C'est un moyen systématique pour éviter de (re)considérer l'élément 'p'.

L'élément ***b*** n'étant pas fréquent, il sera retiré. Par conséquent le FP-tree ***m-conditionné*** devient :

(f : 3, c : 3, a : 3)

Les motifs fréquents identifiés reliés à l'élément ***m*** sont : **m, fm, cm, am, fcm, fam, cam, fcam.** [6]

3.2.2 Algorithme ECLAT :

Introduit par M. J. Zaki en 1997, cet algorithme repose sur le découpage de la base en classes d'équivalences et distribution de la charge de travail sur tous les processeurs. On considère que deux itemsets (ensemble d'items) sont dans la même classe d'équivalence s'ils sont désignés par les items qu'ils contiennent dans l'ordre lexicographique, possédent un préfixe commun. Par exemple, les itemsets ABC et ABD sont dans la classe d'équivalence

AB. Au lieu de transmettre des supports locaux ou desportions de base de données comme dans les principaux algorithmes dérivés d'Apriori, cet algorithme fonctionne en transmettant les listes de transactions correspondant à chaque classe d'équivalence au processeur qui s'occupe de celle-ci. Cet algorithme sera détaillé par la suite [7].

Synthèse :

Les algorithmes ici présentés fonctionnent suivant deux approches différentes, les algorithmes basés sur Apriori étudient les transactions, c'est à dire des ensembles d'items, alors qu'Eclat étudie des 8 ensembles de transactions par classe d'équivalence d'items. On parle ici de placements respectivement horizontal et vertical de la base tels que les illustre la figure3.4

Items / Transactions	A	B	C	D
T_1	1	0	1	1
T_2	0	0	1	1
T_3	1	0	1	0

Tableau3.4 : Formes horizontale et verticale de la base.

Partitionnement en classes d'équivalences :

En définissant une application bijective ϕ de l'ensemble des items vers l'ensemble des entiers, on peut définir un ordre total sur l'ensemble des items. En effet si ϕ est bijective, alors chaque item dispose d'un identifiant unique et chaque identifiant est propre à un item. Dans la suite pour clarifier lanotation, on désignera un item par une lettre majuscule et l'ordre utilisé sera l'ordre lexicographique.

Un ensemble d'items sera représenté par une suite de lettres majuscules qu'on ordonnera donc suivant l'ordre lexicographique. Disposant d'un ordre sur les items, on peut définir des classes d'équivalences sur les itemsets basées sur les préfixes communs de l'écriture de deux itemsets.

Les items A, B, C et D sont tels que $\phi(A) < \phi(B) < \phi(C) < \phi(D)$ alors, les itemsets s'écrivent dans l'ordre croissant de la valeur par ϕ de leurs éléments. Ainsi, l'itemset contenant les items B, A et C s'écrit ABC, celui contenant les items D, B et A s'écrit ABD et ces deux itemsets appartiennent à la classe d'équivalence E_{AB} définie par leur préfixe commun de taille |itemset|- 1, AB [7].

L'algorithme 'Eclat':

Phase d'initialisation

Scanne la partition locale de la base.

Calcul des comptes locaux des itemsets de taille 2.

Construction des comptes globaux de L_2.

Phase de transformation

Partitionnement de L_2 en classes d'équivalences.

Distribution de L_2 sur les processeurs par classe d'équivalence.

Transformation de la base locale en base verticale.

Envoie des listes de transaction aux autres processeurs.

L_2 **local** = listes de transaction des autres processeurs.

Phase asynchrone

Pour chaque classe d'équivalence E_2 dans L_2 **local**

Construction(E_2*)* ;

Phase finale de réduction

Regroupement des résultats et calcule des associations.

❖ Phase d'initialisation :

L'algorithme commence par scanner la base afin de construire les itemsets fréquents de taille 2.En effet, il est possible de générer ces ensembles avec peu de coût supplémentaire par rapport à la génération des itemsets fréquents de taille 1, profitant ainsi du gain obtenu en évitant de scanner labase 2 fois. Cependant, l'auteur de l'algorithme précise que si la base de données contient un grand nombre d'items, il est peut-être préférable de scanner la base deux fois afin d'éliminer les items infréquents avant de générer les itemsets de taille 2. A ce stade, on dispose de l'ensemble des itemsets fréquents L_2.

- On scanne la base de données.
- On construit un tableau de deux dimensions indexé par les items sur la hauteur et la largeur.

items	A	B	C	D
A	-	-	-	-
B	1	-	-	-
C	1	0	-	-
D	0	1	1	-

Tableau3.5 : Table de transaction

Dans cet exemple pour une base contenant 4 items, on rencontre les itemsets AB, AC, BD et CD chacun respectivement dans au moins une transaction de la base.

Chaque processeur calcule le support local des itemsets puis effectue une réduction de sommedes résultats des autres processeurs afin de contruire les support globaux de chaque itemset de L_2 [7].

❖ La phase de transformation :

L'algorithme commence par partitionner L_2 en classes d'équivalences qui seront redistribuées sur les processeurs avec une politique d'équilibrage de charge basée sur une heuristique. On effectue alorsune transformation de la base afin d'obtenir, non plus une liste d'items par transaction mais une listede transactions par item (transformation verticale de la base.)

- Partitionnement de L_2 en classes d'équivalences.
- Calcul de la charge de travail pour chaque classe d'équivalence.

La mesure est effectuée en fonction du nombre d'éléments s de la classe d'équivalence. Onconsidère toutes les paires à traiter par la suite. Ainsi, la charge est calculée par la valeur ${}_{s}^{2}C$.

Par exemple, si dans la classe d'équivalence [A] on trouve les itemsets AB, AC et AD, lacharge calculée sera ${}_{3}^{2}C = 3$. Il s'agit alors de repartir les tâches à effectuer sur les processeursen fonction d'une heuristique sur les charges calculées : On assigne toutes les classes d'équivalences par charge décroissantes sur le processeur qui a la charge la plus petite au moment del'assignation.

[A] = {AB, AD, AE, AF, AH} s = 10
[B] = {BC, BD, BE, BF} s = 6
[C] = {CD, CE, CF} s = 3
[D] = {DE, DF, DH, DI, DJ, DK} s = 15
[E] = {EF, EG, EH} s = 3
[F] = {FG, FH} s = 1

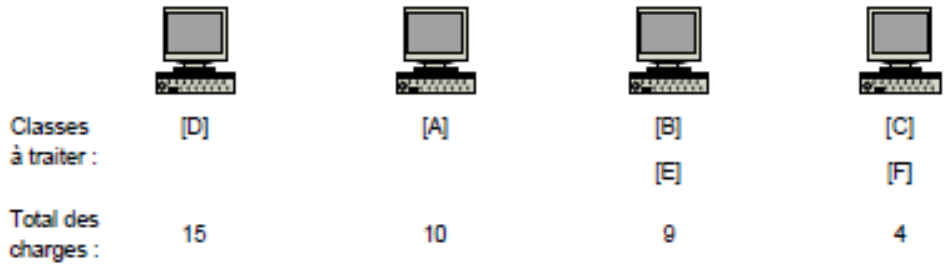


Figure 3.5 – Répartition des charges de travail.

Chaque processeur peut effectuer cette tâche indépendamment des autres puisqu'ils disposentalors tous des supports globaux de L_2.

- Phase de transformation verticale de la base.

Chaque processeur scanne sa portion locale de la base afin de construire les listes de transactionscorrespondant aux classes d'équivalence de L_2. Il faut alors transmettre respectivement les listes aux processeurs chargés de la classe d'équivalence correspondante [7].

- La phase asynchrone :

Algorithme *Construction (*E_{k-1})
Pour tous les itemsets I_1et $I_2 \epsilon$ E_{k-1}
Si ((I_1.transactions ∩ I_2.transactions) >= support minimum)
Ajouter (I_1U I_2) à L_k
Partitionner $L_k$en classes d'équivalences.
Pour chaque classe d'équivalence $E_k \epsilon L_k$
Construction (E_k*)*

Les processeurs effectuent concurramment la construction des itemsets de tailles croissantes par intersection des listes de transactions des éléments de chaque classe d'équivalence entre eux. Eliminant les itemsets de support insuffisant, on réduit rapidement le travail a effectuer en même temps qu'on augmente la taille des itemsets construits. C'est du moins ce qu'il se passe sur des données réelles.

- La phase de réduction finale :

La dernière tâche de l'algorithme consiste en l'accumulation et la réunion des résultats de chaque processeur.

Complexité :

La phase d'initialisation, qui consiste en la création de l'ensemble L_2 des itemsets fréquents de taille 2, nécessite une lecture complète de la base locale T et $\frac{2}{n}C$ opérations pour construire les potentiels $\frac{2}{n}C$ itemsets fréquents de taille 2 avec n le nombre d'items distincts dans T. On effectue alors la communication des supports locaux aux autres processeurs en envoyant au maximum un vecteur de$\frac{2}{n}C$ entiers. Les processeurs dans la construction de L_2, nécessitent un espace mémoire de l'ordre de 2^nafin de construire la table de fréquences des itemsets de taille 2.

Durant la phase de transformation, si on a pris soin de trier les itemsets de L_2, on partitionne l'ensemble en n classes au maximum en temps linéaire par rapport à la taille de L_2 (on parcours une seule fois L_2 afin de trouver les indices de début et de fin de chaque classe d'équivalence). En n étapes au maximum, on aura attribué les classes d'équivalence de chaque processeur. On effectue alors une seconde lecture de la base afin de construire les listes de transactions par item (transformation verticale de la base) ceci est réalisé en N opérations, N étant le nombre total d'apparition de tous les items dans T. On transmet alors les listes de transactions de chaque classe d'équivalence aux autres processeurs [7].

On transmet donc potentiellement des vecteurs de |T| identifiants à chaque processeur. Les listesreçues sont alors réunies et triées afin de faciliter la suite des opérations.

3.2.3 Algorithme FIN (Deng et al., 2014):

Au cours des dernières années, deux nouvelles structures de données ont été proposées N-list (Deng et al, 2012) et Nodelist (Deng et Wang, 2010) pour représenter les itemsets. Les deux structures sont basées sur une structure arborescente appelé PPC-Tree, qui stockent les informations pour chaque motif fréquent.

Un N-liste ou Node-liste est un ensemble trié de nœuds dans un PPC-Tree (prefix post code - tree). Habituellement, les nœuds d'un N-liste ou Node-liste sont triés par l'ordre croissant de la pré-commande de noeuds. Leur principale différence est que chauqe node-listes sont construites par des nœuds descendants tandis que N-listes sont construits par nœuds ancêtres. N-listes ou Node-listes ont deux importantes Propriétés [8] :

- La première est que la N-liste ou Node-list d'une longueur (k + 1) itemset peut être construit en joignant les N-listes ou Node-listes de son sous-ensemble d'une longueur de k.
- L'autre est que le support d'un jeu d'éléments est la somme des chiffres enregistrant dans les noeudsde son N-liste ou Node-liste.

Les deux structures de données précèdentes possèdent deux avantages :

- Le premier est que la longueur de l'extrémité N-liste d'un jeu d'éléments est beaucoup plus petite que la longueur de son nœud liste.
- L'autre est que la N-listes possède un propriété qui s'appele propriété de chemin unique.

Le premier avantage rend le temps d'assemblage de deux N-listes est beaucoup avantagueux que pour joindre deux Node-listes. Cela provoque que l'efficacité du Pré Post est supérieure à celle du PPV [8].

Principe de base :

POC-tree definition (Prefix Ordre code-Tree) : est une structure arborescente basée sur :

- Il se compose d'une racine étiquetée comme"null", et un ensemble deNoeudpréfixe comme sous-arbres (enfants).
- Chaque noeuddans le sous-arbre préfixese compose de cinq domaines : Nom-Noeud, nombre d'occurrence, la liste-enfants, pré-commande. Nom-Noeud registres quel élément le noeud représente. nombre d'occurrence enregistre le nombre des transactions présentées par la partie du trajet à atteindre ce noeud. liste-enfants

enregistre tous les enfantsdu nœud. Pré-commande est le rang de pré-commande du noeud [8].

On peut écrire l'algorithme de construction de POC-Tree comme suit :

Algorithme 1 : la construction de POC-Tree

Entrée : Une base de données de transaction DB et un support minimum ε.

Sortie : Un POC-Tree et F_1 (l'ensemble des 1-itemsets fréquents).

1. Génération des 1-itemsets fréquents :

Parcourir la BD pour une seule passe pour trouver des F_1, l'ensemble des 1-itemsets fréquents (Noeud fréquents), et de leurs supports. Trier F_1 en ordre décroissant en L_1. On notera que, si les supports de certains éléments fréquents sont égaux, les pré-commandes peuvent être affectées de façon arbitraire.

2. [POC-Tree Construction] :

La procédure de construction de POC-arbre est identique à celui de la construction d'un FP-Tree (Han, Pei, et Yin, 2000).Créer la racine de la POC-Tree(Tr), libellé comme "null". Pour chaque transaction Trans dans BD, procédez comme suit : Sélectionnez les éléments fréquents dans Trans et trier en fonction de l'ordre de F_1. Mettez la liste des items fréquent dans le Trans [p | P], où p est le premier élément et P la liste restante. Appelle Ajouter arbre ([p | P], Tr).La fonction Ajouter arbre ([p | P], Tr) est réalisée comme suit :

Si Tr a un enfant N tel que N.item-name = p.item-nom, puis incrémenter le nombre de N par 1;D'autrepart, créer un nouveau noeud N, avec son compteur initialisé à 1, et l'ajouter à la liste des enfants de Tr. Si P est vide, appeler Ajouter arbre (P, N) de manière récursive [8].

3. [Génération de pre-code] :

Scannez le POC-Tree pour générer le pré-ordre de chaque noeud par la traversée pré-ordre.

Example **:** on prend la base de données de transaction, DB, représenté par les informations provenant des deux colonnes de gauche du tableau et avec un seuil Ɛ = 0,4. Les fréquents 1-itemsets mis F1 = {a, b, c, e, f}.

ID	Items	Ordre des items fréquents
1	**a, c, g, f**	**c, f, a**
2	**e, a, c,b**	**b, c, e, a**
3	**e, c, b,i**	**b, c, e**
4	**b, f,h**	**b, f**
5	**b, f, e, c,d**	**b, c, e, f**

Tableau3.6 : Table de transaction

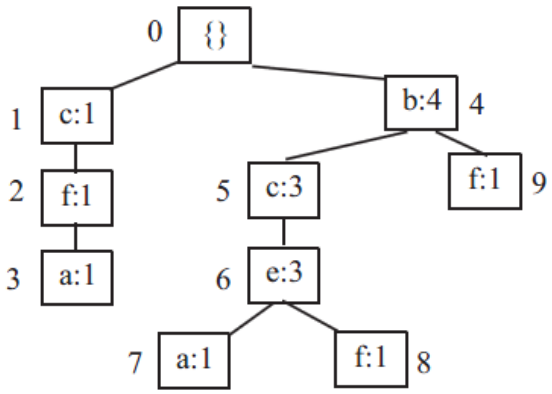


Figure3.6 : construction de l'arbre aprés l'exécution de l'algorithme 1

Nous présentons quelques notions avant d'entamer la description de l'algorithme, parmi ces notions les suivants [8] :

1- Nodesets : definitions et propertiés :

(N-info). Pour un nœud N dans un POC-arbre, nous appelons le paire de son pré-commande et son compteur comme ça (pré-commande, compteur).

Nodeset des items fréquents i est une séquence de tous les N-infos de nœuds enregistrementdans le POC-arbre.

Propriété 1 **:** Étant donné un élément fréquent i, supposant que ses nodeset sont $\{(pre_1, c_1), (Pre_2, c_2),. . ., (Pre_m, c_m)\}$. Le support de i est égal à $c_1 + c_2 +. . . + C_m$.

(>Relation**)** Pour tous deux articles i_1 et i_2 ($i_1, i_2 \in L_1$). $i_1 > i_2$ si et seulement si i_1 est en avance sur i_2 en L_1. Pour plus de simplicité, un itemset P est notée $I_1I_2. . . i_k$, où $i_1 > i_2 > ..... > i_k$,

(Nodesets de 2-itemsets):on prend deux items quelconque i_1 et i_2 ($I_1, i_2 \in L1 \wedge i_1 > I_2$) et nodeset i_2 est $\{(pre_1, C_1), (pre_2, C_2). . .,(Pre_n, c_n)\}$, respectivement. Le nodeset de 2-itemset i_1i_2, et on l'écrit Nodeset (I_1I_2), est un sous-ensemble de nodeset i_2, qui est définie comme suit :

$b \longrightarrow \{(4,4)\}$
$c \longrightarrow \{(1, 1), (5, 3)\}$
$e \longrightarrow \{(6, 3)\}$
$f \longrightarrow \{(2, 1), (8, 1), (9, 1)\}$
$a \longrightarrow \{(3, 1), (7, 1)\}$

Propriété 2**:** Soit P ($=p_1p_2$, $p_1, p_2 \in$ L1 et $p_1 > P_2$) un 2-itemset etson nodeset est $\{(pre_1{:}c_1), (pre_2, c_2),. . ., (Pre_m, c_m)\}$. Le support P est égal à $C_1 + C_2 +. . . + C_m$.

(Nodesets de k-itemsets):Soit $P = p_1p_2p_3. . .p_K$ est un itemset ($p_j \in L_1$ et $p_1 > p_2 > p_3 > ... > p_k$). Supposons que le nodeset de $P1 = p_1p_3. . .p_k$ est le nodeset P_1 et nodeset de $P_2 = P_2P_3. . .p_k$ est

NodesetP_2. L'ensemble de noeuds P, notée Nodeset P, est définie comme intersection de nodeset P_1 et P_2. Autrement dit,

Nodeset P = Nodeset P_1 ∩ Nodeset P_2.

Par exemple, nous savons que les Nodesets de b_f et c_f sont respectivement {(8, 1), (9, 1)} et {(2, 1), (8, 1)},. le nodeset de bcf est {(8, 1), (9, 1)} ∩{(2, 1), (8, 1)} = {(8, 1)}.

Propriété 3: on prend k-itemset P, son nodeset est {(pre_1: c_1), (pre_2, c_2),. . . (Pre_m, c_m)}, le support de P est égal à $C_1 + C_2 +. . . + C_m$.

Algorithme FIN : (la méthode proposée) :

L'algorithme FIN est présenté comme suit :

(1) Construire le POC-arbre et identifier tous les 1-itemsets fréquents;

(2) analyser le POC-arbre pour trouver tous 2-itemsets fréquentes et leurs Nodesets;

(3) fouiller tout k (> 2)-itemsets fréquents.

Pour faciliter le processus de l'exploitation, FIN emploie un arbre d'ensemble enuméré d'éléments (Rymon, 1992) pour représenter l'espace de recherche.

Étant donné un ensemble d'éléments I = {i_1, i_2 ,. . ., I_m} où $i_1<i_2 . . . <i_m$, un arbre d'ensemble énuméré peut être construit comme suit [8]:

Tout d'abord, la racine de l'arbre est crée. Deuxièmement, les m nœuds enfants de la racine de 1-itemsets sont créés et entigstrés, respectivement. Troisièmement, pour un noeud représentant itemset {$i_{js}i_{js-1}$ ... i_{j1}} et i_{js} enregistrement, les (m -j_s) nœuds enfants du nœud représentant itemsets {$i_{js+1}i_{js}i_{js-1}...i_{j1}$}, {$i_{js+2}i_{js}i_{js-1}...i_{j1}$}, {$I_m i_{js}i_{js-1}...i_{j1}$} et i_{js+1}, i_{js+2}, ..., i_m, respectivement sont créés. Enfin, l'arbre d'ensemble énumérer est construit par l'exécution de la troisième étape jusqu'à ce que tous les nœuds feuilles soient créés.

Le pseudo-code de l'algorithme FIN :

Ligne (1) initialise F, qui est utilisé pour stocker des motifs fréquents, en le définissant comme nulle.

Ligne (2) construit le POC-Tree et trouver F_1, l'ensemble de tous les 1-itemset fréquent, en appelant l'algorithme 1.

Ligne (3) initialiser F_2, qui est utilisé pour stocker les 2-itemsets fréquents, en le définissant comme nul.

Ligne (4) –(15) insérer tous les candidats 2-itemsetsfréquents en F_2, en scannant le POC-Tree avec la pré-ordrede la traversée.

Ligne (16) - (22) supprimer tous les rares 2-itemsets de F_2, comme la ligne (18) faire, et initialiser l'ensemble des Noeud de tous 2-itemsets fréquentes en les définissant comme nulle, que la ligne (20) faire.

Ligne (23) - (31) génèrent l'esmble des Noeud de tous 2-itemsets fréquentes en scannant le POC-Tree avec la pré-ordre de la traversée.

Ligne (33) - (36) génèrent tous les k-itemsets fréquents (k >= 3) en appelant procédure Construire Arbre Motif () pour générer tous les k-itemsets fréquents (k >= 3) a prolongé de chaque 2-motifs fréquents [8].

Algorithm 2: AlgorithmeFIN

Entrée : base de donnée du transaction et support minimaleε.

Sortie : F, ensemble de tous les elements fréquents**.**

F_1←Ø;

Appelé l'Algorithm 1 pour construire POC-tree ettrouverl'nsemble des 1-items frequents F_1,

F_2← Ø;

Parcourir la POC-tree avec le pre order traverse

N ← les noueds visités;

i_y← les elements enregistré en N;

Pour chaque ancêtre de N, Na, faire

i_x← elements enresigtré en Na;

Si $i_x i_y \in F_2$, Alors

$i_x i_y$.support←$i_x i_y$.support + N.account;

Sinon $i_x i_y$.support←N.account;

F_2←$F_2 \cup \{i_x i_y\}$; FinSi

Finpour

Pour chaque itemset, P in F_2 faire

Si P.support<ε x |DB|, Alors

F2 ←F2 - {P}; Sinon

P. Nodeset←Ø;

FinSi

FinPour

Scannez le POD-tree par la pré-commande traversal faire

Nd ← Node visitée en cours;

i_y←item registereé dans Nd;

Pour chaque ancêtre de N_d N_{da} faire

ix ←item registeré dans Nda;

Si $i_x i_y \in F_2$ Alors

i_xi_y.Nodeset←i_xi_y.Nodeset U Nd.N_info;

FinSI

FinPour

F← F U F1;

Pour chaque ItemSet fréquent,i_si_t, dans F2 faire

Créerla racine de l'arbre, R_{st}, et nommez le par i_si_t;

Construire_Modèle_Arbre(R_{st}, {i | i ϵ F_1, i > i_s}, Ø**)**;

FinPourRetourne F;

Procédure Construire_Modèle_Arbre(Nd, Cad_set,FIS_parent)

Nd.equivalent_items ←Ø

Nd.childnodes ←Ø

Next_Cad_ set ←Ø

Pour chaque i ∈ Cad_set faire

X ←Nd.itemset;

Y ←{i} U (X - X[1]);

P ← {i} U X;

P.Nodeset ←X.Nodeset ∩ Y.Nodeset;

Si P.support = X.support Alors

Nd.equivalent_items←Nd.equivalent_items U {i};

Sinon SI P.supportP≥|DB| x Ɛ Alors

Créer la Noued Nd_i;

Nd_i.label←i;

Nd_i.itemset← P;

Nd.childnodes← Nd.childnodes U {Nd_i};

Next_Cad_ set ←Next_Cad_ set U {i};

FinSi FinSi

FinPour

SI Nd.equivalent_items ≠ ϕ Alors

SS ← ensemble de tous les sous-ensembles de Nd.equivalent_items;

PSet ← {A | A = Nd.label U A' , A'ϵSS};

SI FIS_parent = Ø, then

FIT_Nd ←PSet;

Sinon FIT_Nd ←{P' | P' = P1 U P2, (P1 ≠ Ø ∧ P1 ϵ PSet) and (P2 ≠ Ø ∧ P2 ϵ FIS_parent};

FinSI

F← F U FIT_Nd;

FinSi

Si Nd.childnodes ≠ ØAlors

Pour chauqe $Nd_i \epsilon$ Nd.childnodes faire

Construire_Modèle_Arbre(Nd_i, {j | j ϵ Next_Cad_set, j _ i}, FIT_Nd**)**;

FinPor

Sinon retourne;

FinSI [8];

3.2.4 Algorithme PrePost+ (Deng et Lv, 2015) :

Au cours des dernières années, nous présentons un algorithme appelé PrePost (Deng et al, 2012) pour l'extraction des itemsets fréquents. Le rendement élevé de PrePost est obtenu par [9] :

- utilisant une structure de données nommée N-liste pour représenter les itemsets;
- l'adoption de la propriété de chemin unique de N-liste directement qui permet de découvrir les motifs fréquents sans générer itemsets candidats dans certains cas.

Les expériences de Deng et al. (2012) montrent que PrePost exécute vite que certains algorithmes d'extraction des itemset fréquents, y compris FP-Growth et FP-growth*.

Bien que PrePost adopte la propriété de chemin unique de N-list pour élaguer l'espace de recherche, il encourt toujours le problème de plus candidats car elle emploie l'approche de l'algorithme Apriori pour l'extraction des itemsets fréquents. PrePost+ utilise N-liste pour représenter les itemsets et découvrir les itemsets fréquents dans un itemset & N-list pour l'arbre de recherche. Pour éviter la répétition de recherche, il adopte également l'approche enfants-parents dans le but de réduire considérablement la taille de l'espace de recherche. Pour évaluer la performance de PrePost +, nous procédons à une étude approfondie de la performance pour comparer à trois state-of-the-art algorithmes : PrePost, FIN, et FP-growth* [9].

Les résultats expérimentaux montrent que PrePost+ est efficace et fonctionne beaucoup plus rapidement que les trois algorithmes comparés alors qu'il ne consomme un peu plus de mémoire que FP-growth*, la meilleur en termes de consommation de mémoire.

Principe de base :

Définition 1 : On prend une base de données de transactionDB et un seuil Ɛ, la Version succint de DB est une base de données générée par la suppression des items non fréquents de chaque transactionet de trierles items selon l'ordre décroissant de

support.La version succint de DB contient toutes les informations nécessaires lié aux motifs fréquents. Par conséquent, nous pouvons directement extraire les itemsets fréquents de la version succint de DB.

Définition 2 **:** Étant donné un DB succinct, PPC-arbre est une structure arborescente définies comme suit :

(1) Il se compose d'une racine libelée "null", et un ensemble d'items préfixe sous abre qui consiste comme les enfants de la racine.

(2) Chaque noeud dans le sous-arborescence se compose de cinq champs : le nom-item, nombre d'occurrence, la liste d'enfants, pré-commande, et post-commande. Le nom-item enregistre quel élément de ce noeud représente. Compteur enregistre le nombre de transactions présenté par la partie du chemin pour atteindre ce noeud. La liste d'enfants registre tous les enfants du noeud. Pré-commande est le rang de pré-ordre de la noeud. post-ordre est le rang post-ordre du noeud. Pour un noeud, son pré-commande est le numéro de séquence du noeud lors de la parcourt de l'arbre par pré-commande traversal et son post-commande est le numéro de séquence du noeud lorsque balayant l'arbre par post-ordre traversal [9].

Example01 **:** on prend la base de données de transaction, DB, représenté par les informations des deux colonnes de gauche du Tableau et Ɛ = 0,4.

ID	Items	l'ordre des items fréquents
1	**a, f, g**	**a, f**
2	**A, b, c, e**	**b, c, e, a**
3	**b, c, e,i**	**b, c, e**
4	**b, c, e, h**	**b, c, e**
5	**b, c, d, e, f**	**b, c, e, f**

Tableau3.7 : Table des transactions

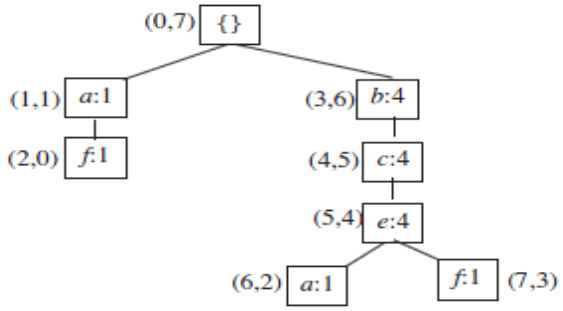


Figure 3.7 : PPC Arbre

Définition 3 : (PP-code). Pour chaque nœud N dans un PPC-arbre, nous appelons {(N.pré - ordre; N.poste - ordre) : compteur } le PP-code de N.

Définition 4 : (liste de N Items fréquents). Étant donné un PPC-arbre, la liste des NItems fréquents est une séquence de tous les enregistrements de PP code des nœuds dans le PPC-arbre. Les PP codes sont disposés dans un ordre croissant de leurs valeurs pré-ordre.

Définition 5 : (> Relation). Chaque deux items fréquents différéents i_1 et i_2, nous décrivons $i_1 > i_2$ (ou $i_2 > i_1$) si et seulement si i_1 est en avance sur i_2 dans DB.

Pour simplifier la description, Chaque itemset P est notée $I_1I_2 \ldots i_k$, où $i_1 > i_2 > \ldots > i_k$par convention.

Définition 6 : (N-liste de 2-itemset). On prend deux items fréquents quelconques i1 et i2, dont les N-listes sont $\{<(x_{11}, y_{11}): z_{11}>, <(x_{12},y_{12}): z_{12}>,..., <(x_{1m}, Y_{1m}): Z_{1m}>\}$ et $\{< (x_{21}, y_{21}): z_{21}>, <(x_{22}, y_{22}): z_{22}>,. . ., <(x_{2n}, y_{2n}): Z_{2N}>\}$ respectivement. La N-liste de 2-itemset I_1I_2 est une séquence de PP codes triés en pré-Odre par ordre croissant, ils ont générées en intersection entre les N-listes de i_1 et i_2, en suivant la règle ci-dessus :

Tout d'abord, pour tout $<(X_{1P}, y_{1p}): z_{1p}>\epsilon$ N-liste de i_1 ($1\leq p\leq m$) et $< (x_{2q}, y_{2q}): z_{2q}>\epsilon$ N-liste de i_2 ($1\leq q\leq n$), si $< (X_{1P}, y_{1p}): z_{1p}>$ est un ancêtre de $< (x_{2q}, y_{2q}): z_{2q}>$, alors $< (X_{1P}, y_{1p}): z_{2q}>$ est ajouté à N-liste de I_1I_2. Après, nous obtenons N-liste initiale de I_1I_2. Ensuite, en vérifier N-liste initiale de I_1I_2. en Fusionne les noeuds avec la forme de

$<(x_{1b}, Y_{1B}): z_{1b}><(x_{1b}, Y_{1B}): z_{2b}>... <(x_{1b}, Y_{1B}): z_{rb}>$ pour obtenir un nouveau noeud

$<(x_{1b}, Y_{1B}): (Z_{1B} + 2b ... + zrb)>$.

Définition 7 : (N-liste de k-itemset). Soit $P = i_x i\ i_1\ i_2. . .i_{(k-2)}$ un itemset ($K\geq 3$ et chaque élément de P est un item fréquent), la N-liste des $P1 = i_x\ I_1\ I_2. . .i_{(k-2)}$ va être

$\{<(x_{11}, y_{11}): z_{11}>, < (x_{12}, y_{12}): z_{12}>,. . ., < (x_{1m}, Y_{1m}): Z_{1M}>\}$, et la N-liste de $P2 = i_y\, i_1\, i_2 \ldots i_{(k-2)}$ va être $\{<(x_{21}, y_{21}): z_{21}>, <(x_{22},y_{22}):z_{22}> ...<(x_{2n},y_{2n}): z_{2n}>$

N-liste P est une séquence de PP codes selon la pré-ordre par ordre croissant et il est généré en coupant les N-listes de P_1 et P_2, d'après la règle ci-dessous :

Tout d'abord, pour tout $< (X_{1P}, y_{1p}): z_{1p}>\epsilon$ N-liste de P_1 ($1 \leq p \leq m$) et $< (x_{2q}, y_{2q}): z_{2q}>\epsilon$ N-liste de P_2 ($1\leq q\leq n$), si $< (X_{1P}, y_{1p}): z_{1p}>$ est un ancêtre de $< (x_{2q}, y_{2q}): z_{2q}>$, alors $< (X_{1P}, y_{1p}): z_{2q}>$ est ajouté à N-list $i_x i\ I_1\ I_2. . .i_{(k-2)}$. Après cela, nous obtenons une N-liste initiale de P [9].

Deuxièmement, nous vérifions N-liste initiale de P à nouveau et en fusionne les nœuds avec la forme de $< (x_{1b}, Y_{1B}): z_{1b}><(x1b, Y1B): z_{2b}> ... <(x_{1b}, Y_{1B}): z_{rb}>$ pour obtenir un nouveau noeud $< (x_{1b}, Y_{1B}): (Z_{1B} + {}_{2b} ... + z_{rb}) >$.

Propriété01 : on'a une N-list de k-itemset $P=i_1i_2...i_k$, qui est notée $\{< (x_1, y_1): Z_1>,$

$<(x_2, y_2):z_2>...,<(x_m, y_m): z_m>\}$, le support correspond de cet itemset P est $z_1+z_2+...+ Zm$.

La méthode proposée de Prepost+ :

Cette algorithme ressemble à l'algorithme **PrePost**, son fonctionnement et décrit comme suit :

- Construisez PPC-arbre et identifier tous les 1-itemsets fréquents;
- à partir de PPC arbre, construisez N-liste de chaque 1-itemset fréquente;
- analyser le PPC-arbre pour trouver tous les 2-itemsets fréquents;
- extraire tout les k-itemsets fréquents (k>2). La différence entre PrePost et PrePost+ est que PrePost adopte l'équivalence superset que la stratégie d'élagage tandis que PrePost + adopte la propriété de chemin unique de N-liste comme stratégie d'élagage.

Pour faciliter le processus d'extraction, PrePost + adopte une set-énumération arbre (Burdick et al., 2005) pour représenter la recherche espace. Étant donné un ensemble d'éléments $I = \{i_1, i_2,. . ., I_m\}$ où $i_1< i_2< . . . < i_m$, une arbre est une set-énumération peut être construit comme suit : [9]

Tout d'abord, la racine de l'arbre est créé. Deuxièmement, les m nœuds enfants de la racine l'enregistrement et représentant m 1-itemsets sont créés, respectivement. En troisième lieu, pour un noeud représentant itemset $\{i_{js}\ i_{-1}. . .i_{j1}\}$ etl'enregistrement i_{js},

les (m - js) nœuds enfants qui représentent les itemsets $\{i_{js+1}i_{js}i_{-1}. . .i_{j1}\}$, $\{i_{js+2}i_{js}i_{-1}. . .i_{j1}\}$,. . ., $\{I_m i i_{js-1}. . .i_{j1}\}$ et i_{js+1}, i_{js+2} ,. . ., I_m enregistrement respectivement sont créés.

Enfin, l'arbre de set-énumération est construit par l'exécution de la troisième étape à plusieurs reprises jusqu'à ce que tous les noeuds de feuille soient créés.

Propriété 02 **:** On prend un itemset P et un élément i (Ï P), si le support de P est égal à l'appui de PU{i}, pour tout itemset A (A∩P = ∅Λ i Ï A),le support de A [P est égal à l'aide de A U P U {i}.

Algorithm 1: PrePost+ Algorithm

Entrée: Une base de transaction DB et un support minimum Ɛ.

Sortie: F, l'ensemble de tous les itemset fréquents.

(1) parcourir DB pour obtenir F_1, l'ensemble de tous les1-itemset fréquentes, etconstruire le PPC-tree;

(2) parcourir le PPC-arbre pour générer les N-listes de 1-itemset fréquentes;

(3) parcourir le PPC-arbre pour obtenir F_2, l'ensemble de tous 2-itemset fréquents

(4) Pour chaque $i_s i_t$ (ϵ F_2), générer sa N-liste en appelantla procédure **NL_intersection** (i_s.AND-list, i_t.N-liste);

(5) F ←F_1;

(6) Pour chaque itemset fréquent, noté $i_x i_y$, dans F2 faire

(7) Créer la racine del'arbre, FPT_{xy}, et libelléepar $i_x i_y$;

(8) **Building_Pattern_Tree** (FPT_{xy}, {i | i ϵ F_1, i >i_x}, ∅);

(9) Retour F;

Procédure Building_Pattern_Tree (Nd, Cad_items, Parent_fit)

(1) Nd.equivalent_items ←∅;

(2) Nd.childnodes ←∅;

(3) Next_Cad_items ←∅;

(4) Pour chaque i ϵ Cad_items faire

(5) P1 ←Nd.itemset;

(6) P2 ←{i} U (P_1 - P_1 [1])

(7) P ←{i} U P_1;

(8) P.N-list ←**NL_intersection** (P_1, P_2);

(9) Si P.support = P1support Alors

(10) Nd.equivalent_items ←Nd.equivalent_items U {i};

(11) Sinon si P.support ≥ |DB| xƐ,

(12) Créer noeud Nd_i;

(13) Nd_i.label← i;

(14) Nd_i.itemset ←P;

(15) Nd.childnodes ←Nd.childnodes U {Nd_i};

(16) Next_Cad_items ← Next_Cad_items U {i};

(17) Si Nd.equivalent_items ≠ ∅Alors

(18) sous-ensembles ← l'ensemble de tous les sous-ensembles de Nd.equivalent_items;

(19) Cand_itemsets ← {A | A = Nd.label U A', A'ϵ SS};

(20) Si Parent_fit = ∅Alors

(21) Nd_fit ←Cand_itemsets;

(22) Sinon

(23) Nd_fit ← {P' | P' = P_1U P_2, (P1≠∅∧ P_1ϵ Cand_itemsets)et (P_2≠∅∧ P_2ϵ Parent_fit};

(24) F ← F U Nd_fit;

(25) Si Nd.childnodes ≠ƐAlors

(26) Pour chaque nœud enfant, Nd_i, faire

(27) **Building_Pattern_Tree** (Nd_i, {j | j ϵ Next_Cad_items, j>i}, Nd_fit);

(28) Sinon Retour.

Fonction **NL_intersection** (NL_1, NL_2)

Entrée: NL_1 = {((x_{11}, y_{11}): z_{11}), ((x_{12}, y_{12}): z_{12}),. . ., ((x_{1m}, y_{2n}): z_{2n})}, qui sont respectivement les N-listes de P1 = i_u i_1 i_2. . .$i_{(k-2)}$ et P2 =i_v i_1 i_2. . .$i_{(k-2)}$ (i_u> i_v).

Sortie: NL_3est N-liste de P_3 = $i_u i_v$ $I_1 I_2$. . .$i_{(k-2)}$.

(1) i ←1;

(2) j←1;

(3), tant que (i ≤ m etj ≤ n)

(4) si (x_{1i}< x_{2j}) Alors

(5) si (Y_{1i}>Y_{2J}) Alors

(6) **Insérez**((x_{1i}, Y_{1i}): z_{2j}) en NL_3;

(7) j++;

(8) else i++;

(9) else j++;

// Fusionner les éléments avec la même (pré-code, post-code) en paire sur NL_3.

(10) ptr_1← NL_3.first_element; // Le premier élément de NL_3;

(11) ptr_2 ←ptr_1.next_element ; // L'élément suivant de ptr_1;

(12), tant que ptr_1n'est pas le dernier élément de NL_3faire

(13) si ptr_1.pre-code = ptr_2.pre-code et ptr_1.post-code= ptr_2.post-code

(14) ptr_1.compteur ←ptr_1.compteur + ptr_2.compteur;

(15) supprimer ptr_2 de NL_3;

(16) ptr_2 ←ptr_1.next_element de;

(17) Sinon

(18) ptr_1←ptr_1;

(19) ptr_2←ptr_1.next_element;

(20) retour NL_3;

3.3 Génration des Régles d'association :

La génération des règles d'association est une tâche fondamentale de fouille de données. Cependant, en fonction du choix des paramètres (la confiance minimale et le support minimum), les algorithmes actuels peut devenir très lent et générer un très grande quantité de résultats ou de générer trop peu de résultats, en omettant précieux information. Ceci est un problème grave, car les utilisateurs ayant des ressources limités pour l'analyse des résultats donc sont souvent intéressés parla découverte d'un certain nombre de résultats, et le réglage des paramètres prend plus du temps.

Certains algorithmes utilisent l'approche classique qui consistent de passer par deux étapes, basé sur l'extraction des itemset fréquents, Ainsi, il y'a d'autres algorithmes qui utilisent nouvelle approche pour la génération des règles d'association qui s'appelle **l'expansion des règles**, cette approches est basé sur la recherche des règles avec un seul item à gauche et un seul item à droite.

Pour résoudre ce problème, nous proposons des algorithmes de chaque approche pour générer les régles d'association, parmi ces algorithms en cite :

- **Génération des régles d'association par approches de deux étapes :**
  - **L'algorithme Apriori pour génération des règles d'association.**
- **Génération des régles d'association par approche d'expansion de règles :**
  - **Génération des Régles d'Association de Top-K**
  - **Génération des Régles d'Association de Top-K Non-Redundantes**

3.3.1 Génération des régles d'association par deux étapes :

3.3.1.1 *L'algorithme Apriori pour génération des règles d'association :*

Soit I = $\{i_1; i_2; :::; i_m\}$ un ensemble d'éléments, appelé items. Soit D un ensemble de transactions, où chaque transaction T est un ensemble d'éléments tels que T⊆I. Chaque transaction a un identificateur unique, appelé son TID. Nous disons qu'une transaction T contient X, un ensemble des items dans I, si X ⊆ T. Une règle d'association est une implication de la forme X⇒Y, où X⊂I, Y⊂I, et X ∩ Y =∅. La règle X⇒Y détient dans l'ensemble de transactions D avec un confiance c si c% des transactions D qui contiennent X et Y également. La règle X⇒Y a un support s dans la transaction D définie si s% des transactions en D contient XUY.

Étant donné un ensemble de transactions D, le problème de génération des règles d'association est de générer toutes les règles d'association qui ont le support et la confiance supérieure au support minimum définie par l'utilisateur (appelé minsup) et le minimum confiance (appelé minconf) [15].

Notre discussion est neutre par rapport à la représentation de D. Par exemple, D peut être un fichier de données, une table relationnelle, ou le résultat d'une expression relationnelle.

Le problème de découvrir de toutes les règles d'association peut être décomposée en deux sous-problèmes :

1. Trouver tous les ensembles d'éléments (itemsets) qui ont un support minimum de transaction.Le supportd'un itemset est le nombre de transactions qui contiennent cet élément (item). Itemsets avec un support minimum sont appelés itemsets.

2. Utilisez les itemset fréquents pour générer les règles souhaitées. L'idée générale est que si, par exemple, ABCD et AB sont des itemset fréquents, alors nous pouvons déterminer si la règle AB⇒CD valide en calculant le rapport conf = support (ABCD) / support (AB). Si conf≥minconf, la règle est valide. (La règle va sûrement avoir un support minimum parce que ABCD est fréquents.) [15]

Découvrir les itemset fréquents :

Les Algorithmes de génération des itemsets fréquents font plusieurs passages sur les données. Dans le premier passage, on compte le support des éléments individuels et de déterminer lequel d'entre eux sont fréquent, à savoir a un support minimum. Dans chaque passage suivant, nous commençons par une distribution d'ensemble de itemsets jugées fréquentes dans le passage précédent. Nous utilisons cet ensemble de distribution pour générer de nouvelles itemsets potentiellement fréquentes, appelés itemsets candidats, et compter le support réel pour ces itemsets candidats au cours de la passe sur les données. A la fin de passage, on détermine lequel des itemsets candidats sont en fait souvent, et ils devenir la distribution pour le passage suivant. Ce processus se poursuit jusqu'à ce qu'aucune nouvelle itemsets fréquents ne soit trouvés.

Les algorithmes Apriori et AprioriTid génére les itemsets candidats comptés dans une passe en utilisant uniquement les itemsets fréquents trouvés dans le passage précédent sans considérer les transactions dans la base de données. L'intuition de base est que tout sous-ensemble d'un itemset fréquent doit être fréquente. Par conséquent, les itemsets candidats ayant des éléments de k peuvent être générées en se joignant à motifs fréquents comportant des éléments k-1 et en supprimant celles qui contenir un sous-ensemble qui ne sont pas fréquents. Cette procédure se traduit par la génération d'un plus grand petit nombre d'itemsets candidats [15].

Nous appelons que le nombre d'éléments dans un itemset est la taille, et nous appelons un itemset de taille k une k-itemset. Items au sein d'un itemset sont conservés dans l'ordre lexicographique. Nous utilisons la notation

k-itemset	**An itemset having k items.**
$\mathbf{L_k}$	Ensemble de k-itemsets fréquents (ceux avec le support minimum). Chaque membre de cet ensemble comporte deux extrémités : i) itemset et ii) compteur de support.
$\mathbf{C_k}$	Ensemble de candidats de k-itemsets (les itemsets potentiellement fréquentes). Chaque membre de cet ensemble a deux domaines : i) itemset et ii) compteur de support.

C'$_k$	Ensemble de candidats de k-itemsets quand TIDs de génération des transactions sont conservés aux candidats.

Tableau3.8 : Notation for Algorithms

Le premier passage de l'algorithme compte simplement les occurrences des items pour déterminer les fréquents 1-itemsets. Un deuxième passagese compose de deux phases. Tout d'abord, les motifs fréquents L_{k-1} trouvés dans le (k-1) ième passe sont utilisés pour générer des ensembles d'éléments candidat C_k, en utilisant la fonction de génération des candidats décrite ci-dessous. Ensuite, la base de données est numérisée et le support des candidats en C_k est compté. Pour comptage rapide, nous devons Déterminer les candidats C_k qui sont contenues dans un base de transaction T.

L1 := 1-itemsets fréquents;

k := 2; // k présente le nombre de passes

Tant que ($L_{k-1} \neq \varnothing$) faire

C_k := Nouveau candidat de taille k génére à partir de L_{k-1};

Pour toutes les transactions T ϵ D faire

Incrémenter le compteur de tous les candidats dans C_k qui contenant dans T.

FinPour

L_k := Tous les candidats dans C_k avec support minimum.

k := k + 1;

Fin

RécupérerU$_k$ L_k;

Apriori Algorithm :

Fonction de génération des candidats

Etan donne L_{k-1}, l'ensemble de tous les k-1-itemset fréquents, l'algorithme retourne l'ensemble de tous les itemset fréquents, la fonction est présentée comme suit .

Premièrement,

Nous mettons L_{k-1} avec L_{k-1}

Ajouter dans C_k

Selectionner p.item$_1$, p.item$_2$, ..., p.item$_{k-1}$, q.item$_{k-1}$

Depuis L_{k-1} p, L_{k-1} q

OU p.item$_1$ = q.item$_1$, : : :, p.item$_{k-2}$ = q.item$_{k-2}$, p.item$_{k-1}$< q.item$_{k-1}$;

Deuxièment, on supprimetous les itemset c ϵ C_k telque (k-1)-sous-ensemble de itemset c sont pas dans L_{k-1}:

Pour tous les itemset c ϵ $C_k$faire

For all (k-1)-sous ensemble itemset s de c faire

If ($s \notin L_{k-1}$) Alors

Supprimer c de C_k;

Découvrir des Règles d'associations :

Pour générer les règles, pour chaque itemset l fréquente, nous cherchons tous les sous ensembles non vide de l. Pour chaque sous-ensemble a, la règle en sortie est de la forme a⇒(l-a) si le rapport du support (l) sur le support(a) est au moins minconf. Nous considérons tous les sous-ensembles de l pour générer les règles avec de multiples conséquences. Étant donné que les motifs fréquents sont stockés dans des tables de hachage [15],

Algorithme Apriori pour Génération des régles

Pour tous les itemset fréquents l_k, $k \geq 2$ faire

Call Genrules (l_k, l_k); // Genrules génère tous les règles valide â ⇒ (l_k - â), pour tous â⊂a_m

Procedure Genrules(l_k: k-itemset fréquents, a_m: m-itemset fréquent)

A: = {(m-1)-itemsets a_{m-1} | $a_{m-1} \subset a_m$};

Pour tous $a_{m-1} \in$ A faire

Début

conf := support(l_k)/support(a_{m-1});

Si (conf ≥ minconf) Alors

Début

La règle en sortie $a_{m-1} \Rightarrow (l_k - a_{m-1})$, avec confiance = conf and

Support = support(l_k);

SI (m - 1 > 1) Alors

Call Genrules (l_k, a_{m-1}); //

Fin

Fin

3.3.2 Génération des régles d'association par approche d'expansion de règles **:**

3.3.2.1*Génération des Régles d'Association de Top-K* **:**

La génération des règles d'Association consiste à découvrir les associations entre les éléments de transactions. Il est l'une des tâches les plus importantes d'exploration de données. Il a été intégré dans de nombreux logiciels d'exploration de données commerciales et a de larges applications dans plusieurs domaines.

Le problème de l'extraction de règles d'association est énoncée comme suit.

Soit I = {a_1, a_2, ... a_n} un ensemble fini d'éléments. Une base de données de transaction est une ensemble de transactions T = {t_1, t_2 ... t_m} où chaque t_j de transaction ⊆ I ($1 \leq j \leq m$) représente un ensemble d'articles achetés par un client à un moment donné.

Un itemset est un ensemble d'éléments X ⊆ I. Le support d'un itemset X est notée sup(X) est défini comme étant le nombre de transactions qui contiennent X. Une règle d'association X → Y est une relation entre deux itemsets X, Y tels que X, Y ⊆ Iet X∩Y = Ø. Le support d'une règle X → Y est défini comme [10]:

$$\sup (X \rightarrow Y) = \sup (X \cup Y) / |T|.$$

La confiance d'une règle X → Y est défini comme :

$$\text{conf} (X \rightarrow Y) = \sup (X \cup Y) / \sup (X).$$

Le problème de génération des règles d'association est de trouver toutes les règles d'association dans une base de données pour un support ayant au moins un seuil minsup défini par l'utilisateur et une confiance pas moins à un seuil minconf défini par l'utilisateur. Par exemple, la figure3.9 montre une transaction base de données (à gauche) et les règles d'association trouvées pour minsup = 0,5 et minconf = 0,5 (droite). Génération des règles d 'Associations se fait en deux étapes.

Étape 1 : est de découvrir tous les fréquents itemsets dans la base de données (itemsets apparaissant dans au moins minsup × | T | transactions).

Étape 2 : est de générer des règles d'association à l'aide des motifs fréquents trouvés dans l'étape 1.

Pour chaque itemset fréquent X, des paires de itemsets fréquents P et Q = X - P est Sélectionnés pour générer une règle de la forme P → Q. Pour chaque règle P → Q,

Si sup(P → Q) ≥ minsup et conf (P → Q) ≥ minconf, la règle est sortie.

Cette idée d'extraction de top-k règles d'association présentée dans notre travail est analogue à l'idée de extraction de top-k itemsets et top-k motifs séquentiels dans le domaine d'extraction de motifs fréquents. Notez que bien que certains auteurs aient déjà utilisé le terme « règles d'association top-k », ne pas utiliser la définition standard d'un règle d'association. KORD ne trouve que des règles avec un seul élément dans le conséquent [10].

Pour atteindre cet objectif, une question est de savoir comment combiner le concept de top-k motif d'exploitation avec les règles d'association ? Pour l'extraction de règles d'association, deux seuils sont utilisés.

Mais, dans la pratique minsup est beaucoup plus difficile à mettre à minconf parce que minsup dépend des caractéristiques de base de données qui sont inconnues à la plupart des utilisateurs, alors que minconf représente la confiance minimale que les utilisateurs veulent

pour trouver des règles et généralement est facile à déterminer. Pour cette raison, nous définissons "top-k" sur le support plutôt que la confiance.

Par conséquent, l'objectif de ce travail est d'exploiter les règles top-k avec le soutien le plus élevé que répondre à une confiance désiré. A noter cependant que l'algorithme présenté peut être adapté à d'autres mesures d'intérêt. Mais nous ne discutons pas ici en raison de la limitation de l'espace [10].

ID	Transactions
T_1	{a, b, c, e, f, g}
T_2	{a, b, c, d, e, f}
T_3	{a, b, e, f}
T_4	{b, f, g}

ID	Règles	Support	Confiance
R1	{a}→ {b}	0.75	1
R2	{a}→ {c, e, f}	0.5	0.6
R3	{a, b}→ {e, f}	0.75	1
...			

Tableau3.9:(a) base de transations et (b) quelques règles d'association trouvées

Extracter les règles d'association top-k est difficile en raison l'algorithme Top-k ne peut pas compter sur les deux seuils pour élaguer l'espace de recherche. Dans le pire cas, l'algorithme top-k aurait pour générer toutes les règles pour trouver les règles top-k, s'il y a un nombre d d'éléments dans une base de données, on peut y avoir jusqu'à 3^d-2^d+1 règles à prendre en considération. Deuxièmement, l'extrraction top-k règle d'association est difficile parce que les deux étapes de processus ne seront plus utilisées à la fois. La raison est que l'étape 1 devrait être modifié pour que les itemsets fréquents commence par minsup = 0 à veiller à ce que toutes les règles top-k peut être généré à l'étape 2. Ensuite, l'étape 2 devrait être modifié mesure de trouver le règles top-k en générant des règles et en gardant les règles top-k. Cependant, cette approche poserait un problème de performance énorme, car aucune taille de l'espace de recherche n'est pas déterminée, Ainsi, à l'étape 1 est trop plus coûteux que l'étape 2. Par conséquent, un défi pour la définition d'un algorithme d'extraction de règles d'association top-k est de définir une approche efficace des règles d'association qui ne reposent pas sur le processus en deux étapes [10].

Définition 1: Le problème de l'extraction top-k des règle d'association est de découvrir un ensemble L contenant des règles de k dans T telle que pour chaque règle

r ϵL|conf (r) $\geq$ minconf, il n'existe pas une règle s$\notin$L|conf(s)$\geq$minconf $\wedge$ sup (s)> sup (r).

Définition 2: Une règle X $\rightarrow$ Y est de taille p * q si | X | = P et | Y | = Q. Par exemple, la taille de {a} $\rightarrow$ {e, f} est 1 * 2. De plus, nous disons qu'une règle de taille p * q est supérieure à une règle de taille r * s si p> r et q $\geq$ s, ou si p $\geq$ r et q> s.

Définition 3: Une règle d'association r est fréquente si sup (r) $\geq$ minsup.

Définition 4: Une règle d'association r est valide si sup (r) ≥ minsup et conf (r) ≥ minconf.
Définition 5: L'ensemble tid d'un itemset X est définies tid(X) = {t | t ϵ T ∧ X ⊆ t}. Par Exemple, tids ({a, b}) pour la base de données de transaction de la figure 1 (a) est
{t1, t2, t3}.
Définition 6**:** L'ensemble tid d'une règle d'association X → Y est notée tids (X → Y) et défini comme tids (X∪Y). Le support et la confiance de X → Y peuvent être exprimés en termes d'ensembles tid que:
sup (X → Y) = | tids (X∪Y) | / | T |,
CONF (X → Y) = | tids (X∪Y) | / | Tids (X) |.
Propriété 1. Pour toute règle X → Y, tids (X → Y) ⊆ tids (X) ∩ tids (Y).
L'algorithme TOP-K-RULES :

L'algorithme TopKRules prend en entrée une base de données de transaction, un nombre k de règles que l'utilisateur veut découvrir, et le seuil de minconf.
L'idée principale de l'algorithme est le suivant. il définit d'abord une variable minsup interne à 0. Ensuite, l'algorithme commence à rechercher les règles. Dès qu'une règle est trouvée, il va ajouter à une liste de règles L guidé par le support. La liste est utilisée pour maintenir les règles top-k trouvées jusqu'à présent. Une fois les k règles valides sont trouvés, le variable minsup interne est portée à l'appui de la règle avec le support le plus bas en L. l'augmentation de la valeur de minsup est utilisé pour élaguer l'espace de recherche lors de la recherche pour plus de règles [10].
Par la suite, chaque fois qu'une règle valide est trouvée, la règle est insérée dans L, L'algorithme continue à chercher d'autres règles jusqu'à ce qu'aucune règlese trouvent, ce qui signifie qu'il a trouvé les règles top-k.
TopK Rules ne repose pas sur l'approche en deux étapes classiques pour générer des règles, car il ne serait pas efficace en tant que top-k algorithme (comme indiqué dans l'introduction). La stratégie utilisée par TopK Rules consiste à générer les règles contenant un seul élément dans l'antécédent et un seul élément dans le conséquent.
Ensuite, chaque règle est récursive cultivé en ajoutant des éléments à l'antécédent ou conséquent. Pour sélectionner les éléments qui sont ajoutés à une règle pour le cultiver, TopK Rules scanne les transactions contenant la règle pour trouver des éléments simples qui pourraient élargir sa gauche ou à droite partie. Nous appelons les deux processus pour étendre les règles en TopKRules l'expansion gauche et l'expansion droite. Ces processus sont appliqués de manière récursive pour explorer les règles d'association [10].

Une autre idée incorporée dans TopK Rules est d'essayée de générer le plus premier prometteur des règles. En effet, si les règles avec un support élevé se trouvent plus tôt, TopKRules peutaugmenter sa variable minsup interne plus rapide pour élaguer l'espace de recherche.

Pour effectuer cela, TopKRules utilise une variable R interne pour stocker toutes les règles qui peuvent être étendus à avoir une chance de trouver des règles plus valides. TopKRules utilise cet ensemble pour déterminer la des règles qui sont les plus susceptibles de produire des règles valides avec un soutien de haut pour augmenter minsup plus rapidement et tailler une plus grande partie de l'espace de recherche [10].

L'algorithme TOP-K Rules

TOPKRULES (T, k, minconf) R: = Ø. L: = Ø. minsup: = 0.

1. Analyser la base de données T une fois pour enregistrer tidset de chaque item.
2. Pour chaque paire d'éléments i, j tels que :

|tids(i)| X |T|≥minsup et |tids(j) | X |T| ≥ minsup.

3. sup ({i} → {j}): = | tids (i) ∩ tids (j) | / | T |.
4. sup ({j} → {i}): = | tids (i) ∩ tids (j) | / | T |.
5. conf ({i} → {j}): = | tids (i) ∩ tids (j) | / | Tids (i) |.
6. conf ({j} → {i}): = | tids (i) ∩ tids (j) | / | Tids (j) |.
7. Si sup ({i} → {j}) ≥ minsup ALORS
8. SI conf ({i} → {j}) ≥ minconf ALORS **SAUVEGARDER** ({i} → {j}, L, k, minsup).
9. SI conf ({j} → {i}) ≥ minconf ALORS **SAUVEGARDER** ({j} → {i}, L, k, minsup).
10. Changer l'état expandLR de {i} → {j} à Vrai.
11. Changer l'état expandLR de {j} → {i} à Vrai.
12. R: = R ∪ {{i} → {j}, {j} → {i}}.
13. FinSI
14. FINPour
15. Tant que∃r ∈ R et sup (r) ≥ minsup Faire
16. Sélectionnez la règle ayant le support le plus élevé en R
17. SI rule.expandLR = vrai ALORS
18. **EXPAND-L** (règle, L, R, k, minsup, minconf).
19. **EXPAND-R** (règle, L, R, k, minsup, minconf).
20. Sinon**EXPAND-R** (règle, L, R, k, minsup, minconf).
21. RETIRER la règle r.
22. Retirer toutes les règles contenant dans R, r ∈ R | sup (r) <minsup.

23. FIN tant que

Procédure SAUVEGARDER(r, R, k, minsup)

1. L: = L∪ {r}.

2. SI | L | ≥ k ALORS

3. SI sup (r)> minsup ALORS

4. Tant que | L | > K et ∃ s ∈ L | sup (s) = minsup, ENLEVER s de L.

5. FIN SI

6. Modifier minsup vers le support le plus bas de règles en L.

7. FIN SI

procédure EXPAND-R (I → J, L, R, k, minsup, minconf)

1. Pour chaque tid ∈ tids (I∩J), parcourir la transaction tid. Pour chaque item c ∈ I figurant dans la transaction tid qui est lexicalement plus grand que tous les éléments de J, enregistrer tid dans un tids variable (I → J∪ {c}).

2. Pour chaque item c où | tids (I → J∪ {c}) | / | T | ≥ minsup:

3. Modifier le drapeau expandLR de I → J∪ {c} à fausse.

4. R: = R ∪ {I → J∪ {c}}.

5. SI | tids (I → J∪ {c}) | / | Tids (I) | ≥ minconf ALORS

SAUVEGARDER (I → J∪ {c}, L, k, minsup).

6. FIN Pour

Procédure EXPAND-L (I → J, L, R, k, minsup, minconf)

1. Pour chaque tid ∈ tids (I∩J), parcourir la transaction tid. Pour chaque item c ∈ J apparaissant dans la transaction tid qui est lexicalement plus grand que tous les
éléments de I, enregistrer tid dans un tids variable (I∪ {c} → J)

2. Pour chaque élément c où | tids (I∪ {c} → J) | / | T | ≥ minsup:

3. Modifier le drapeau expandLR de I∪ {c} → J à vrai.

4. tids (I∪ {c}) | = Tids (I) ∩ tids (c).

5. **SAUVEGARDER**(I∪ {c} → J, L, k, minsup).

6. SI | tids (I∪ {c} → J) | / | Tids (I∪ {c}) | ≥ minconf ALORS R: = R ∪ {I∪ {c} → J}.

7. FIN

3.3.2.2 *Génération des Régles d'Association de Top-K Non-Redundantes*

Malgré beaucoup de recherches ont été effectuées sur l'extraction de règles d'association, une importante question qui a été négligé est la façon dont les utilisateurs doivent choisir le minsup et minconf de seuils pour générer une quantité souhaitée de règles. Ceci est une

importante problème parce que les utilisateurs de pratique ont des ressources limitées (temps et d'espace de stockage) pour l'analyse des résultats et donc sont souvent intéressés à découvrir une certaine quantité des règles, et peaufiner les paramètres est temps. En fonction du choix de les seuils, les algorithmes actuels peuvent devenir très lent et générer un très grande quantité de résultats ou de générer aucun ou trop peu de résultats, en omettant précieuxinformation. Pour résoudre ce problème, il a été proposé de remplacer la tâche d'associationl'exploitation minière de la règle avec la tâche d'association top-k règles minière, où k est le nombre derègles d'association se trouvent, et est défini par l'utilisateur [11].

Cependant, un problème majeur reste avec ces algorithmes. les règles d'association top-k contiennent souvent une grande proportion de règles redondantes. à savoir les règles qui fournissent des informations qui est redondant à l'utilisateur [11].

Nous avons trouvé que jusqu'à 83% des règles d'association Top-k sont redondantes pour les ensembles de données couramment utilisés. Cela signifie que l'utilisateur doit analyser une grande proportion de règles redondantes.

ID	Transactions
T1	$\{a, b, c, e, f, g\}$
T2	$\{a, b, c, d, e, f\}$
T3	$\{a, b, e, f\}$
T4	$\{b, f, g\}$

ID	Transactions	Support	Confiance
R1	$\{a\} \rightarrow \{e, f\}$	0.75	1
R2	$\{a\} \rightarrow \{c, e, f\}$	0.5	0.6
R3	$\{a, b\} \rightarrow \{e, f\}$	0.75	1
R4	$\{a\} \rightarrow \{c, f\}$	0.5	0.6

Tableau3.10 :(a) base de donnée de transaction and (b) quelque régles d'association trouvés.

Le problème de la redondance dans la génération des règles d'association a été largement étudié et diverses définitions de redondance ont été proposées. Cependant, il reste un défi ouvert à combiner l'idée de la génération d'un ensemble de règles non redondantes avec l'idée de génération d'un top-k ee règles d'association, de proposer un algorithme efficace pour la génération de top-k non redondants de règles d'association.

L'avantage d'un tel algorithme serait de présenter un petit ensemble de k règles non redondantes à l'utilisateur. Cependant, la conception d'un algorithme pour extraire ces règles est difficile. La raison est que l'élimination de la redondance ne peut être effectuée comme une étape de post-traitement après génération des règles d'association top-k, car il se traduirait par moins de règles k. Le processus d'élimination de la redondance a par conséquent intégrés dans le processus d'extraction [11].

Découvrir des régles d'association Non-Redundantes :

Le problème de la redondance a été largement étudié dans la génération des règles l'association. Les chercheurs ont proposés de l'extraction de plusieurs ensembles de règles d'association non redondants comme le base générique, la base informative, la base informative et générique, la Base générique minimale [9] et Condition minimaledes Règles et résulatat Maximale. Ces ensembles de règles peuvent être comparées en fonction de plusieurs critères tels que leur compacité, la possibilité de récupérer des règles redondantes avec leurs propriétés (support et confiance), et de leur signification pour les utilisateurs.

La raison est que les critères les plus importants dans les règles d'association de Top-k sont la signification des règles pour les utilisateurs, car il définit des règles non redondantes comme des règles avec un antécédent minimale et un résultat maximuale. En d'autres termes, MCMR sont les règles qui permettent de déduire la montant maximal de l'information basée sur la quantité minimum d'informations. C'est ont fait valoir que ces règles sont les plus significatives pour plusieurs tâches. Dans le contexte d'extraction des règles d'association de Top-k, d'autres critères tels que la compacité et de la recouvrabilité des règles redondantes ne sont pas pertinents parce que l'objectif d'un algorithme top-k est de présenter un petit ensemble de k règles significatives pour l'utilisateur. MCMR sont définis en fonction de ce qui suit définition de redondance [11].

Un règle d'association $r_a : X \rightarrow Y$ est redondant par rapport à un autre régle r_b :

$X1 \rightarrow Y1$ si et seulement si $conf(r_a) = conf(r_b) \wedge sup(r_a) = sup(r_b) \wedge X1 \subseteq X \wedge Y \subseteq Y1$.

Exemple. Considérez les règles d'association présentées dans la partie droite de Figure 1. La règle $\{a\} \rightarrow \{c, f\}$ est redondante par rapport à $\{a\} \rightarrow \{c, e, f\}$. De plus, la règle $\{a, b\} \rightarrow \{e, f\}$ est redondante par rapport à $\{a\} \rightarrow \{e, f\}$.

Définition du problème :

Sur la base de la définition précédente de redondance et de ladéfinition d'extraction des règles d'association Top-k, nous définissons le problème d'extraction des régles l'association Non redondante comme suit.

Le problème de d'extraction des régles l'association Non redondante de Top-k est de découvrir un ensemble L contenant des règles d'association k dans une base de données de transaction.

Pour chaque règle $r_a \in L$ |conf (ra) $\geq$ minconf, il n'existe pas une règle $r_b \notin L \mid conf(r_b) \geq minconf \wedge sup(r_b) > sup(r_a)$, sinon r_b est redondante par rapport à r_a.

L'algorithme Top-k Non redundante :

Pour résoudre le problème de l'extraction des règle non redondante de Top-k, nous proposons un algorithme nommé **TNR**. Il est basé sur la recherche en profondeur d'abord

comme dans la procédure de recherche de l'lagorithme TopKRules. la différence entre TNR et TopKRules réside dans la façon d'éviter de générer des régles redundantes.

La procédure de recherche :

Pour expliquer la procédure de recherche, nous introduisons quelques définitions.

Une règle X → Y est de taillep * q si | X | = P et | Y | = Q. Par exemple, la taille de {a} → {e, f} est 1 * 2. De plus, nousdire qu'une règle de taille p * q est supérieure à une règle de taille r*s si p> r et q ≥ s, ou si p ≥ r et q>s. Une règle d'association r est valide si sup (r) ≥ minsup et conf(r) ≥ minconf.La procédure de recherche prend comme paramètre une base de données de transaction, un entier k et le seuil minconf. La procédure de recherche définit d'abord une variable d'minsup interne à 0 àveiller à ce que toutes les règles top-k sont trouvés. Ensuite, la procédure commence à chercher des règles.

Dès qu'une règle est trouvée, il est ajouté à une liste de règles L guidés par le support. La listeest utilisé pour maintenir les règles trouvées jusqu'à présent top-k et toutes les règles qui ont le mêmesupport. Une fois que k règles valides sont présents dans L, la variable minsup interne est élevée à lal'appui de la règle avec le support le plus bas en L. l'augmentation de la valeur de minsup est utilisé pourélaguer l'espace de recherche lors de la recherche pour plus de règles. Ensuite, chaque fois qu'une règle trouvée est valable, la règle est insérée dans L, les règles en L ne respectant plus minsupsont retirées de L, et minsup est portée à l'appui de la règle avec leplus faible support en L. L'algorithme continue la recherche de plus de règles jusqu'à ce qu'aucune règle esttrouvée. Cela signifie qu'il a trouvé des règles top-k en L. Les règles top-k sont des règles de kavec le plus grand support en L [11].

Pour chercher des règles, la procédure de recherche explore d'abord la base de données pour identifier des éléments uniques qui apparaît dans moins des opérations de minsup. Il utilise ces éléments pour générer des règles detaille 1 * 1 (contenant un seul élément dans l'antécédent et un seul élément de la partie résultat). Ensuite, chaque règle est récursive en ajoutant des éléments à son antécédent ourésultat (une recherche en profondeur d'abord). Pour déterminer les éléments qui devraient être ajoutés à unerègle générale, la procédure de recherche analyse les transactions contenant la règle pour trouver des objets simplesqui pourrait élargir son antécédent ou résultat. Les deux processus d'expansion des règlessont nommés d'expansion gauche et l'expansion droite. Ces processus sont appliqués de manière récursivepour explorer l'espace de recherche de règles d'association. la procédure d'expansion Gauche et droite sont formellement définies comme suit:

Une expansion gauche est le processus d'ajout d'un élément i sur le côté gauche dela règle X → Y pour obtenir une plus grande règle X∪ {i} → Y. Une expansion droite est le processus deajout d'un élément i sur le côté droit d'une règle X → Y pour obtenir une règle plus large X → Y∪ {i}.

L'adaptation de la méthode de recherche pour découvrir les régles d'association de Top-k Non redundante :

Nous expliquons maintenant comment nous avons adapté la procédure de recherche pour concevoir un algorithme efficace de génération des régles d'association de Top-k Non Rdundantes. en se basant sur deux stratégies décrit comme suit :

Stratégie N°01 :

Pour chaque règle r_a qui est généré de telle sorte que sup (r_a) ≥ minsup, si il existe $r_b \in L$ | sup (r_b) = sup (r_a) et r_a est redondante par rapport à r_b, Alors r_a ne sera pas ajouté à L. Dans le cas contraire, r_a est ajouté à L.

Strategy N°2:

Pour chaque règle r_b qui est a été généré de telle sorte que sup (r_b) ≥ minsup, s'il existe $r_a \epsilon$ L | sup (r_b) = sup (r_a) et r_a est redondante par rapport à r_b, Alors r_a est retiré de L [11].

3.4 Conclusion :

Nous venons présenter dans ce chapitre, les différents algorithmes de génération des règles d'association, dans le chapitre suivant, on va voir les outils d'implémentation de notre application, Ainsi, discussion sur les résultats obtenus après l'exécution de chaque algorithme choisis.