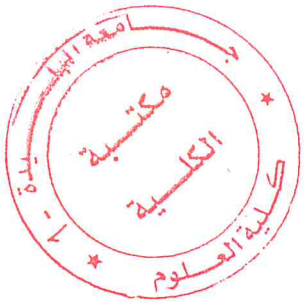
République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

**Université Saad Dahlab Blida**

N° D'ordre :.....




Faculté des sciences

**Département d'informatique**

Mémoire Présenté par :

TALEB Mohamed Akrem

Messsouter Mhamed

**En vue d'obtenir le diplôme de Master**

**Domaine** : Mathématique et informatique

**Filière** : Informatique

**Spécialité** : Informatique

**Option** : Ingénier logicielle

## *Thème*

# *Extraction des motifs fréquents à partir du flux de donnée*

**Soutenu le :**

| | |
|---|---|
| Mme OUAHRANI LIELA | Présidente |
| M FERFERA SOFIANE | Examinateur |
| Mme ZAHRA FATMA ZOHRA | Promotrice |

**Promotion :** 2017 / 2018

# *Remerciements*

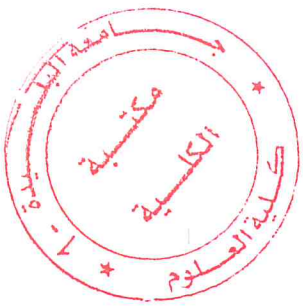
*C'est avec l'aide de Dieu que ce travail a vu le jour,*

*Il n'aurait pu être achevé sans le soutien, les conseils et les encouragements de certaines personnes auxquelles nous tenons à exprimer ici nos sincères remerciements.*

*En premier lieu, nous exprimons toute notre gratitude pour notre Promotrice, Mlle. ZAHRA pour ces précieux conseils, sa disponibilité, la confiance qu'elle nous a toujours témoigné et la sollicitude dont elle nous a entouré, et ce tout au long de l'élaboration du présent travail. Nous n'oublions pas non plus nos Enseignants, qui nous ont transmis leur savoir tout au long du cycle d'études à l'UNIVERSITE DE BLIDA 1.*

*Nous adressons une pensée particulièrement affective à nos Amis qui ont rendu agréables nos longues années d'études.*

*Nous tenons enfin à remercier tous ceux qui ont collaborés de près ou de loin à l'élaboration de ce travail. Qu'ils acceptent nos humbles remerciements.*



## RESUME

Ces dernières années nous avons remarqué l'apparition de nombreuses applications qui traitent les données générées en continu à grandes vitesses, ces données sont connues sous le nom « flux de données ou bien Data Stream». En effet, on trouve ce genre de base de données dynamiques dans des nombreux domaines (transactions bancaires, l'usage du Web, la surveillance des réseaux, etc.). Les fameuses caractéristiques des flux de données sont la vélocité, la continuité dans l'arrivé des données et la taille importante de données générées.

L'extraction de motifs fréquents à partir de flux de données pose beaucoup des problèmes, parmi ces problèmes citons par exemple l'impossibilité de bloquer le flux de données, afin de produire des résultats en temps réel.

Dans ce travail nous avons proposé une méthode d'extractions des itemsets fréquents a partir de flux de donnée de façon incrémentale, l'expérimentation effectuée a montré l'efficacité de notre algorithme proposé.



Mots clés : Extractions des motifs fréquents, flux de données, itemsets fréquents, algorithmes incrémental.



## ABSTRACT

In recent years, we have noticed the appearance of many applications that dealt with data generated continuously at high speeds; these data are called "Data Streams". We find this kind of dynamic data in many applications (like banking transactions, the use of the Web, network monitoring, etc.), the famous characteristics of data streams is speed and quantity, where quantity is very large or infinite.

There are many problems with processing the data streams for extracting frequent patterns, among these problems, for example, the impossibility of blocking the data stream, to produce results in real time.

In this work, we proposed a method of frequent pattern mining from data streams in incremental way. The experiments have shown the efficiency of our proposed algorithm.



Keyword(s) : Frequent Pattern Mining, Data Streams, Frequent itemsets, Incremental Algorithms.

## ملخص

لقد لاحظنا في السنوات الأخيرة ظهور العديد من التطبيقات التي تتناول البيانات التي يتم إنشاؤها بشكل مستمر بسرعات عالية ، وتعرف هذه البيانات باسم "تدفق البيانات" ما نراه في العديد من المجالات (مثل المعاملات المصرفية ، استخدام شبكة الإنترنت ، ومراقبة الشبكة ، وما إلى ذلك) ، والخصائص الشهيرة لتدفق البيانات هي السرعة والكمية حيث الكمية كبيرة جدا وغير متناهية . معالجة تدفق البيانات لاستخراج أنماط متكررة، وتشمل هذه المشاكل، على سبيل المثال، استحالة منع تدفق البيانات، لتحقيق نتائج في الوقت الحقيقي. في هذا العمل اقترحنا طريقة للاستخراج نمط المتكررة من البيانات المتدفقة. التجارب اثبت فعالية الطريقة المقترحة .



كلمات مفتاحيه: استخراج نمط متكرر، تدفق البيانات ,خوارزميات تزايدية.

# Sommaire

# Sommaire

# Sommaire

# Liste des figures

## Liste des figures

# Liste des Tableaux

## Liste des Tableaux

# Chapitre 1 : Extraction des motifs fréquentes

Introduction générale

## 1. Contexte et problématique

L'extraction des motifs fréquents à partir de données est une technique importante de Data Mining. En effet, cette technique est largement utilisée depuis la publication de l'algorithme Apriori par *Agrawal et al* en 1993. La plupart des algorithmes d'extraction des motifs fréquents sont conçus pour être appliqués à des bases de données statiques. Cependant, dans une grande partie des applications du monde réel, les données arrivent d'une façon continue (appelée Data Stream ou flux de données), constituant ainsi des bases de données dynamiques.

L'extraction des motifs fréquent à partir de données de flux de données est intéressante dans plusieurs domaine afin de trouver des liens entre les données générées à partir de plusieurs applications telles que :

- Les réseaux de capteurs.
- La surveillance et la gestion du trafic réseau.
- Les processus de fabrication.
- La surveillance écologique.
- l'analyse des marchés boursiers.
- La santé bio-informatique.
- et autres …

Dans les modèles de flux de données, les données arrivent à une grande vitesse et d'une façon continue. Par conséquent, les algorithmes de traitement de ce genre de données travaillent sous des contraintes très strictes d'espace et de temps d'exécution. En effet, les flux de données posent plusieurs défis pour la conception d'algorithmes de fouille de données. Premièrement, les algorithmes doivent utiliser des ressources limitées (temps et mémoire). Deuxièmement, ils doivent traiter des données dont la nature numérique ou textuelle et prend en considération la distribution des donnée en d'autres terme les arrives régulier et non régulier du données, plusieurs travaux ont été proposés dans ce sens ce qui a engendré un nouveau champ de recherche en Data Mining.

# Chapitre 1 : Extraction des motifs fréquentes

## 2. Objectif du travail

Notre travailler consiste à développer une méthode d'extractions des items set fréquent à partir du flux de donnée de manier incrémentale c'est à dire lorsque les donner arrive par lot on fait l extractions sauf sur le dernier lot contrairement aux algorithmes de traitement par lots, qui traitent toute la base de données à partir de zéro à chaque arrivée de nouvelles de données, les algorithmes incrémentaux mettent à jour et extraient progressivement les itemsets sans réplication de processus d'extraction sur toute la masse de données à chaque mise à jour de données, réduisant ainsi le coût de découverte des itemsts fréquents.

De plus, dans ce travail, on s'intéresse à l'intégration d'une méthode aux algorithmes incrémentaux d'extraction des itemsets à partir de data streams afin d'améliorer d'une façon continue la qualité des itemsets extraits.

## 3. Organisation du mémoire

Le mémoire se réparti en cinq chapitres.

**Chapitre 1 : « extraction des motifs fréquents »**

Ce chapitre est consacré à étudié les différentes d'algorithmes d'extraction des motifs fréquentes.

**Chapitre 2 : « Méthodes de traitement du flux de donnée»**

Dans ce chapitre nous allons présenter différent model qui permet de traitait les flux avec des algorithmes et une comparaison sur les différents algorithmes selon le model

**Chapitre 3 : « Extraction incrémentale »**

Dans ce chapitre nous détaillerons les différentes approches qui permet de faire une extraction incrémentales sur les flux de données »

**Chapitre 4 : « Approche proposée »**

# Chapitre 1 : Extraction des motifs fréquentes

Ce chapitre, sera réservé pour notre propres méthodes que nous allons utiliser dans notre application

**Chapitre 5 : « Test et Validation »**

Le dernier chapitre sera pour valider les méthodes adoptées, et on terminera par une conclusion

# Chapitre 1 : Extraction des motifs fréquentes

## Chapitre I : Extraction des motifs fréquents

### 1. Extraction des Itemsets frequents

L'extraction des Itemsets fréquents est une étape intermédiaire pour découvrir des règles d'association et dans certains domaines d'application, le résultat de cette technique est directement utilisé et interprété ou bien, il sera utilisé comme une entré pour d'autre méthodes de découverte de connaissance.

L'idée principale du Data Mining est La recherche des régularités dans les bases de données. Ces régularités s'expriment sous différentes formes, on donne un exemple très connu, par exemple dans l'analyse du panier d'achats de consommateurs, l'extraction des itemsets consiste à mettre en évidence les cooccurrences entres les produits achetés c.-à-d. Déterminer les produits (les items) qui sont « souvent » achetés simultanément. On parle alors d'itemsets fréquents.

**Tableau 1. 1 Base de transaction (Représentation Binaire )**

| S1 | S2 | S3 | S4 |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 |

# Chapitre 1 : Extraction des motifs fréquentes

Par exemple, en analysant les tickets de caisse d'un supermarché, on pourrait produire un ensemble des item set du type « le pain et le lait sont présents dans 10 % des caddies ». S1 et S2 respectivement. Le fichier comporte 10 observations (transactions) et 4 items (figure 1.1)

Voici quelques définitions de base concernant les concepts d'extraction des itemsets fréquents :

**Item :** Un item correspond à un produit. Nous avons 4 items (S1, S2, S3 et S4) dans notre fichier. **[01]**

**Support :** Le support d'un item est égal au nombre de transactions dans lesquelles il apparaît. **[01]**

**Itemset :** Un itemset est un ensemble d'items. Le support d'un itemset comptabilise le nombre de transactions dans lesquelles les items apparaissent simultanément. Un itemset peut être composé d'un singleton. **[1]**

**Itemset fréquent :** Un itemset est dit fréquent si son support est supérieur à un seuil défini à l'avance, paramètre de l'algorithme de recherche. **[1]**

**Superset :** Un super set est un itemset défini par rapport à un autre itemset. **[1]**

**Itemset fermé (closed itemset)** : Un itemset fréquent est dit fermé si aucun de ses super sets n'a de support identique. **[1]**

**Itemset maximal (maximal itemset) :** Un itemset est dit maximal si aucun de ses super sets n'est fréquent. **[1]**

**Itemset générateur (generatoritemset) :** Un itemset A est dit générateur s'il n'existe aucun itemset B tel que B $\subset$A et que SUP(B) = SUP(A). Autrement dit, l'itemset est générateur si tous ses sous itemsets ont un support strictement supérieur. **[1]**

# Chapitre 1 : Extraction des motifs fréquentes

## 2. Algorithme d'extractions des motifs fréquent

### 2.1. L'algorithme A-priori

L'algorithme A-prioriest un algorithme d'exploration de données conçu en 1994, par Rakesh Agrawal et Ramakrishnan Sikrant [3], dans le domaine de l'apprentissage des **règles d'association**. Il sert à reconnaître des propriétés qui reviennent fréquemment dans un ensemble de données et d'en déduire une catégorisation.

A-Priori détermine les règles d'association présentes dans un jeu de données, pour un seuil de support et un seuil de confiance fixés. Ces deux valeurs peuvent être fixées arbitrairement par l'utilisateur.

Une règle d'association est une règle ayant la forme $ai = vi, aj = vj, \dots am = vm$,ce qui s'interprète par **« si les attributs $ai, aj, \dots am$ ont une certaine valeur, alors l'attribut $a?$ prend généralement une certaine valeur $v?$ , $a\beta$ une certaine valeur $v\beta$, … ».**

La difficulté consiste notamment à trouver des règles qui soient significatives et non seulement le résultat du hasard.

Dans un jeu de données, on dispose de xi éléments (ex : lignes dans une table de données) connus aussi sous le nom de Transactions. Chaque élément est décrit par un ensemble d'attributs aj (ex : colonnes dans une ligne de données). Un attribut correspond aussi à un item. Un ensemble d'items est dit fréquent s'il correspond à un motif fréquent dans la base de transactions.

Chaque règle d'association a deux mesures : le « ***support*** » et la « ***confiance*** ».

- Le « ***support*** » d'un ensemble d'items est défini comme la fréquence d'apparition simultanée des items figurant dans l'ensemble des données.
- La « ***confiance*** » d'une règle « si condition alors conclusion » est le rapport :

$$\frac{\textit{Nbre de données où les items de la condition et de la conclusion apparaissent simultanément}}{\textit{Nombre de données où les items de la condition apparaissent simultanément}}$$

- Le tableau suivant illustre l'exemple de paniers d'achats de certains clients :

**Tableau 1. 2 :** Exemple de paniers d'achats de certains clients.

| | Article | Article | Article | Articl |
|---|---|---|---|---|

| | A | B | C | e D |
|---|---|---|---|---|
| **Client 1** | X | X | | |
| **Client 2** | X | | X | |
| **Client 3** | | X | | |
| **Client 4** | X | | X | X |
| **Client 5** | | X | | |

- $Support\ (A, B) = 1/5$ , Car $A$ et $B$ n'apparaissent simultanément que dans le panier du Client 1.
- $Support\ (A, C) = 2/5$ , Car $A$ et $C$ apparaissent simultanément dans le panier des clients 2 et 4.

On dit qu'un ensemble d'items est un ensemble d'***items fréquents*** si le support de cet ensemble d'items est supérieur à un certain seuil ($> 5\%$ par exemple).

- $confiance\ (si\ A, alors\ B) = 1/3$, Car $A$ et $B$ apparaissent simultanément dans le panier de 1 $client$ et $A$ apparait dans le panier de3 $clients$.
- $confiance\ (si\ A, alors\ C) = 2/3$, Car $A$ et $C$ apparaissent simultanément dans le panier de 2 $clients$ et A apparait dans3 $clients$.

On définit un seuil de confiance comme la valeur minimale que $la\ confiance$ doit avoir pour que l'apparition simultanée des items considérés ne puisse pas être simplement due au hasard. Toutefois on ne s'intéresse qu'aux règles ayant une confiance maximale.

### 2.1.1. Fonctionnement

L'algorithme A-Priori fonctionne en deux phases : on commence par rechercher les *ensembles d'items fréquents* (EIF), ensuite, on utilise ces EIF pour déterminer les règles d'association dont la confiance est supérieure au seuil fixé.

La construction des EIF est itérative : on construit d'abord les EIF contenant un seul item, puis ceux contenant 2 items, puis ceux contenant 3 items, …On commence par déterminer les EIF de taille 1, on note cet ensemble**L1**. Ensuite, on construit l'ensemble **C2** des EIF candidats de taille 2 : ce sont tous les couples construits à partir des EIF de taille 1. On obtient la liste des EIF de taille 2 (**L2**) en ne conservant que les éléments de **C2** dont le support est supérieur au seuil. On construit alors**C3**, l'ensemble des triplets d'items dont les 3 sous-paires

sont dans **L2** et on ne retient que ceux dont le support est supérieur au seuil, ce qui produit**L3**. Et ainsi de suite, tant que **Li** n'est pas vide.

**Remarque :** Toutefois, si on considère $\mathbf{X_K}$ un sous-ensemble d'items fréquent, tous les sous-ensembles d'items contenus dans $\mathbf{X_K}$ et qui soient de longueurs inférieures à $k$ sont fréquents. Par exemple si $ABCD$ est un sous-ensemble d'items fréquents, alors, les sous-ensembles : $ABC, ABD, BCD, AB, AC, BC, BD, CD, A, B, C, D$ les sont aussi.

```
procedure Apriori (T, minSupport) { //T is the database and minSupport is the minimum support
        L1= {frequent items};
        for (k= 2; L_k-1 !=∅; k++) {
                C_k= candidates generated from L_k-1
                //that iscartesian product L_k-1 x L_k-1 and eliminating any k-1 size itemset that is not
                //frequent
                for each transaction t in database do{
                        #increment the count of all candidates in C_k that are contained in t
                        L_k = candidates in C_k with minSupport
                }//end for each
        }//end for
        return ∪_k L_k;
}
```

**Figure 1. 1 :** L'algorithme A-priori [3].

On notera que la donnée $C_k$ (ainsi que$L_k$) est un ensemble d'enregistrements contenant deux champs :

- **Itemset :** ce champ contient le sous ensemble d'items,
- **Sup** : ce champ contient la fréquence de cet ensemble dans la base de transactions.

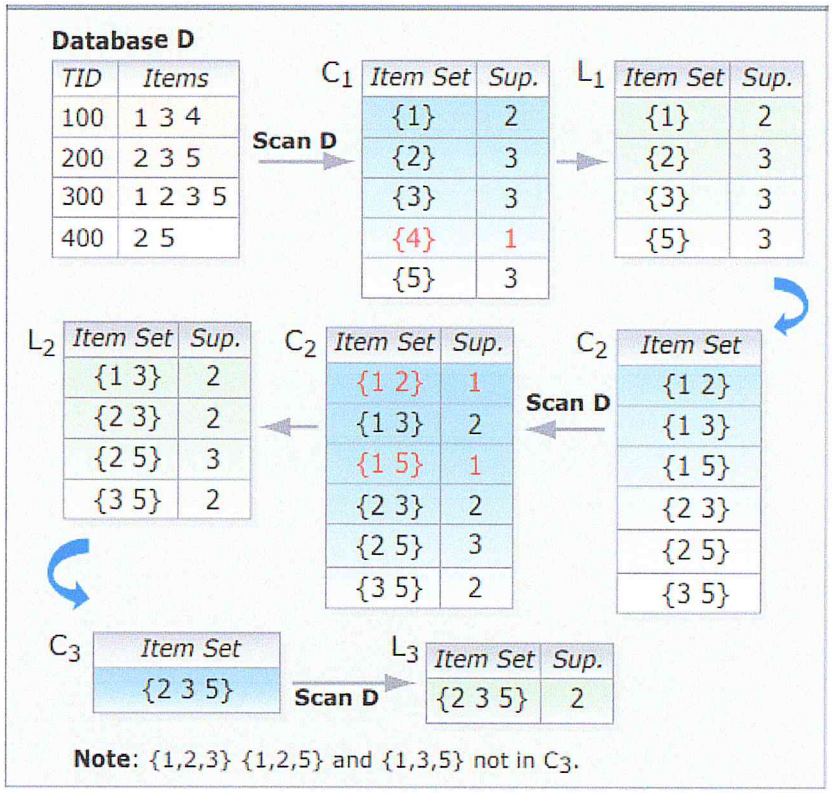



**Figure 1. 2 :** L'algorithme A-priori Exemple [5]

L'inconvénient majeur de cet algorithme est le nombre considérable de l'accès à la base de données ***D***. Une amélioration qui consiste à intégrer les identificateurs des transactions est proposée dans la section suivante.

### 2.2. L'algorithme FP-Growth

L'algorithme **FP-growth** (Fréquent Pattern growth) [4] consiste d'abord à compresser la base de données en une structure compacte appelée **FP-Tree** (*Fréquent 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.

### 2.2.1. Structure d'un FP-tree

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

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

– L'arbre quant à lui est composé, comme indiqué, d'une racine **nulle** et d'un ensemble de nœuds de nœuds préfixé par l'élément représenté.

Un nœud de l'arbre est composé par :

- Le nom de l'item (nom-item). Il s'agit de l'item que représente le nœud.
- Le nombre d'occurrence (count) de transaction où figure la portion de chemin jusqu'à ce nœud.
- Un lien vers le nœud suivant dans l'arbre (node-link). Il s'agit d'un lien inter-nœud vers les autres occurrences du même élément (ayant le même nom-item) figurant dans d'autres séquences de transactions. Cette valeur prend la valeur ***nulle*** s'il n'y a pas un tel nœud.

– 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 nœuds ayant ce même nom-item.

### 2.2.2. 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 alors triée selon l'ordre des items dans L. Le nœud racine de l'arbre nulle est d'abordcrée. Durant ce même parcours, une branche sera créée pour chaque transaction, mais des 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. Ceci fait du FP-tree une bonne structure compacte pour représenter les bases transactionnelles.

### 2.2.3. Récapitulatif de l'algorithme Fp-growth

**Algorithme FP-growth**

1. Balayer la base de transactions T une première fois
   - Créer L, la liste des items fréquents avec leur support
   - Trier L en ordre décroissant du support
2. Créer l'arbre N contenant une racine étiquetée « Null »
3. Procédure FP-growth(Fp-tree, *null*)
   A. Si FP-tree contient un seul chemin *P* alors
      - Pour chaque combinaison $\beta$ de *P* faire
      - Générer l'itemset $\beta$ U $\alpha$ de support = minimum des supports des nœuds de $\beta$

   B. Sinon pour chaque $a_i$ dans l'indexe de FP-tree faire
      - Générer l'itemset $\beta = a_i$ U $\alpha$ de support = *ai.support*
      - Construire l'itemset condition de base de $\beta$
      - Construire FP-tree$_\beta$
      - Si FP-tree$_\beta \neq 0$
        - FP-growth(FP-tree$_\beta$, $\beta$)

**Figure 1. 3 : L'algorithme** Fp-growth [4].

### 2.2.4. Étapes de la construction du FP-tree

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

$$Supportminimum = (50/100 * 5) = 2.5$$

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. Le résultat obtenu, 3 dans notre cas, constitue le support minimum et par conséquent tous les items da la base de transactions ayant un support inférieur à **3**occurrences minimums sera ignoré.

**Tableau 1. 3 : La** base de transaction

| 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 |

2. Parcours de la base des transactions pour trouver la somme totale des différentes occurrences : 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.

**Tableau 1. 4 :** Occurrences des items .

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

3. Définir la priorité des éléments, puis trier les items en fonction de leur priorité : 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'occurrences. Ce tri s'effectue en ordre décroissant, l'élément ayant comptabilisé le plus grand nombre d'occurrences est placé en tête et l'élément ayant comptabilisé le moins d'occurrences est placé en queue.

**Tableau 1. 5** : Items fréquents Ordonnées

| 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 |

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'insertion des différents nœuds.

4. Création du nœud racine : 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.

5. Insertion des nœuds enfants : On commence par parcourir chaque élément de la transaction (voir illustration plus loin). Puis pour chaque élément de la transaction on vérifie l'existence d'un nœud correspondant, s'il n'existe pas, le nœud est créé, dans le cas contraire le nombre d'occurrence 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 la fin du traitement de toutes les transactions de la base la structure finale de Fp-tree est illustrée par la figure suivante :

### 2.2.5. Fonctionnement de FP-growth

A la fin de la deuxième étape de la construction du $\boldsymbol{FP-tree}$ nous avons pu définir la table des entêtes. Cette table, tout comme le FP-tree obtenu à la fin de la phase de construction, vont nous servir à identifier les itemsets fréquents.

Pour identifier les motifs fréquents contenant l'élément p, la table des entêtes est parcourue de bas en haut ; commençant par l'élément $\boldsymbol{p}$ de la table et se terminant par l'élément$\boldsymbol{c}$ (le dernier élément $\boldsymbol{f}$ est ignoré).

A partir de la table des pointeurs, on parcourt le $\boldsymbol{FP-tree}$ en suivants les pointeurs pour l'item fréquent $\boldsymbol{p}$. Puis, nous créons tous les chemins à partir de la racine et se terminant par $\boldsymbol{p}$

pour former la base des motifs $p - conditionnés$. Ceci constitue l'ensemble des préfixes de $p$.

- Les transactions contenant $p$ doivent avoir le nombre d'occurence de p, par conséquent nous obtenons : $(f: 2, c: 2, a: 2, m: 2, p: 2)$ Et $(c: 1, b: 1, p: 1)$
- Etant donné que $p$ fait parti de ces transactions, nous pouvons l'ignorer et nos chemins, appelés **Motif de Base Conditionné** (Conditional Pattern Base, CPB) deviennent : $(f: 2, c: 2, a: 2, m: 2)$ Et $(c: 1, b: 1)$

Un Chemin de Base Conditionné contient toutes les transactions dans lesquelles un élément répète. Pour trouver tous les motifs fréquents contenant un élément $n$ 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**.

Pour terminer l'identification des motifs fréquents 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.

$$(f: 2, c: 2, a: 2, m: 2), (c: 1, b: 1) \Rightarrow (c: 3)$$

Seul l'élément $c$ est fréquent ça c'est le seul élément qui un nombre d'occurrences supérieur au support minimum. Là aussi, il faut éliminer tous les éléments dont le support est inférieur au support minimum et les prioriser comme décrit précédemment. La valeur de l'occurrence 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 : $(p : 3, cp : 3)$

Concernant les autres éléments de la table des entêtes, la procédure est la même. Le tableau suivant récapitule les résultats obtenus :

**Tableau 1. 6 :** Tableau récapitulatif des bases des motifs x-conditionnés [5].

| Item | Conditional Pattern Base | Conditional FP-tree |
|---|---|---|
| p | {(fcam:2), (cb:1)} | {(c:3)} \| p |
| m | {(fca:2), (fcab:1)} | {(f:3, c:3, a:3)} \| m |
| b | {(fca:1), (f:1), (c:1)} | ∅ |
| a | {(fc:3)} | {(f:3, c:3)} \| a |
| c | {(f:3)} | {(f:3)} \| c |
| f | ∅ | ∅ |

### 2.3. L'algorithme Eclat

Introduit par M. J. Zaki en 1997 [2], 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, désignés par les items qu'ils contiennent dans l'ordre lexicographique, possèdent 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 des portions 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.

#### 2.3.1. .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 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 figure 1.5

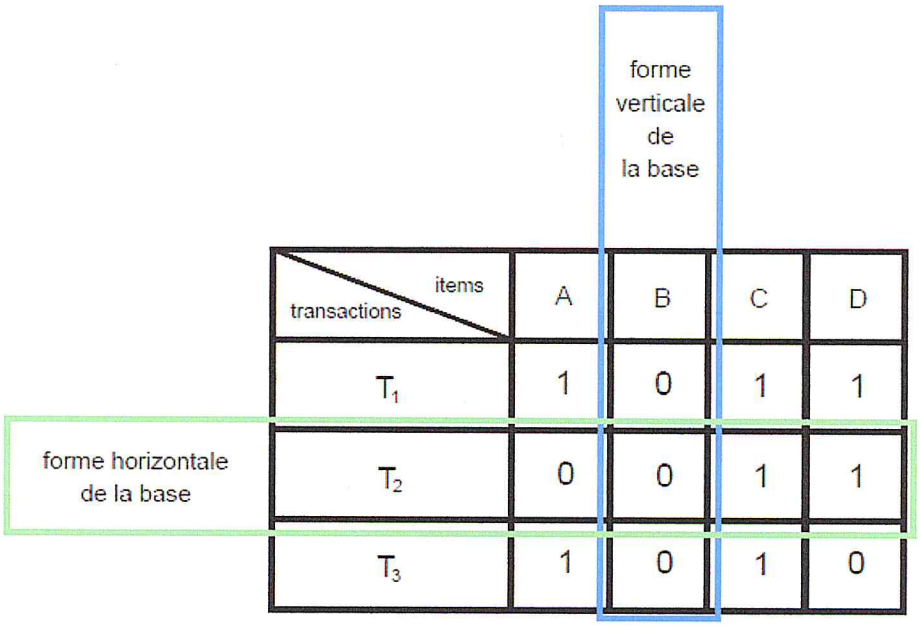



**figure 1. 4 Formes horizontale et verticale de la base.**

2.3.2 Fonctionnement

**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.

**Figure 1. 5 :** L'algorithme Eclat [2].

**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 la base 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 infréquentés avant de générer les itemsets de $taille$ 2. A ce stade, on dispose de l'ensemble des itemsets fréquents $L2$.

– On scanne la base de données.

– On construit un tableau de deux dimensions indexées par les items sur la hauteur et la largeur. Dans cet exemple pour une base contenant 4 items, on rencontre les itemsets $AB, AC, BDetCD$chacun respectivement dans au moins une transaction de la base.

– Chaque processeur calcule le support local des itemsets puis effectue une réduction de somme des résultats des autres processeurs afin de construire les supports globaux de chaque itemset de $L2$.

**Tableau 1. 7 :** Exemple

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

**La phase de transformation :** L'algorithme commence par partitionner L2 en classes d'équivalences qui seront redistribuées surles 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 L2 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. On considère toutes les paires à traiter par la suite. Ainsi, la charge est calculée par la valeur $C_s^2$ .

Par exemple, si dans la classe d'équivalence $[A]$ on trouve les itemsets $AB, AC\ et\ AD$, la charge calculée sera $C_3^2 = 3$. Il s'agit alors de répartir les tâches à effectuer sur les processeurs en 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 de l'assignation.

Chaque processeur peut effectuer cette tâche indépendamment des autres puisqu'ils disposent alors tous des supports globaux de $L2$.

– Phase de transformation verticale de la base. Chaque processeur scanne sa portion locale de la base afin de construire les listes de transactions correspondant aux classes d'équivalence de $L2$. Il faut alors transmettre respectivement les listes aux processeurs chargés de la classe d'équivalence correspondante.

**La phase asynchrone :** Les processeurs effectuent concurremment la construction des itemssets de tailles croissantes par intersection des listes de transactions des éléments de chaque classe d'équivalence entre eux. Eliminant les itemssets de support insuffisant, on réduit rapidement le travail à effectuer en même temps qu'on augmente la taille des itemssets 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

## 3. Conclusion

Dans ce chapitre, une étude globale des algorithmes classiques d'extraction d'itemsets fréquents à partir de données. Apres cette étude on a conclus que la différence entre ces algorithmes est de savoir comment ils génèrent la sortie ou le résultat le résultat est le même, mais le processus pour obtenir le résultat est différent, Apriori utilise une approche de niveau où il va générer des modèles contenant 1 éléments, puis 2 éléments, 3 éléments. D'un autre côté, FPGrowth utilise une recherche en profondeur au lieu d'une recherche en largeur, Dans le prochain chapitre, nous présentons des modèles de traitement des flux des données avec une collection d'algorithmes, ensuite nous proposons une classification et une comparaison entre ces algorithmes

# Chapitre 2 : Extraction des motifs fréquents à partir de flux de données

## 1. Introduction

Dans un flux de données, les données arrivent à grande vitesse, de sorte que les algorithmes utilisés pour extraire les motifs fréquents dans les flux de données doivent les traiter dans des contraintes d'espace et de temps très strictes.

Cela soulève de nouveaux problèmes qui doivent être pris en compte lors du développement d'algorithmes d'exploration de règles d'association pour les flux de données.

Il est donc important d'étudier les algorithmes d'exploration de flux existants pour ouvrir les défis et les possibilités de recherche. Dans ce chapitre, nous discutons différentes méthodes de traitement de flux de données ainsi que les algorithmes que leurs algorithmes d'extraction des motifs fréquents.

## 2.. Approches sur les techniques d'extractions des motifs fréquentes a partir flux du donnée

La nature des motifs fréquents extraits entraîne généralement un grand nombre de générations d'ensembles d'éléments fréquents l'augmentation du nombre motif générés fréquemment entraînera la dégradation de l'efficacité d'extractions en termes de temps et de consommation du mémoire il existe deux technique.

- d'extractions L'extraction des éléments fermés fréquents (FCI)
- Frequent Maximal Item set (FMI)

La première basée sur le concept FCI , le FCI est une représentation non redondante de l'ensemble des itemsets fréquents [8]. La réduction louable de la taille de l'ensemble de résultats conduit à des performances améliorées en termes de vitesse et d'utilisation de la mémoire. Différents algorithmes FCI [7, 9, 10] efficaces sont proposés par différents auteurs.

La deuxième technique d'extraction d'ensemble d'éléments fréquents appelée Frequent Maximal Item set [6] basée sur le concept FMI. Comparé à la technique d'extraction des ensembles d'objets fermés fréquents FCI, il génère relativement moins d'ensembles d'objets, ce qui explique qu'il soit beaucoup plus efficace [11] en termes de CPU et de mémoire. Mais l'inconvénient de l'extraction FMI est qu'elle perd les informations de fréquence du sous-ensemble des IMF, de sorte que la limite d'erreur augmente également. Il y a beaucoup de représentations concises des ensembles items fréquents qui sont proposés [12, 13, 14, 15, 16, 17], ceux-ci économisent significativement de l'espace mémoire, du CPU et montrent une meilleure précision. Cette technique pourrait être appliquée dans l'exploitation des motifs fréquents avec la technique incrémentielle en utilisent traitement par lots

## 3. Approche sur les types des fenêtres aux model flux du donnée

L'un des principaux problèmes dans les flux des donnes c'est l'exploitation, est de trouver un modèle qui conviendra au processus d'extraction de l'ensemble d items fréquents dans le flux de donnée il y a trois model de traitement [18] du flux de donner

- Le modèle "Landmark".
- Le modèle "time-fading".
- Le modèle "sliding window".

Les algorithmes d'extraction des itemsets fréquents sont basés sur ces modèles. Par conséquent, on a classifié ces algorithmes par rapport le modèle qu'ils les utilisent, les flux de données sont classés en deux classes :

- 1) flux de données hors ligne: caractérisés par une discontinuité ou des arrivées massives régulières [18], Comme un ajout en masse de nouvelles transactions comme dans un système d'entrepôt de donnée.
- 2) les flux de données en ligne: qui se caractérisent par des données mises à jour en temps réel qui viennent une par une dans le temps, comme une transaction générée en continu dans un système de surveillance de réseau

. Un flux de données de transaction c'est une séquence de transactions entrantes, et un extrait du flux est appelé une « fenêtre ». Une fenêtre W peut être (1) basée sur le temps ou basée sur le nombre, et (2) soit une fenêtre de point de repère ou une fenêtre glissante.

W est une fenêtre de point de repère (landmark windows) Si W = (T1, T2,. . . , T);W est une fenêtre glissante (sliding windows)si W = (TT-w + 1, ..., TT), où chaque Ti est une unité de temps ou un lot, T1 et TT sont les le plus ancien et l'unité de temps ou le lot actuel, Notez qu'une fenêtre basée sur le comptage peut également être capturée par une fenêtre basée sur le temps en supposant qu'un nombre uniforme de transactions arrive dans chaque unité de temps,

La fréquence d'un ensemble d'items X dans W, notée comme freq (X), est le nombre de transactions dans W , le support de X dans W, noté sup (X), est défini comme freq (X) / N, où N est le nombre total de transactions reçu dans W. X est un ensemble d'articles fréquents (FI) dans W, si sup (X) $\geq \sigma$, où $\sigma$ ($0 \leq \sigma \leq 1$) est spécifié par l'utilisateur. Dans le processus d'extraction de flux de données, il est nécessaire de conserver non seulement les éléments fréquents, mais aussi les éléments peu fréquents qui sont prometteurs d'être fréquents plus tard, car un élément peu fréquent peut devenir fréquent plus tard dans le flux. Par conséquent, beaucoup des algorithmes d'exploration de données approximatifs existants utilisaient un seuil de support minimum relaxé (également appelé paramètre d'erreur spécifié par l'utilisateur), $\varepsilon$, où $0 \leq \varepsilon \leq \sigma \leq 1$, pour obtenir un ensemble supplémentaire de jeux d'éléments susceptibles de devenir fréquents plus tard.

Il existe de nombreux algorithmes pour extraire des ensembles d'objets fréquents à partir de flux de données tous ces algorithmes sont tombés dans l'un des modèles d'exploration de flux de données qu'on a cité précédemment.

### 3.1. Le modèle « Landmark »

Ce modèle prend en considération toutes les données arrivées à partir d'un point spécifique dans le temps appelé point de repère (généralement le temps démarrage du système). Dans ce modèle, la découverte des connaissances est effectuée sur toute la masse de

données collectée entre le point de repère et le moment de lancement de processus d'extraction

## 3.2. Les algorithmes d'extraction d'itemsets fréquents basés sur le modèle « Landmark »

### 3.2.1. Algorithme FP-DS

L'Algorithme est présenté par *Liu X. et al* [22], il utilise une structure de donnée similaire à l'arbre FP-DS dans *Han J. et al, 2000*. L'utilisateur peut obtenir en continu des itemsets fréquents lorsque le motif extrait l'algorithme envoi le résultats directement.

Comparé aux algorithmes existants, l'algorithme FP-DS est particulièrement adapté à l'extraction des motifs fréquemment élevés. Il réduit la capacité de stockage des itemsets. De plus les itemsets sont placés dans l'ordre de la séquence descendante de support de 1-itemset global. Plus les éléments apparaissent fréquemment plus ils sont proches de la racine de l'arbre. Un tel arbre de compression a un taux de compression très élevé.

### 3.2.2. Algorithme StreamMining

L'algorithme StreamMining est proposé par Jin et Agrawal [32]. Il est construit sur l'idée présentée dans [24] pour déterminer des motifs fréquents. C'est un algorithme à deux passages ; qui nécessite seulement (1/θ) de la mémoire, où θ C'est la valeur de support souhaité. Leur premier passage calcule les totaux de motifs fréquents, le deuxième passage élimine tous faux positives k-itemsets. L'algorithme StreamMining adresse trois principaux défis pour l'exploitation des motifs fréquents dans un environnement de streaming :

1. Il développe une méthode pour trouver des k-itemsets fréquents, tout en conservant la taille du mémoire selon les exigences limitées.
2. Il développe un moyen d'avoir un lien entre les totaux calculés après le premier passage.
3. Il développe une structure de données et un certain nombre d'implémentation d'optimisations pour une exécution efficace. Cette structure de données appelée TreeHash, qui implémente un préfixe arbre en utilisant une table de hachage. Qui permet à faire des suppressions faciles dans la table de hachage. Ça aussi utilise un seuil support minimal nommée ε, comme presque tous les algorithmes

d'exploration de données pour les flux de données, de sorte que les exigences du mémoire augmentent proportionnellement à 1 / ε.

Donc, cet algorithme devrait avoir à calculer k-itemsets après le premier passage, et assurer un lien prouvable sur l'exactitude des résultats après le premier passage sur l'ensemble de données ; parce que c'est un environnement de streaming le second passage sur l'ensemble de données n'est pas généralement réalisable.

Par conséquent, il est important que l'ensemble de k-itemsets ne contienne pas beaucoup des faux itemsets. Pour trouver les itemsets fréquents, il prend l'espace O (1/θ), tandis que nécessite un espace O ((1 / θ) log (θ N)).

### 3.2.3. Algorithme Lossy Counting

L'algorithme Lossy Counting est proposé par *Cormode* [19], Il produit un ensemble approximatif des itemsets fréquents à partir d'un flux de données, ce dernier est divisé en une suite de segments et chaque segment est constitué de transactions.

$$B = 1 / \varepsilon.$$

Où chaque lot contient β (segments de transactions) L'idée « d'erreur maximale possible » est utilisée pour maintenir tous les itemsets fréquents possibles.

Bien que la sortie soit approximative, C'est garantie que l'erreur pour ne dépasse un seuil spécifié par l'utilisateur. D'après *Cormode* [19], cette méthode tente d'utiliser l'espace disponible le plus complètement que possible. Pour chaque nouvelle transaction, il génère tous les sous-ensembles et les stocke dans une structure compacte à base de trie. Lorsque l'espace est plein, il utilise un algorithme d'élagage pour supprimer les candidats les moins fréquents. Sachant que dans le modèle Landmark, l'extraction complète des itemsets fréquents est faite, bien qu'ils soient approximatifs.

## 3.3. Le modèle « Time-Fading »

C'est ce qu'on appelle le modèle Time-Fading selon *Chang et Lee 2004,* «Damped model» ou selon *Zhu Y. and Shasha D., 2002* «Time-titled» ou selon *Chen, Y. et al, 2002 et Pauray S. and Tsai M., 2009* ce dernier est considéré comme une variation du modèle landmark.

Les donnes sont prises en considération depuis le début des flux jusqu'au moment actuel, mais la période de temps est divisée en plusieurs tranches de temps et différents poids sont attribués aux transactions telles que les nouvelles transactions ont un poids plus élevé que les anciennes [50].

Le fading model a été proposé pour gérer le temps perdu dans le model landmarck. Ce model diminue l'effet des informations anciennes sur le résultat de l'extraction des motifs fréquents à partir de flux de données. En d'autres termes, il prend en considération l'effet des anciennes transactions d'une manière ou d'une autre. Il peut attribuer des pondérations différentes aux transactions de sorte que les nouvelles ont des poids plus élevés que les anciennes, ces poids diminuent avec le temps.

### 3.4. Les algorithmes bases sur « Fading time model »

#### 2.4.1. Algorithme estdec

L'algorithme est proposé par *Chang et Lee* [20]. Il utilise un paramètre d (0 <d <1), pour diminuer l'effet des anciennes transactions sur le résultat de l'extraction. Quand une nouvelle transaction arrive, la fréquence d'un ancien itemset est réduite par le facteur « d »

L'algorithme **estDec** adopte le mécanisme de *Hidber* [17] pour estimer la fréquence des itemsets. Par exemple, prenant d et v, les paramètres de fading de supports de l'ensemble d'éléments X, respectivement. Lorsqu'une nouvelle transaction contenant (X) élément arrive, le nouveau nombre de prise en charge de X est égal à (v × d + 1).

Évidemment, lorsque d est égal à 1, le modèle temporel (fading model) devient le modèle historique (landmark model). Supposons que le flux a reçu τ transactions ($T_1$,$T_2$,….,$T_\tau$), la fréquence décroissante d'un ensemble d'éléments, freqτ (X) est définis comme suit:

$$N_\tau = d^{\tau-1} + d^{\tau-2} + \cdots + d^1 + 1 = \frac{1-d^\tau}{1-d} \quad (1)$$

### 2.4.2. Algorithme TUF-Streaming

Les étapes clés de cet algorithme peuvent être décrites comme suit [62] :

- Premièrement, pour chaque lot Bi de données du flux, l'algorithme applique l'algorithme UF-growth avec une valeur preMinsup inférieure de vrai minsup pour trouver les motifs fréquents.
- Deuxièmement, il stocke les motifs fréquents extraits et leurs valeurs de support attendues dans une structure arborescente appelée UF-stream, dans lequel chaque nœud d'arbre correspondant à un motif X qui conserve une liste de valeurs de support.
- Troisièmement, il calcule le support attendu de chaque motif X dans chaque lot du flux de données par la fonction ExpSup (X, B) :

$$\text{ExpSup (X, B)}=\sum_{i=1}^{T} (\text{expSup ( X, Bi)} * \alpha^{T-i}) \quad \textbf{(2)}$$

à l'instant T, le support attendu de motif X extrait est calculé en sommant les supports attendus de ce motif X (pondérés par le facteur $\alpha$, où $0 \leq \alpha \leq 1$) sur tous les lots.

### 3.5. Le modèle « Sliding windows »

Le modèle de fenêtre coulissante ne maintient que les données récentes. La taille de la fenêtre coulissante peut être décidée en fonction des applications et des ressources du système.Les transactions générées récemment dans la fenêtre seront géré par rapport la taille de la fenêtre choisie préalablement. La taille de la fenêtre coulissante peut varier en fonction des applications qu'elle peut utiliser [41].

Le modèle « sliding windows » capture les modifications et tendances récentes motifs parce que les utilisateurs ne peuvent être intéressés que par les données arrivées récemment dans un délai déterminé. le model sliding windwos prend en considération que les ensembles d'éléments changent leurs fréquences en fonction de certains intervalles de temps. En d'autres termes, le modèle sliding windows ne tient pas compte de l'historique des fréquences des ensembles d'éléments

## 3.6. Les algorithmes basés sur le modèle « Sliding windows »

### 3.6.1. Algorithme WSW

L'algorithme WSW est présenté par Pauray et Tsai [30]. WSW a proposé un cadre souple pour le traitement continu des requêtes dans les flux de données. L'intervalle de temps pour les requêtes périodiques est défini comme étant la taille d'une fenêtre d'après *Lin C.H. et al, 200.*

Dans le modèle de fenêtre glissante traditionnelle, la taille d'une fenêtre est généralement définie comme étant un nombre donné de transactions (T). Bien que seules les transactions les plus récentes soient considérées, le temps nécessaire pour couvrir ces transactions (T) peut être long. Si nous ignorons la signification des données à différents intervalles de temps, l'efficacité du résultat de l'extraction peut diminuer. Ainsi, le modèle adopté par WSW présente les deux nouvelles caractéristiques suivantes :

- la taille de la fenêtre est définie par le temps et non pas par le nombre de transactions, le but étant d'éviter que les intervalles couvrant les transactions T à différents moments puissent varier considérablement.
- le nombre de fenêtres considérées pour l'exploration est spécifié par l'utilisateur. De plus, l'utilisateur peut attribuer différents poids à différentes fenêtres en fonction de l'importance des données dans chaque section.

Par exemple, les données à proximité du moment actuel peuvent avoir plus d'influence sur l'exploitation, et pourraient avoir un poids plus élevé. Les poids sont attribués pour la détermination des motifs fréquents. Si le nombre total de support d'un motif est important, et si son support dans la fenêtre il a un poids non élevé, alors il ne peut pas devenir un motif fréquent. L'algorithme prend en compte les poids afin d'extraire des motifs pertinents. Ainsi, le résultat de l'extraction serait plus proche des exigences de l'utilisateur.

### 3.6.2.Algorithme MRFI-SW

L'Algorithme MRFI-SW est présenté par Ren et Li [31], Les transactions sont notées avec une représentation spéciale, qui peut indiquer le nombre et l'ordre des éléments contenus dans les transactions. A l'aide de la propriété d'Apriori, les motifs fréquents peuvent être extraits en traitant les informations de la représentation.

Dans cette représentation (qui sous forme d'une séquence de taille w, où w est le nombre de transactions dans une fenêtre glissante), pour chaque itemsets X appartenant aux transactions de la fenêtre on crée une entrée de la forme (bit, ordre), notée R(x). Si itemset X est dans la i-ième transaction dans la fenêtre glissante courante, la i-ième entrée de R(X)_bit est égale à et l'ordre de cette itemset la transaction est mis dans R(X) _ordre. Sinon (R (X) _bit = R (X) _ordre = 0.

Le processus de création de cette séquence d'éléments pour les transactions de la fenêtre glissante courante est appelé transformation par ordre de bits et on anglais « bit-order transform. » Par exemple, la séquence <$T_1$, (acd)>, <$T_2$, (bce)>, <$T_3$, (abce)>, et <$T_4$, (be)>, la taille de la fenêtre glissante est 3, d'où SW1 = [$T_1$, $T_2$, $T_3$] et SW2 = [$T_1$, $T_2$, $T_3$]. Dans SW1, parce que l'élément a apparaît dans $T_1$ et $T_3$ et est le premier élément dans les deux transactions, alors R (a) est < (1, 1), 0, (1, 1) >. De même, R (c) = < (1, 2), (1, 2), (1, 3)>, R (d) = <(1, 3), 0, 0>, R (b) = < 0, (1, 1), (1, 2)>, et R (e) = <0, (1, 3), (1, 4)>.

### 3.6.2. Algorithme BFI- Steam

L'Algorithme BFI-Steam est présenté par Kun Et al [25]. Il s'agit d'un algorithme d'extraction des itemsets fréquents qui utilise une borne, tel que la borne c'est une valeur qui différencié entre les itemsets fréquents et les itemsets peu fréquents.

L'extraction de tous les itemsets fréquents avec des fréquences précises ce fait à l'aide d'un arbre qui utilise une structure basée sur l'arbre préfixe, appelée BFI-tree, pour maintenir tous les itemsets fréquents dans la fenêtre glissante. Il utilise une fenêtre glissante avec une taille fixe de N transactions, qui contient toujours les N transactions récentes.

BFI-tree surveille les changements de la borne (limite) pour maintenir efficacement la partie sélectionnée des itemsets peu fréquents. Si le statut d'un nœud change, soit de rare à fréquent ou vice versa, ce ci provoque des changements et des mises à jour récursives au niveau de l'arbre BFI-tree. Cela peut également entraîner la création de nouveaux nœuds, qui nécessitent un balayage supplémentaire sur toutes les transactions dans la fenêtre glissante pour calculer leurs fréquences.

Afin de renvoyer des ensembles d'itemsets fréquents précis, toutes les transactions dans la fenêtre glissante doivent être conservées dans une structure très compacte. Cependant, la limite est stable au maximum, ce qui signifie que le coût de mise à jour est très faible. BFI-

tree utilise la propriété d'Apriori [34] dans la construction et les mises à jour pour élaguer les nœuds peu fréquents. Ainsi, tous les enfants d'un nœud peu fréquent doivent être élagués et par conséquent tous les nœuds peu fréquents sont des feuilles dans BFI-tree, et certains enfants d'un nœud fréquent peuvent être peu fréquents et doivent être élagués.

### 3.6.3. Algorithme MineSW

L'Algorithme MineSW est présenté par *Cheng J. et al, 2006* Contrairement à d'autres algorithmes qui utilisent un paramètre d'erreur, ε, pour contrôler la précision de l'extraction, il considère que $\varepsilon = r.\sigma$ où $r$ $(0 \leq r \leq 1)$ c'est le taux de relaxation pour extraire l'ensemble des itemsets fréquents sur chaque unité de temps t dans la fenêtre glissante , ce qui permet d'augmenter ε au détriment de la précision légèrement dégradée, mais améliore considérablement l'efficacité d'exploitation des résultats et économise l'utilisation de la mémoire.

### 3.6.4. Algorithme MFI-TransSW

L'algorithme MFI-TransSW est présenté par Li et al [26] et Li et Lee [27]. Il utilise les séquences de bits pour représenter les items afin de réduire le temps et la mémoire nécessaires pour faire glisser les fenêtres. Pour chaque item X, un SW (SW une Séquence de W bits), notée Bit (X), est construite.

Si un élément est dans la i-ième transaction du SW actuel, le i-ème bit de Bit (X) est mis à 1 sinon, il est défini sur 0. L'algorithme se compose de trois phases :

1. la première phase : lorsque le nombre de transactions générées (jusqu'à présent) dans un flux de données est inférieur ou égale à la taille de la fenêtre glissante W définie par l'utilisateur, dans ce cas chaque élément de la nouvelle transaction entrante est transformé en sa représentation de séquence de bits.
2. la deuxième phase c'est le glissement de fenêtre : après que la fenêtre glissante soit pleine, « (TransSW) c'est la représentation binaire d'une transaction », lorsqu'une nouvelle transaction entrante est ajoutée à la fenêtre glissante, par conséquent la transaction la plus ancienne est supprimée de la fenêtre. L'opération de décalage à gauche est utilisée pour supprimer la transaction ancienne. Après avoir fait glisser la fenêtre, un élément X dans la fenêtre glissante apartaient à la transaction en cours est supprimé si et seulement si sup TransSW = 0.

3. la troisième phase : c'est la phase de génération d'ensembles de motifs fréquents : elle est effectuée uniquement lorsqu' un ensemble d'itemsets fréquents est demandé. Une méthode de niveau est utilisée pour générer l'ensemble des ensembles d'items candidats CIk (ensembles d'itemsets candidats avec k items) et des ensembles d'items fréquents pré-connus FIk-1 (itemsets fréquents avec k-1 items) selon la propriété Apriori [9].

Ensuite, l'opération AND au niveau du bit est utilisée pour calculer le support (le nombre de bits de 1) de ces candidats afin de trouver les k-itemsets fréquents.

### 3.6.5. Algorithme Estwin

L'algorithme Estwin est présenté par Chang et Lee [21]. Il est considéré comme une méthode similaire à Chang et Lee [14] basé sur le mécanisme d'estimation de Manku et Motwani [29] et Lin et al [28], la première fenêtre coulissante sensible au temps a été proposée, qui considère une période fixe comme l'unité de base pour l'extraction. Au cours de laquelle le flux de données de transaction TDS =$T_1$, $T_2$,. . ., $T_N$ est une séquence continue de transactions, où N est l'identifiant de transaction, la dernière transaction entrante $T_N$

Une transaction T = ($TU_{id}$, $T_{id}$ itemset), où $TU_{id}$ est l'identifiant de l'unité de temps, et $T_{id}$ est l'identifiant de la transaction. Une fenêtre glissante sensible au temps (TimeSW) dans le flux de données de transaction est une fenêtre qui glisse vers l'avant pour chaque unité de temps (TU). Chaque unité de temps $TU_i$ est constituée d'un nombre variable | $TU_i$ |, de transactions, et | $TU_i$ | est également appelée la taille de l'unité de temps. Par conséquent, la fenêtre glissante sensible au temps avec W unités de temps est $\text{TimeSW}_{N\_w+1}$ = [$\text{TU}_{N\_w+1}$,$\text{TU}_{N_w+2,}\text{TU}_N$] où $N_{N\_w+1}$ est l'identifiant de l'unité de temps du TimeSW actuelle, et N, c est le $TU_{id}$ de la dernière unité de temps $TU_N$.La fenêtre à un nombre fixe W, l'unités de temps. La valeur W = | $\text{TU}_{N\_w+1}$ | + | $\text{TU}_{N\_w+1}$ | + ..... + | $\text{TU}_N$ | est appelée la taille de la fenêtre glissante sensible au temps et désignée par | TimeSW |.Il n'utilise pas de seuil de support minimum.

Comme presque tous les algorithmes d'exploration de flux de données, il a maintient une structure de données nommée table d'actualisation (DT) pour conserver les itemsets fréquents avec leur nombre de support dans la base individuelle de $\text{TU}_S$ de la fenêtre en cours.

## 4. Classification des algorithmes

Le tableau 2.1 présente les paramètres comparatifs importants qui sont les suivants :

1) Le modèle d'extraction.

2) L'exactitude des résultats.

3) La stratégie de traitement.

4) Les unités de traitement.

5) Les données à traiter (structure de données utilisée).

**Tableau 2. 1 : classification** des algorithmes d'extraction des itemsets fréquents à partir de flux de données [35].

| Model | Algorithme | Résultat Approximative/ Exacte | Basé sur le comptage / Unité de temps | Traitement Séquence /Lot | Toutes / Les transactions récentes |
|---|---|---|---|---|---|
| Landmark | StreamMining | Approximative | Comptage | Séquence | Tous |
| | Lossy Counting | Approximative | Comptage | Séquence | Tous |
| | FP-DS | Approximative | Comptage | Lot | Tous |
| | INC-apriori | Approximative | Temps | Lot | Tous |
| Fading | TUF-Streaming | Approximative | Temps | Lot | Tous |
| | Estdec | Approximative | Comptage | Séquence | Tous |
| Sliding window | MFI-TransSW | Approximative | Comptage | Lot | Récent |
| | WSW | Approximative | Temps | Séquence | Récent |
| | BFI Steam | Exacte | Comptage | Séquence | Récent |

| | | | | | |
|---|---|---|---|---|---|
| | Mine-SW | Approximative | Temps | Lot | Récent |
| | MRFI-SW | Exacte | Comptage | Séquence | Récent |
| | Est win | Approximative | Temps | Séquence | Récent |

Selon le tableau 2.1, le model du traitement du flux du donne et le dernier paramètre sont étroitement liés, dans les deux modèles (landmark) et (fading-time) parce que ils traitent l'ensemble du flux de données à partir du démarrage du système. Tandis que Le modèle de fenêtre glissante (Sliding window) ne traite qu'une partie récente du flux de données.

Dans le modèle sliding windows, tous les algorithmes utilisent un modèle de fenêtre qui se base sur le comptage « basée sur un nombre spécifique », à l'exception des algorithmes MFI-TransSW [26], BFI Steam [25] , MRFI-SW[31] qui considèrent une période fixe de temps comme l'unité de base pour le traitement (c'est-à-dire la fenêtre basée sur le temps) Les fenêtres basées sur le nombre sont faciles à gérer pour les programmeurs et n' pas faciles à spécifier pour les programmeurs.

En revanche, dans les fenêtres basées sur le temps, il est naturel pour les utilisateurs de spécifier une période de temps comme unité de base, mais il est plus difficile de traiter des fenêtres de tailles variables en termes d'octets.

Du point de vue de la performance, il n'y a pas de différence entre l'utilisation des fenêtres basées sur le temps ou sur le nombre Arasu et Widom . Les algorithmes de Lossy Counting[19] , StreamMining [32], Estdec [20] , WSW[30] et BFI-Stea [25] utilisent un mécanisme de traitement de séquence ou (tuple) en anglais, au cours duquel le traitement est effectué transaction par transaction; le reste des algorithmes ils utilisent un traitement par lots.

Dans la plupart des cas, le traitement de chaque transaction sur l'ensemble du flux est moins efficace que le traitement d'un lot de transactions sur l'ensemble du flux. En général, le traitement par lots est plus adapté aux flux de données à haute vitesse selon *Cheng et al 2006* l'algorithme Estwin favorise les itemsets récents en diminuant exponentiellement l'effet des anciennes transactions. Mais en estimant les fréquences des itemssets de leurs sous-ensembles conduisant à une erreur propagée.

Toute les variantes des modèles de (landmark) et (Fading-time) ont la limitation de fournir des réponses approximatives pour les lux du données à long terme et d'améliore leurs besoins de stockage en fonction de l'espace disponible. Tous les algorithmes qu'on a cités précédemment produit des résultats approximatifs, sauf BFISteam, et MRFI-SW, qui produisent des résultats exacts, mais seulement pour une partie récente du flux de données (c.-à-d.). Tous les algorithmes approximatifs adoptent des approches faussement positives,.

L'approche faux-positive utilise un minsup minimal ε, pour réduire le nombre de faux positifs, ce qui permet d'obtenir un résultat plus précis. Cependant, pour obtenir un résultat plus précis, une valeur plus petite de ε doit être définie, ce qui conduit à conserver un plus grand nombre des motifs qui ne sont pas forcément fréquents (sous-FI), ce qui consomme une grande quantité de mémoire.

L'approche faux-négative utilise également un seuil de minsup minimum relaxé, ε; cependant, son utilisation réduit la probabilité de rejeter un item peu fréquent qui peut devenir fréquent plus tard. L'erreur liée au support calculé et les faux résultats d'extraction possibles de la plupart des approches faux-positives sont impliqués par l'équation suivante selon la dérivation de [49] : noter que N, c est le nombre de transaction :

$$\text{Erreur de support} = \text{err}(x) / N \qquad \textbf{(3)}$$

$$\text{Faux résultats} = \{X \mid \text{freq}(X) < \sigma N \leq (\text{freq}(X) + \text{err}(X))\} \qquad \textbf{(4)}$$

Cependant, si Ɛ approche σ, plus de réponses faussement positives seront incluses dans le résultat, puisque tous les sous-FI dont la fréquence calculée est au moins $(\sigma - Ɛ) * N \approx 0$ sont émis alors que la fréquence calculée des sous-FI peut être moins que leur fréquence réelle jusqu'à $\sigma * N$ .Presque tous les algorithmes d'approximation produisent des résultats faussement positifs, pour lesquels l'ensemble des sous-item fréquent conservés est souvent trop important pour obtenir une réponse très précise.

Ainsi, le débit est diminué et la consommation de mémoire est augmentée en raison du traitement d'un grand nombre de sous-FI. Mais pour les algorithmes d'approximation qui produisent des résultats faussement négatifs, cela porte atteinte à l'exactitude. En conséquence, il existe une proportion directe entre la précision et la consommation de

mémoire ; des résultats plus précis nécessitent plus d'utilisation de la mémoire, ce qui augmente le temps de traitement.

## 5. Conclusion

Dans ce chapitre, nous avons vu les différents modèles de traitement de flux de données ainsi que les différents algorithmes d'extraction des motifs fréquent à partir de flux de données.

Nous avons également présenté la façon dont les différents algorithmes traitent le d'extraction des motifs fréquents à partir de flux de données avec une classification des algorithmes selon le modèle utilisé. Le prochain chapitre est consacré à l'extraction incrémentale des motifs fréquents.

Chapitre 3 : Extraction incrémentale des motifs fréquents à partir de flux de données

# 1. Introduction

Les techniques d'exploration de flux de données [36] peuvent être catégorisées sur la base du temps de réponse en deux types : le traitement par lots et le traitement en temps réel. En cas de traitement par lots, les ensembles d'éléments fréquents sont découverts à partir d'ensembles d'éléments statiques, tandis que dans les méthodes en temps réel, les ensembles d'éléments fréquents sont extraits sur place à partir d'ensembles d'éléments dynamiques variables dans le temps. Spécifiquement pour les flux de données, une méthode en temps réel est préférée à une méthode de traitement par lots. A cet effet les méthodes classiques d'extraction des motifs fréquents à partir de flux de données ne sont pas applicable lors d'un traitement en temps réel. Les méthodes incrémentales sont une alternative aux méthodes classiques dans ce contexte.

# 2. Méthodes incrémentales d'extraction des itemsets fréquents

## 3.1. Approche basée sur apriori

Une approche naïve pour résoudre le problème d'exploration incrémentale consiste à ré exécuté l'algorithme d'exploration de données sur la base de données mise à jour. Cependant, elle manque manifestement d'efficacité puisque les résultats antérieurs ne sont pas utilisés à l'appui de la découverte de nouveaux résultats alors que la partie mise à jour est habituellement petite par rapport à l'ensemble des données. Par conséquent, l'efficacité des algorithmes d'exploration incrémentale sont intéressantes De ce fait, ils vont être étudiés en détail dans ce chapitre.

## 3.2. Algorithme base sur apriori

### 3.2.1. Algorithme FUP (Fast Update)

Algorithme FUP (Fast UPdate) présenter par (Cheung et al, 1996) est le premier algorithme proposé pour résoudre le problème de l'exploration incrémentale des règles

d'association. Il gère uniquement les bases de données avec insertion de transaction, mais n'est pas en mesure de gérer la suppression de transaction. Plus précisément, compte tenu de la base de données originale D et de ses itemsets fréquents correspondants L = $\{L_1, ..., L_K\}$. Le but est de réutiliser l'information pour obtenir efficacement les nouveaux itemsets fréquents L '= $\{L'_1,' ..., L'_K\}$ sur la nouvelle base de données D' = D U$\Delta$ +. En utilisant la définition de support et la contrainte de support minimum Smin. Les étapes suivantes sont généralement utilisées dans l'algorithme FUP :

1. Un itemset fréquent X, c'est-à-dire X Є L, devient peu fréquent dans D' si et seulement si X.support D' <Smin.
2. Un élément X rare original, c'est-à-dire que X n'appartient pas à L, peut devenir fréquent dans D' seulement si X.support $\Delta + \geq$ Smin.
3. Si un k-itemset X dont (k-1) – sous ensemble (s) devient peu fréquent, c'est-à-dire, le sous-ensemble est dans Lk-1 mais pas dans Lk'-1, X doit être peu fréquent dans D '[37, 38,39].

Considérons un exemple, prenant une base de données simple avec 9 transactions :

**Tableau 3. 1 :** base de données

| | TID | Itemset |
|---|---|---|
| D | T1 | A,B,C |
| | T2 | A,F |
| | T3 | A,B,C,E |
| | T4 | A,B,D,F |
| | T5 | C,F |
| | T6 | A,B,C |
| | T7 | A,B,C E |
| | T8 | C,D,E |
| | T9 | A,B,D,E |

**Tableau 3. 2 : Un** ensemble d'items candidats C1.

| Item | Support |
|---|---|
| A | 7/9 |
| B | 6/9 |
| C | 6/9 |
| D | 3/9 |
| E | 4/9 |
| F | 3/9 |

**Tableau 3. 3 :** Un ensemble d'objets fréquents L1.

| Item | Minimum Support |
|---|---|
| A | 7/9 |
| B | 6/9 |
| C | 6/9 |
| E | 4/9 |

**Tableau 3. 4 :** ensembles de deux d'éléments candidats C2.

| Item | Support |
|---|---|
| AB | 6/9 |
| AC | 4/9 |
| AE | 3/9 |
| BC | 4/9 |
| BE | 3/9 |
| CE | 3/9 |

**Tableau 3. 5 :** Ensemble de deux éléments fréquents L2.

| Item | Support |
|---|---|
| AB | 6/9 |
| AC | 4/9 |
| BC | 4/9 |

On obtient un ensemble des itemsets fréquente A, B, C de taille 2 : AB, AC, BC et finalement après avoir répété le même processus pour des itemsets de taille 3. Nous obtenons ABC. Maintenant, après avoir inséré une nouvelle transaction et après avoir supprimé une ancienne transaction, l'ensemble de données mis à jour est D'.

**Tableau 3. 6 :** Base de donnée (D').

| | | TID | Itemset |
|---|---|---|---|
| D' | D | T1 | A,B,C |
| | | T2 | A,F |
| | | T3 | A,B,C,E |
| | | T4 | A,B,D,F |
| | | T5 | C,F |
| | | T6 | A,B,C |
| | | T7 | A,B,C E |
| | | T8 | C,D,E |
| | | T9 | A,B,D,E |
| | D+ | T10 | A,B,D |
| | | T11 | D,F |
| | | T12 | A,B,C,D |

La base de données D est mis à jour, maintient nous effectuons le même processus pour cette nouvelle base de données D+ qui inclus ensemble d'éléments fréquente, L'algorithme FUP fonctionne pour une seule base de données incrémentale [41,42].

### 3.2.2. Algorithme FUP 2

Dans le cas général où des transactions sont ajoutées et supprimées, l'algorithme FUP2 peut fonctionner correctement avec la partie supprimée D- et la partie ajoutée D+ de

l'ensemble de données. Une caractéristique particulière est que les anciens items fréquents k du résultat d'extraction précédent sont utilisés pour diviser l'ensemble candidat Ck en deux parties : Pk = Ck ∩Lk et Qk = Ck -Pk. En d'autres termes, Pk (Qk) est l'ensemble des ensembles d'items candidats qui sont auparavant fréquents (peu fréquents) par rapport à D.

Pour les itemsets candidats en Qk, leurs supports sont inconnus car ils étaient peu fréquents dans la base de données originale D, ce qui pose quelques difficultés pour générer de nouveaux itemsets fréquents. Heureusement, il est noté que si un ensemble d'items candidat dans Qk est fréquent dans D-, il doit être peu fréquent dans D-. Ce jeu d'éléments est en outre identifié comme étant peu fréquente dans la base de données mise à jour D', si elle est également peu fréquente dans D+.

Cette technique permet de réduire efficacement le nombre de jeux d'éléments candidats à vérifier par rapport à la partie inchangée D- qui est généralement beaucoup plus grande que D- ou D + [40,41,43].

Prenant un exemple simple avec la base de donnée (D') Lorsque nous appliquons le même processus, nous obtenons ces éléments dans le (Tableau 3.7) :

**Tableau 3. 7 :** K item-set fréquent.

| | |
|---|---|
| item taille 1 | {A},{B},{C},D} |
| item taille 2 | {A,B},{A,D},{B,D} |
| item taille 3 | {A,B,D} |

## 3.3. Approche basée sur les arbres

Les derniers progrès en technologie permettent de recueillir facilement une grande quantité de données. Ceci, à son tour, pose un problème de maintenance. Plus précisément, lorsque de nouvelles transactions sont insérées dans un BD de base de données existant et / ou lorsque certaines anciennes transactions sont supprimées de BD, il peut être nécessaire de mettre à jour la collection de patterns fréquents (par exemple, ajouter à la collection base de données BD mais sont fréquentes dans la base de données mise à jour BD'). Les Algorithmes FUP [44], FUP2 [45], et UWEP [13] ont été développés pour résoudre ce problème.

## Chapitre 3 : Extraction incrémentale des motifs fréquents à partir de flux de données

Les algorithmes mentionnés avant sont basés sur Apriori, c'est-à-dire qu'ils dépendent du paradigme de génération et de test. Ils calculent des motifs fréquents en générant des candidats et en vérifiant leurs fréquences.

Pour améliorer l'efficacité du processus d'extraction, Han et al. [47,48] ont proposé un cadre alternatif, à savoir un cadre basé sur l'arbre. L'algorithme qu'ils ont proposé dans ce cadre construit une structure préfix-tree étendue, appelée Frequent Pattern tree (FP-tree), pour capturer le contenu de la base de données de transactions. Plutôt que d'utiliser la stratégie de génération et de test des algorithmes basés sur Apriori, un tel algorithme basé sur l'arborescence met l'accent sur la croissance fréquente du motif, qui est une approche restreinte uniquement (c'est-à-dire qui ne génère pas de candidats)

D'un tel cadre à base d'arbre FP [49], certaines études ont été proposées pour améliorer la fonctionnalité (par exemple, l'exploration interactive à base d'arbre FP) et la performance (par exemple, les techniques de segmentation à arbre FP). Alors, qu'en est-il de l'exploration incrémentale basée sur FP-tree ? Rappelons que des algorithmes tels que FUP, FUP2 ont été développés pour gérer l'exploration incrémentale dans le cadre basé sur Apriori.

Ce genre d'algorithme ne peuvent pas être facilement adoptés par l'exploration incrémentale basée sur un arbre FP. Heureusement, certains algorithmes d'extraction incrémentale basés sur des arbres ont été récemment développés. Par exemple, Cheung et Zaiane proposé l'algorithme FELINE avec l'arbre CATS, tandis que Koh et Shieh ont proposé l'algorithme AFPIM.

Dans cette section, nous discutons les trois algorithmes basés sur des arbres FP qui gèrent l'exploration incrémentale :

- L'algorithme CanTree-GTree.
- L'algorithme FELINE avec l'arbre CATS.
- L'algorithme AFPIM.

### 3.3.1. Algorithm CanTree-GTree (Can –Tree)

L'algorithme CanTree-GTree [50] utilise la technique de fenêtre glissante [54]. Un lot contient un groupe de transactions traité comme une entité unique. Une fenêtre glissante est composée de groupes de transactions «k», où «k» est la taille de la fenêtre. Lorsqu'une fenêtre est pleine, le premier lot est supprimé et un nouveau lot est inséré. Les structures de données suivantes sont utilisées dans cette méthode :

1. CanTree [52,55] est un arbre de base qui stocke efficacement les transactions dans la fenêtre en cours. Un CanTree aura une table d'itemsets (iTable) et une table de dernier nœud de transaction (lTable) associées. Chaque ligne de (iTable) comprend l'identifiant de l'itemset, son support d'une liste des nœuds de cet élément. Dans CanTree Chaque ligne de lTable se compose de l'index du lot et d'une liste des derniers nœuds de transactions dans CanTree.
2. GTree est un arbre de projection, construit à partir du CanTree et il est utilisé pour extraire les motifs fréquents. Un GTree a également un iTable associé à cela

Pour construire un CanTree [52,55] les itemsets appartenant à chaque transaction dans le nouveau lot sont triés de manière canonique, puis ils sont ajoutés à (CanTree). Si chaque élément de données appartenant à une transaction sur un chemin de la racine est le même que chaque nœud sur le chemin, le compte de support du nœud sur le chemin est incrémenté de un. Sinon, un nouveau nœud est créé et ajouté à CanTree en tant qu'enfant du nœud en cours.

Dans GTree pour chaque itemset fréquent est construit en utilisant l'iTable du (CanTree) et en utilisant le lTable, les transactions du lot le plus ancien sont rejetées du CanTree. Les ensembles d'items se produisant fréquemment de chaque élément de données sont trouvés en utilisant le GTree. À partir de chaque GTree d'éléments de données, les ensembles d'éléments fréquents commençant par son nœud racine sont découverts. Les sous GTrees sont construits qui représente récursivement ses sous-problèmes à partir de ceux-ci.

GTree est un arbre qui représente un groupe de sous arbres comme un seul arbre, où les éléments de données dans leurs nœuds de racine sont les mêmes que dans CanTree Une fenêtre avec des K' lots est fournie en entrée pour cet algorithme, et des ensembles d'éléments fréquents dans la fenêtre coulissante courante sont découverts.

Toutes les GT sont évaluées par l'algorithme, car le nombre de prises en charge de leur élément racine sera supérieur ou égal au nombre minimum de prises en charge. Les sous arbres ne sont également traversés que lorsque le compte de soutien de la racine d'un GTree est fréquent. Par exemple, si le nombre de prise en charge de la racine de GTree « A » est supérieur au nombre de prise en charge minimum donné, « A » ou A est un ensemble d'articles fréquent et ses sous-GTrees sont construits. Tous les éléments de données fréquents ou peu fréquents des transactions sont stockés dans l'arborescence de base.

Le CanTree est un peu différent de FP-Tree dans CanTree, les éléments de données d'une transaction sont triés par ordre canonique avant d'être ajoutés à CanTree, alors que dans FP-Tree, l'ordre est basé sur la fréquence. Deux analyses de base de données sont requises dans le cas de l'algorithme de l'arborescence FP, alors qu'une seule analyse de base de données sera requise par CanTree. Ainsi, en temps réel, l'algorithme CanTree-GTree fonctionne mieux et est plus adapté que l'algorithme de FP-growth. [51,53, 56].

### 3.3.2. Algorithme FELINE (CATS-Tree)

Cheung et Zaiane [57] a conçu l'arbre CATS principalement pour l'exploration interactive. L'arborescence CATS étend l'idée de l'arborescence FP pour améliorer la compression du stockage et permet l'exécution de motifs fréquents sans générer les itemsets candidats. Le but est de construire un arbre CATS aussi compact que possible.

Les nouvelles transactions sont ajoutées au niveau de la racine à chaque niveau, les éléments de la nouvelle transaction sont comparés aux nœuds enfants (ou descendants) si les mêmes éléments existent à la fois dans la nouvelle transaction et dans les nœuds enfants (ou descendants), la transaction est fusionnée avec le nœud au niveau de fréquence le plus élevé.

Le reste de la transaction est ensuite ajouté aux nœuds fusionnés et ce processus est répété de manière récursive jusqu'à ce que tous les éléments communs soient trouvés. Tous les éléments restants de la transaction sont ajoutés en tant que nouvelle branche au dernier nœud fusionné. Si la fréquence d'un nœud devient plus élevée que celle de ses ancêtres, il doit alors échanger avec ces derniers pour assurer que sa fréquence soit inférieure ou égale aux fréquences de ses ancêtres.

Considérons l'exemple suivant pour mieux comprendre comment l'arbre CATS est construit :

**Tableau 3. 8 :** Bases de données {BDUBD1UBD2}.[60]

| | | TID | Le contenu |
|---|---|---|---|
| BD | BD original | T1<br>T2<br>T3<br>T4 | {a,d,b, g,e,c}<br>{d,f,b,a,e }<br>{a}<br>{d,a,b} |
| BD1 | 1 ère groupe d'insertion | T5<br>T6 | {a,c,b}<br>{b,c,a,e} |
| BD2 | 2 eme groupes d'insertion | T7<br>T8 | {a,b,c}<br>{a,b,c} |

La figure 3.1 montre l'arbre CATS résultant après l'ajout de chaque transaction. Certaines étapes importantes sont mises en évidence comme suit. Initialement, l'arbre CATS est vide. La transaction t1 = {a, d, b, g, e, c} est alors ajoutée telle quelle. Lorsque la transaction t2 = {d, f, b, a, e} est ajoutée

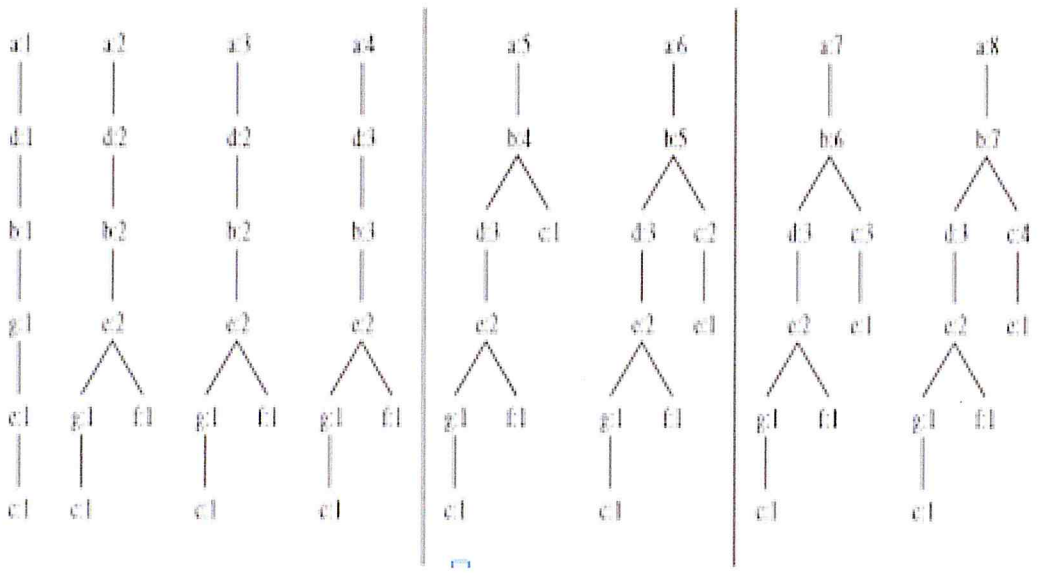


**Figure 3. 1 : CATS** –Tree (l'algorithme FELINE) [60].

Les éléments communs (c'est-à-dire, a, d, b, e) sont fusionnés avec l'arbre existant. Pour ce faire, le nœud (e) est échangé avec son ancêtre (g) (c'est-à-dire, e est "déplacé vers le

haut"). Comme il n'y a pas d'autres éléments communs, l'élément restant de transaction t2 à savoir (f) est ajouté en tant que nouvelle branche à (e).

Les transactions t3 et t4 sont ajoutées de la même manière. Lorsque la transaction t5 est ajoutée, elle est fusionnée avec les éléments communs (a) et (b) le nœud (b) et échangé avec (d) et "déplacé" pour un autre élément commun (c), il ne peut pas être échangé et fusionné.

Autrement, la propriété de l'arbre c'est la fréquence d'un nœud est au moins aussi élevée que la somme des fréquences de ses enfants - serait violée. Par conséquent, c'est ajouté en tant que nouvelle branche (la branche droite) à (b). Les transactions t6, t7 et t8 sont ajoutées de la même manière.

Il est intéressant de noter ce qui suit. Tout d'abord, CATStrees conserve tous les éléments dans chaque transaction. Ceci est différent des arbres FP, qui ne conservent que les éléments fréquents. Deuxièmement, les nœuds dans les arbres CATS sont classés en fonction de la fréquence locale dans les chemins. Par exemple, après l'ajout de t6, (e) est au-dessus de (c) sur la branche de gauche tandis que l'opposé est sur la branche de droite.

Etant donné que l'étape de construction d'arbre ci-dessus ne nécessite qu'un seul balayage de données (c'est-à-dire, construire l'arbre sans connaissance préalable des données), Cheung et Zaiane admettent que leur arbre CATS n'a pas la compression maximale garantie.

De plus, la compression de l'arborescence est sensible à (a) l'ordre des transactions dans la base de données et (b) l'ordonnancement des articles dans chaque transaction. De plus, lors de la gestion de mises à jour incrémentales, l'algorithme FELINE (Fréquent / Large patterns mining avec CATS tree) souffre des problèmes faiblesses décrits ci-dessous.

- Premièrement, la construction d'arbres peut être coûteuse en calcul, car elle recherche des éléments communs et tente de fusionner la nouvelle transaction (la totalité ou une partie de celle-ci) dans un chemin d'arbre existant lorsque chaque transaction est ajoutée. Il vérifie les chemins d'arbre existants un par un jusqu'à ce qu'un fusible soit trouvé. Comme les éléments sont disposés selon leur fréquence locale dans le chemin de l'arbre CATS, un élément (par exemple (e) dans l'exemple peut apparaître au-

dessus d'un autre élément par exemple(c) sur une branche, mais en dessous d'une autre branche. Cela rend une telle recherche très couteuse en termes de CPU.

- Deuxièmement, beaucoup de calculs sont consacrés à la construction d'arbres avec l'espoir que l'arbre est "construit une fois, miné plusieurs" (par exemple, dans l'exploitation interactive où la base de données reste inchangée et seulement le seuil de support minimum sup est modifié interactivement). Cependant, un tel principe ne s'applique pas nécessairement à l'exploitation des motifs fréquents. Spécifiquement, pour une extraction incrémentale, la base de données peut être modifiée par des insertions, des suppressions et / ou des modifications de transactions. Par conséquent, après qu'un arbre a été construit, il peut être extrait seulement une fois.
- Troisièmement, un coût supplémentaire est requis pour l'échange et/ou la fusion de nœuds
- Quatrièmement, puisque les items sont disposés en ordre décroissant de fréquence locale dans l'arbre CATS. Ainsi, lors de la création de bases de données projetées (pendant le processus d'extraction), l'algorithme FELINE doit traverser les deux vers le haut et vers le bas pour inclure des éléments fréquents.

Ceci est différent de l'extraction d'arborescence FP habituelle (par exemple, en utilisant l'algorithme de croissance FP [58]) où il est seulement nécessaire d'effectuer une migration vers le bas. Spécifiquement, le CATStree utilise l'ordre de fréquence locale (par exemple, l'élément (e) est au-dessus de (c) sur la branche gauche mais est inférieur à (c) sur la branche droite dans l'arbre final de l'exemple (1), la traversée vers le bas est nécessaire pour éviter de manquer l'item (c) à la sortie de la branche gauche.

### 3.3.3. Algorithme AFPMI (FP-TREE)

Les deux chercheures Koh et Shieh [59] ont développé l'algorithme AFPIM (Ajuster FP-tree for Incremental Mining). L'idée clé de leur algorithme peut être décrite comme suit. Il utilise la notion originale d'arbres FP-TREE, dans laquelle seuls les éléments "fréquents" sont conservés dans l'arbre.

Ici, un élément est "fréquent" si sa fréquence n'est pas inférieure à un seuil appelé pre-Minsup, ce qui est inférieur au seuil habituel du support de l'utilisateur. Comme d'habitude, tous les éléments "fréquents" sont disposés dans l'ordre décroissant de leur fréquence globale.

Ainsi, les insertions, les suppressions et/ou les modifications de transactions peuvent affecter la fréquence des articles chaque un à son tour, et affecte l'ordre des éléments dans l'arbre. Plus précisément, lorsque la commande est modifiée, les éléments de l'arborescence doivent être ajustés.

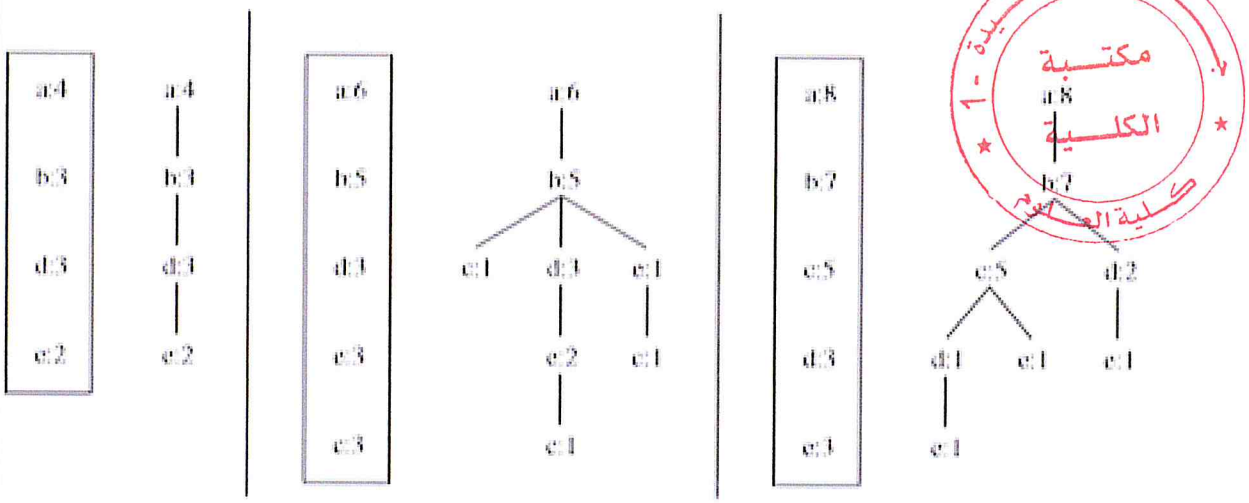


**Figure 3. 2 :** FP-Trees (l'algorithme AFPIM)[60].

L'algorithme AFPIM le fait en échangeant des éléments via le tri à bulles, qui échange récursivement des éléments adjacents. Cela peut être très coûteux en calcul car le tri à bulles doit s'appliquent à toutes les branches affectées par le changement de la fréquence des itemsets. En plus des changements dans l'ordre des éléments, les mises à jour incrémentales peuvent également conduire à l'introduction de nouveaux éléments dans l'arborescence

Cela se produit lorsqu'un élément auparavant rare devient "fréquent" dans la base de données mise à jour. Face à cette situation, l'algorithme AFPIM ne peut plus produire un arbre FP mis à jour en ajustant simplement les éléments dans l'ancien arbre au lieu de cela, il doit ré scanner toute la base de données mise à jour pour construire un nouvel arbre FP Cela peut être coûteux, en particulier pour les grandes bases de données.

Un autre problème de l'algorithme AFPIM est son besoin d'un paramètre d'exploration de données supplémentaire preMinsup, qui est défini à une valeur inférieure au paramètre habituel minsup (le seuil de support minimum) avec ce paramètre supplémentaire, seuls les éléments dont la fréquence rencontre preMinsup sont conservés dans l'arborescence.

Cependant, il est bien connu qu'il est difficile de trouver une valeur appropriée pour un minuteur, quel que soit le choix de l'interlocuteur pour lequel l'usager peut ajuster ou rétablir l'interactivité de manière interactive. Donc, trouver des valeurs appropriées pour minsup et preMinsup peut être encore plus difficile.

## 4. Synthèse

Pour résumer, les algorithmes d'exploration incrémentale basés sur Apriori existants ne peuvent pas être facilement adoptés par l'exploration incrémentale. Des algorithmes basée l'utilisation d'une structure de donnée arbre ont été proposée, la Tableau 3.8 résume ces différents algorithmes basés sur les arbres. Parmi ces algorithmes basés sur l'arbre FP, l'algorithme FELINE avec l'arbre CATS a été principalement conçu pour l'exploitation minière interactive, où le principe de «construire une fois, le mien plusieurs». Cependant, un tel principe ne s'applique pas nécessairement à l'exploitation des motifs fréquents supplémentaires. L'algorithme AFPIM a été proposé pour réduire mais pas pour éliminer. La possibilité de ré analyser la base de données mise à jour.

Tableau 3. 9 : **Algorithmes basés sur une structure de données « Arbre »[60].**

| FELINE / CATS tree | AFPIM/ FP-tree | CanTree-GTree /CanTree |
|---|---|---|
| Un balayage sur bd (c'est-à-dire, insérer, supprimer et / ou des transactions mises à jour) est nécessaire pour maintenir l'arbre CATS | Dans le pire des cas, l'algorithme AFPIM nécessite deux balayages sur BD'= BD ∪ bd pour mettre à jour / reconstruire l'arbre FP | Un seul scan sur bd est requis pour maintenir le CanTree |
| Les éléments sont disposés dans l'ordre décroissant de la fréquence locale dans chaque chemin de l'arbre CATS | Dans l'arbre -FP, les items correspondent-ils à l'ordre indécis de la fréquence (globale) de la BD' | Dans CanTree, les éléments sont arrangés selon un ordre canonique, qui n'est pas affecté par les changements de fréquence |
| Les mises à jour de la base de données peuvent entraîner l'échange et / ou la fusion de nœuds d'arbre | Les mises à jour peuvent causer l'échange (via le tri à bulles), le fractionnement et / ou la fusion de nœuds d'arbre | Les mises à jour de la base de données n'entraînent aucun échange de nœuds d'arbre |

## Chapitre 3 : Extraction incrémentale des motifs fréquents à partir de flux de données

Contrairement à l'algorithme FELINE, l'algorithme AFPIM a été proposé pour l'extraction incrémentale. Spécifiquement, il a été conçu pour produire l'arborescence FP de la base de données mise à jour, dans certains cas, en ajustant l'ancien arbre via le tri à bulles. Cependant, dans de nombreux autres cas, il faut réscanner l'ensemble de la base de données mise à jour BD' afin de construire l'arbre-PF correspondant.

Les algorithmes FELINE (avec l'arbre CATS) et l'algorithme AFPIM (avec l'arborescence FP) souffrent de plusieurs problèmes et faiblesses. Ceux-ci peuvent être résumés comme suit :

- L'algorithme FELINE nécessite une grande quantité de calculs pour la recherche d'éléments communs et des chemins fusionnables lors de la construction d'arbres CATS.
- L'algorithme AFPIM nécessite un paramètre d'exploration de données supplémentaire (à savoir, preMinsup). Trouver une valeur appropriée pour ce paramètre n'est pas facile.
- Les algorithmes FELINE et AFPIM nécessitent beaucoup d'échange, de fusion et de division des nœuds d'arbres, car les éléments dans les arbres sont disposés selon une dépendance de fréquence. Ainsi, lorsque la base de données est mise à jour, les fréquences des éléments peuvent avoir changé. Cela entraîne des changements dans la commande.

Au fur et à mesure que la fenêtre glisse, l'algorithme CanTree-GTree(Can–Tree) ajoute de nouvelles transactions au CanTree sans nécessiter de restructuration. Ici, GTree est utilisé pour construire l'arbre de projection afin de découvrir les ensembles d'items qui se produisent fréquemment. Ainsi, cet algorithme serait plus rapide pour extraire les ensembles d'éléments fréquents complets à partir de flux de données dynamiques. Par conséquent, il fonctionne bien dans l'extraction des motifs fréquents à partir de flux de données en temps réel.

## 5. Conclusion

Dans ce chapitre on a introduit l'extraction des motifs fréquents en utilisent l'aspect incrémentale et les méthodes incrémentales d'extraction des itemsets fréquents avec les deux types d'approche d'extraction l'approche base sur apriori et FP-growth

Chapitre 4 : Solution proposée

# 1. Introduction

Dans ce chapitre, nous proposons un algorithme inspiré de l'algorithme Apriori, avec nos propres contributions qui consistent à proposer une version incrémentale appropriée à l'extraction des itemsets fréquents à partir de flux de données, cet algorithme est appelé INC-Apriori.

## 2. Choix du modèle de traitement des flux de données

On a choisi Le model Landmark qui enregistre toutes les transactions à partir d'un point x dans le temps, appelée point de repère. Ce modèle a été choisi principalement parce que toutes les données sont enregistrées à partir de point de repère, et par conséquent il y'aura pas une perte d'information comme dans les autres modèles. En effet, dans le modèle sliding window les anciennes données sont carrément écrasées par les nouvelles. Et dans le fading time modèle, les anciennes données sont supprimées suivant un paramètre choisi.

Cependant, dans une fenêtre de point de repère (landmark), chaque fois qu'une transaction arrive, si son temps d'arrivée succède le point de repère, elle est ajoutée à la fenêtre sans autre traitement. La taille du flux de données est illimitée. Ainsi, la taille de la fenêtre augmente au fur et à mesure que le temps passe. L'ensemble des Itemsets fréquents (IF) extraits de ces grandes fenêtres est très grand. Le modèle Landmark est préférée uniquement lorsque l'application l'exige en raison des inconvénients susmentionnés.

Comme par exemple, La surveillance des fichiers journaux de serveur Web pour déterminer les modèles d'accès des utilisateurs [61] qui peut aider les propriétaires de sites Web à réorganiser et redessiner les pages web. En effet, une telle application exige l'utilisation des fenêtres (landmark) malgré les problèmes liés à leur maintien.

Dans notre travail, on s'intéresse à ce genre d'application et au fait qu'il y'a6 pas une perte d'information avec ce modèle tout en palliant à ses inconvénient par la proposition d'une méthode incrémentale.

## 3. Algorithme proposé

L'algorithme INC-Apriori est conçu pour extraire des itemsets fréquents à partir de flux de données d'une manière incrémentale. En effet, à chaque requête à l'instant t, l'algorithme prend en entrée l'ancienne structure de motifs non fréquent à t-1 et l'ensemble de nouvelles transaction. Il génère l'ensemble des itemsets fréquents sans rebalayer toute la base de données.

### 1.1. L'algorithme (INC-Apriori)

**Algorithme INC-apriori**

**Entrée :** S// la structure qui contient les éléments non fréquents avec leur fréquence

Minsup // la valeur du Minsup, seuil minimum de support.

D+ // l'ensemble des nouvelles transactions.

-t // moment dans lequel l'extraction des itemsets a été demandée

**Sortie :** IF : ensemble des itemsets fréquents à l'instant t.

**Initialisation :** S=Item-non-fréquente ($D_0$, minsupinf) ; // $D_0$ est l'ensemble des premières transactions après démarrage du système ;

minsupinf est un seuil inférieur de vrai Minsup (.valeur entrer par l'utilisateur)

**INC-Apriori (S, D+, Minsup, t)**

{

IF = **INC-Apriori (S, D+, Minsup, t);**

S=fréquence- Item-non-fréquente (D+, minsupinf) ;

**Sortie : IF** // tous les k-itemsets fréquents à l'instant t.

## 1.2. Scenario

Dans cette section on représente la solution proposée qui consiste à trouver les itemsets fréquents dans un flux du données de manière incrémentale en utilisent le modèle landmark de traitement des flux de données. L'algorithme proposé prend en entrée la base de données ($D^+$) et la valeur du support minimum (Minsup) et il suit ces étapes :

- Premièrement : il extrait les k-itemsets fréquents en utilisent l'approche apriori dans la l'ensemble de données initial ($D_0$).
- Deuxièmement : il extrait les itemsets de taille (1) non fréquents avec leurs fréquences de la base de donnée ($D_0$). Ces (items non fréquent) sont conservés avec leurs fréquences dans une structure compatible.
- Troisièmement : il extrait les k-itemset fréquents dans (D+) après une période du temps définie par l'utilisateur avec la prise en considération les k-itemsets non fréquents extraits dans la base de données (D-).

## 1.3. Extraction des éléments non fréquent

L'extraction des itemsets non fréquents se fait a l'aide d'algorithme apriori lorsque l'algorithme extrait les k-itemsets fréquents de taille 1, on peut distinguer les items non fréquents. Considérons un exemple qui Prendre une base de données simple avec 8 transactions dans le (Tableau 4.1) et un min-sup de 40%.

**Tableau 4. 1 :** base de données (D).

| T-id | Item set |
|---|---|
| T1 | a,b,c,d |
| T 2 | a,b,e,f |
| T 3 | a,b,c,d,e,f |
| T 4 | a,b,g,h |
| T 5 | a,c |
| T 6 | a,b,c,d,e,g |
| T 7 | b,g,h |

| | | |
|---|---|---|
| | T 8 | a,b,c,d |

**Tableau 4. 2 :** Un ensemble d'items candidats C1.

| Item | Minimum support |
|---|---|
| **A** | 6/8 |
| **B** | 6/8 |
| **C** | 5/8 |
| **D** | 3/8 |
| **E** | 3/8 |
| **F** | 2/8 |
| **G** | 2/8 |
| **H** | 2/8 |

Avec une valeur de Minsup de 40% plusieurs items deviennent non fréquent [d,e,f,g,h] ces items sont conservés avec leur fréquence dans une structure S afin d'être utiliser ultérieurement.

**Tableau 4. 3 :** Structure de donner (S).

| Item | D | E | F | G | H |
|---|---|---|---|---|---|
| **Fréquence** | 3 | 3 | 2 | 2 | 2 |

Par la suite on genre les k-itemsets fréquents :

**Tableau 4. 4 :** L'ensemble item fréquent L1.

| Item | Support minium |
|---|---|
| **A** | 6 |
| **B** | 6 |
| **C** | 5 |

**Tableau 4. 5 L'ensemble item fréquent L2**

| Item | Support minium |
|---|---|
| **Ab** | 6 |
| **Ac** | 6 |
| **Bc** | 4 |

L'extraction se fait par niveau en suivant l'algorithme Apriori. Comme que les flux du donnée sont arrivé séquentiellement sur des formats des transactions on a obtenu une nouvelle base de données (D+).

**Tableau 4. 6 :** base de données ($D^+$).

| | | T-id | Item set |
|---|---|---|---|
| D' | D | T 1 | a,b,c,d |
| | | T 2 | a,b,e,f |
| | | T 3 | a,b,c,d,e,f |
| | | T 4 | a,b,g,h |
| | | T 5 | a,c |
| | | T 6 | a,b,c,d,e,g |
| | | T 7 | b,g,h |
| | | T 8 | a,b,c,d |
| | D+ | T9 | a,,g,h,e |
| | | T10 | b,g,h |
| | | T11 | k,j |
| | | T12 | a,b,n,j |
| | | T13 | c,n,j |
| | | T14 | x,z,s |

La solution proposée consiste à extraire tous les itemsets fréquents de la base de données (D') sans refaire le passage sur (D) l'analyse se fait seulement sur (D+). Dans la prochaine section, on va expliquer en détail le travail que nous avons fait.

### 1.3.1. Algorithme Item-non-fréquente (D, minsupinf)

L'algorithme permet d'extraire les items non fréquents de taille 1. Il prend en entrée la base de donnée et la valeur du minsupinf et à l'aide d'un tableau contenant tous les itemsets (C1Table) rempli dans la première phase. L'algorithme génère en sortie un tableau qui contient tous les itemsets non fréquents par rapport à la valeur minsup.

**Item-non-fréquente (D, minsup)**

```
Item-non-fréquente (D, minsup)  {
      Pour (chaque item dans C1Table)
    {
        String Item = C1Table .get(Item);
        Count = CountFrequent(Item);
        if (Count  < MIN_SUPPROT)
        {
                ITEM.add(Item)  ;
        }
    }
Retourne  ITEM ;
}
}
}
```

### 1.2.2. Algorithme fréquence- Item-non-fréquente (D, minsupinf)

L'algorithme permet de tirer les fréquences des items non fréquent de taille 1 elle prend en entrer la base de donnée et la valeur du minsup et à l'aide d'un tableau contient tous les item set ( C1Table) qu' on a remplir dans la premier phase. L'algorithme prend en sortie un tableau des entiers contient tous les fréquences des items non fréquent par rapport a la valeur minsup.

```
fréquence- Item-non-fréquent (D, minsupinf)
Pour (chaque item dans C1Table
{
String Item = C1Table .get(Item)
Count = CountFrequent(Item);
if (Count  < MIN_SUPPROT)
{
fréquence .add(count) ;
}
}
Retourne  fréquence ;
}
```

### 1.2.3. Algorithme d'extraction-Incrémentale (D+, Minsup, structure,temps)

C'est l'algorithme principal du notre travail. IL prend en entrée un nouveau (D+) ce fichier contient tous les transactions qui sont arrivées après une période du temps définit par l'utilisateur a la aide de la (méthode Transférer) , et la valeur du minsup , et la structure S, obtient la liste des itemsets fréquents.

```
INC-Apriori (S, D+, Minsup, t)
{
  D+=Transfer (D, temps) ;
  IF=apriori (D+, Minsup, S) ;
  Return   IF ;
}
```

Continuant avec le même exemple, nous avons les nouvelles transactions (D+) c'est la nouvelle BD sur laquelle nous avons appliqué cette méthode. Dans D, on a obtenu 3 itemsets fréquents (a,b,c) et (ab,ac,bc) de taille 2 et (abc) de taille 3, avec un support min-sup=40% , les autres itemsets non fréquentes sont stoker dans une structure nommer S {d,e,f,g,h}avec leur fréquences dans BD (D) voir la (figure 4.1)

base de donner (D)

| | T-id | Item set |
|---|---|---|
| D | T1 | a,b,c,d |
| | T2 | a,b,e,f |
| | T3 | a,b,c,d,e,f |
| | T4 | a,b,g,h |
| | T5 | a,c |
| | T6 | a,b,c,d,e,g |
| | T7 | b,g,h |
| | T8 | a,b,c,d |

les item candidats L1

| Item | Minimum support |
|---|---|
| a | 6/8 |
| b | 6/8 |
| c | 5/8 |
| d | 3/8 |
| e | 3/8 |
| f | 2/8 |
| g | 2/8 |
| h | 2/8 |

Structure (S)

| Item | D | E | F | g | h |
|---|---|---|---|---|---|
| Fréquence | 3 | 3 | 2 | 2 | 2 |

base de donner (D +)

| | | |
|---|---|---|
| D+ | T9 | a,g,h,e |
| | T10 | b,g,h |
| | T11 | k,j |
| | T12 | a,b,n,j |
| | T13 | c,n,j ,g,h |
| | T14 | x,z,s,n |



**Figure 4. 1 :** Exemple de déroulement INC-apriori.

Après l'application de notre processus, les itemsets qui sont fréquents dans la première phase peuvent devenir non fréquents comme ils peuvent rester fréquents. Par exemple {a,b,c}, était fréquent, l'item {(a)} reste fréquent. Aussi, les itemsets non fréquents peuvent devenir fréquents ou rester non fréquents. Afin de vérifier ces cas, on calcule le nombre d'occurrence dans (D+) pour chaque itemsets et on vérifier s'il existe dans la structure (S). Si il est présent dans la structure S, on rajout son nombre d'occurrence (la fréquence de litemset dans S) à son support. Par exemple, l'item (g) sa fréquence dans la structure S égale 3 et le support de g dans (D+) égale à 2 donc la fréquence de l'item (g) devient a égalé 5. g était pas fréquent dans BD (D) par la suite il devenu fréquent dans (D+). Pareille pour l'item (h).

Les itemsets fréquents de taille K avec un min-sup de 40 % dans base de données (D+) sont :

- Taille 1 : {g,h,n}.
- Ttaille 2 : {g,h}.

## 2. Ajustement des résultats

Comme notre travaille produits des résultats approximatifs le choix de la valeur du support minimum inférieur de vrai Minsup permet de garder les motifs non fréquent dans la requête courante mais probablement ils vont devenir fréquents avec l'arrivée de nouvelles transactions. Par conséquent, le choix adéquat de cette valeur permet d'améliorer la qualité des résultats.

En effet, l'ajustement des résultats se fait par rapport l'algorithme apriori à l'aide d'un algorithme d'ajustement qui permet d'améliorer les résultats. Après quelques tests et comparaison avec les résultats générés par apriori qui est un algorithme exact. On a trouvé que le choix de minsupinf suivant cette formule permet d'améliorer les résultats.

$$minsupinf = (Minsup - 0.1)* DATA\text{-}SIZE$$

« DATA-SIZE » c'est le nombre de transactions .l'ajustement ce fait par rapport la valeur du vrai Minsup .Pour avoir des résultats proche à l' algorithme apriori il faut soustrait une valeur minimal égalé a « 0.1 » . Si on ne soustrait pas cette valeur il y aura une perte des itemsets réellement fréquents. Sinon, si on ajoute une valeur minimale à la valeur Minsup on aura des faux positifs.

## 3. Conclusion

Dans ce chapitre, nous avons présenté notre approche proposée pour l'extraction des itemsets fréquents à partir de flux de données de façon incrémental en utilisant le modèle landmark. Le chapitre suivant est consacré aux tests et validation et l'étude de l'efficacité de l'algorithme proposé INC-Apriori.

# Chapitre 5 : Validation et tests

## 1. Introduction

Après avoir décrit notre solution nous aborderons dans ce chapitre la partie tests et validation de notre application en premier lieu nous présentons l'enivrement de travail et les outils de développement utilisées .Ensuite nous présentons notre jeu d'essai, ainsi que les résultats de des tests.

### 1.4. Présentations de l'environnement de travail

### 1.1. Matériel

Un ordinateur portable Dell avec la caractéristique suivant

- 2 GO RAM
- 512 GO DDR
- Intel Core(TM) i5-2467M CPU 1.6 GHz
- System d'exploitation : Windows 8.1 de 64-bit

### 1.2. Outils de développement

**Java :**

Java est un langage de programmation et une plate-forme informatique qui ont été créés par Sun Microsystems en 1995. Beaucoup d'applications et de sites Web ne fonctionnent pas si Java n'est pas installé et leur nombre ne cesse de croître chaque jour.

Java est rapide, sécurisé et fiable. Défini à l'origine comme un langage, **Java** a évolué au cours du temps pour devenir une technologie, qui intègre une bibliothèque complète pour exécuter ou développer une multitude d'applications. La particularité principale de **Java** est que les applications écrites dans ce langage sont très facilement portables sur plusieurs systèmes d'exploitation tels que l'UNIX, Windows, Mac OS ou GNU/Linux, avec peu ou pas de modifications. C'est la plate-forme qui garantit la portabilité des applications développées en **Java**.

## 2. Résultats et discutions

Dans cette section nous faisons un test sur une partie d'un benchmarks qui contient des transactions de type numérique chaque transaction contient 10 items avec une valeur de Minsup égale « .5 » l'extraction cela était fait après 59 seconde de l'arriver la première transaction.

**Remarque**

- Le MinSup représente la valeur que l'utilisateur saisie dans l'application, le minsupinf est calculé en utilisant la formule présenté dans le chapitre précédant ou bien choisie à partir d'un intervalle.
- On a utilisé une fiche Txt contient 12 transactions avec une taille de 10 item.

```
1 2 3 8 99 56 88 46 68 44
1 2 6 5 55 69 87 23 14 57
1 2 8 9 99 56 88 46 67 78
5 6 9 7 55 68 99 78 48 12
5 6 7 4 12 36 58 98 42 48
5 6 9 7 99 46 79 55 79 87
5 6 7 4 56 99 45 79 48 97
5 6 9 7 23 66 98 78 15 25
1 2 4 3 99 56 89 45 69 45
1 2 8 3 56 19 99 78 59 78
1 2 8 9 89 55 60 40 86 48
```

**Figure 5. 1 Fichier de test base de données (D).**

## 3. Discussion

### 3.1. Résultats

Les résulta de l'extraction après 59 seconde et avec une valeur de minsup égale « .5 » ou elle a donné des item set fréquent de taille 1 taille 2 et taille 3.

```
L'ensemble  fréquents 1-itemsets:
[[88], [66], [67], [45], [23], [89], [46], [68], [25], [69], [48], [97],
l ensemble frequntes 2-itemsets:
[[5, 6], [5, 7], [6, 7]]
l ensemble frequntes 3-itemsets:
[[5, 6, 7]]
```

**Figure 5. 2 Résulta**

L'extraction elle a était fait sur les transactions qui était venu après la seconde 59 de l'arrivée de la première transaction du fichier la figure 5.3 montre le ficher sur lequel nous avons travaillé qui contient 9 transaction.

```
99 56 88 46 67 78
5 6 9 7 55 68 99 78 48 12
5 6 7 4 12 36 58 98 42 48
5 6 9 7 99 46 79 55 79 87
5 6 7 4 56 99 45 79 48 97
5 6 9 7 23 66 98 78 15 25
1 2 4 3 99 56 89 45 69 45
1 2 8 3 56 19 99 78 59 78
1 2 8 9 89 55 60 40 86 48
```

**Figure 5. 3 : Base** de données (D+).

Après la comparaison avec la figure 4.1 on remarque que l'extraction elle a commencé à partir la transaction 3.

## 3.2. Performance

Cette partie discute les performances de notre algorithme en fonction de temps d'exécution et consommation du mémoire par le processeur. On remarque que le temps d'exécution et la consumation du mémoire par CPU augmente lorsque le nombre des

transactions (line) augmentent ou la valeur du Minsup diminuer les figure suivant montre nos résultats.

La figure (5.4) montre nous ces résultats avec une valeur du minsup égale « 0.5 ».

```
le temps d execution 22ms.
 la  consomation du memoire  par  le proocessure  25.0
```

**Figure 5. 4 :** interprétation des résultats (1).

La figure (5.5) montre nous ces résultats avec une valeur du Minsup égale « 0.4 ».

```
le temps d execution 32ms.
 la  consomation du memoire  par  le proocessure  26.9
```

**Figure 5. 5** : interprétation des résultats (2).

### 3.3. Ajustement

L'ajustement c'est une partie essentielle dans notre travaille elle permet d'améliorer la qualité de nos résultats comme notre algorithme c'est un algorithme approximatif.

Il faut faire un ajustement des résultats, les graphes suivant montre le taux de réussite de notre résulta exploiter par rapport l'algorithme apriori en fonction de temps.

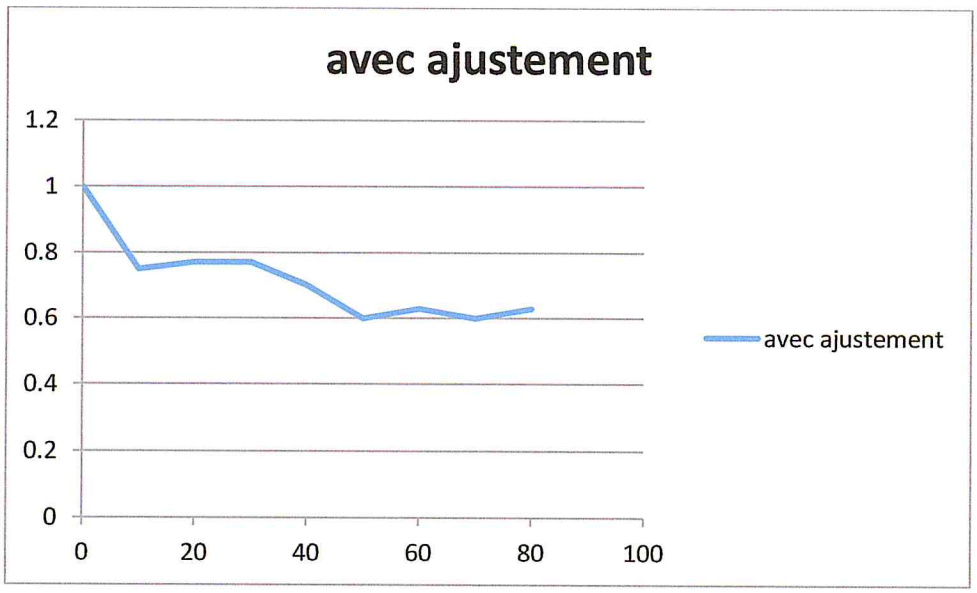



**Figure 5. 6 :** Taux de réussite en fonction de temps en seconde (1).

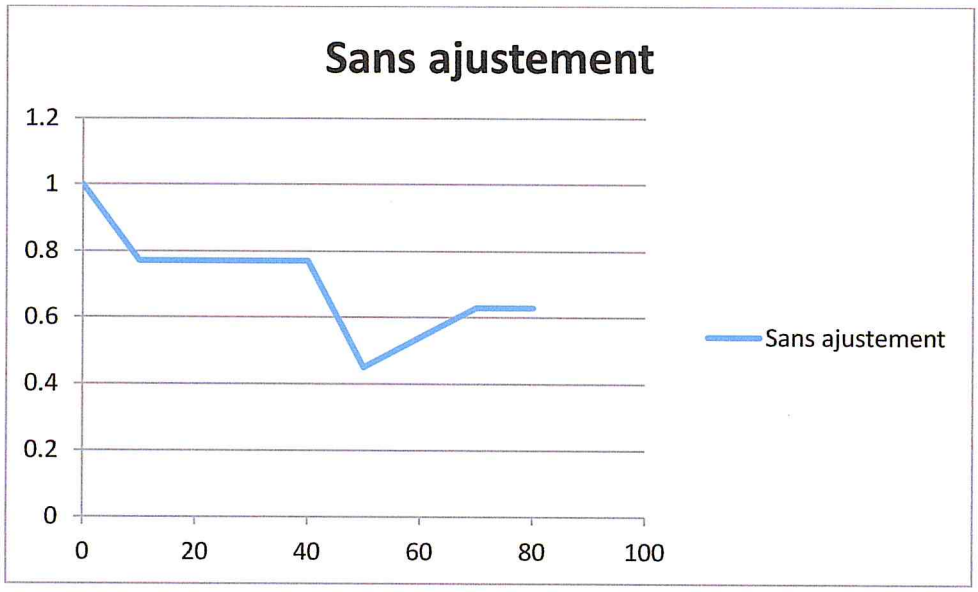



**Figure 5. 7 :** Taux de réussite en fonction de temps en seconde (2).

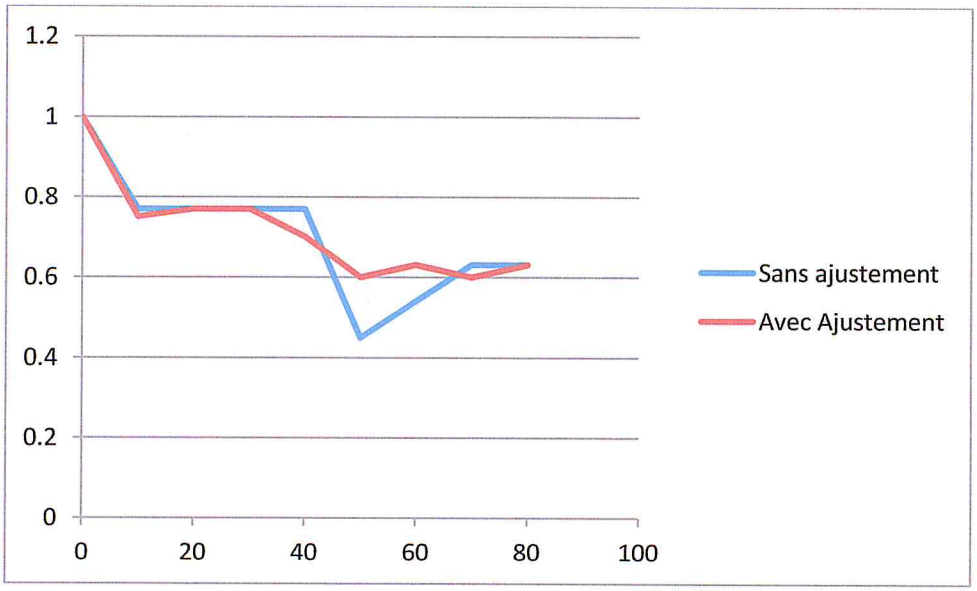



**Figure 5. 8 :** Taux de réussite en fonction de temps en seconde (3).

D'après les trois derniers graphes on a remarqué que l'ajustement permet d'améliorer rapidement la qualité des résultats.

## 4. Conclusion

Dans ce chapitre nous avons présenté notre travail qui consiste en l'implémentation de notre algorithme ou nous avons testé leurs temps d'exécution par rapport à trois paramètres d'entrer bien précis (nombre de transactions, nombre des items,) ou nous avons remarqué que le temps d'exécution augmente quand le nombre paramètres augmentent.

## 1. Conclusion

L'extraction des itemsets à partir du flux de données est une technique qui utilise une variété d'outils d'analyse de données pour découvrir des motifs fréquents et des relations cachés. Ces connaissances découvertes sont très intéressantes, elles sont utilisées dans différents domaines pour faire des prédictions valides et / ou prendre des décisions.

Pour comprendre et cerner la problématique de l'extraction d'itemsets fréquents à partir de flux de données, nous avons consacré le premier chapitre à l'étude des différents algorithmes classiques d'extraction des itemsets fréquents. Dans deuxième chapitre, on a présenté les modèles du traitement de flux de donnée tout en explicitant les algorithmes d'extraction des itemsets fréquents de chaque modèle avec une comparaison selon des critères bien précis. Dans le troisième chapitre, on a introduit les algorithmes incrémentales d'extraction des itemsets fréquents. Le Quatrièmes chapitre présente la solution proposée, l'algorithme INC-apriori basée sur le modèle (landmark). Le dernier chapitre (05) a été consacré aux tests et validation de la solution proposée.

## 2. Perspectives

Les perspectives de ce travail préliminaire sont nombreuses. Tout d'abord, la validation de l'approche proposée sur les grand bases de donnée (BIG DATA).

Il est probablement intéressant de pouvoir créer des méthodes qui lancent plusieurs machines sur plusieurs bases de données en même temps pour extraire des itemsets fréquents à partir des flux de données dans le contexte de Big Data.

Tester par la suite les performances du travail accomplit sur les algorithmes qui traitent le même modèle du traitement du flux de donnée (landmark).

## Bibliographie

[1]https://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/fr_Tanagra_Itemset_Mining.pdf consulter 2 juilllet 2017

[2] Mohammed Javeed Zaki, Srinivasan Parthasarathy, andWei Li. A localized algorithm for parallel association mining. In ACM Symposium on Parallel Algorithms and Architectures, pages 321–330, 1997.

[3] Apriori algorithm (Agrawal, Mannila, Srikant, Toivonen, & Verkamo, 1996) is a data miningmethod which outputs all frequent itemsets and association rules from given data.

[4] 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

[5] Mihaescu, Christian. 2012. "Mining Frequent Itemsets – Apriori Algorithm." *Laboratory Module 8*. http://software.ucv.ro/~cmihaescu/ro/teaching/AIR/docs/Lab8-Apriori.pdf

[6] D. Lee and W. Lee. Finding Maximal Frequent Itemsets over Online Data Streams Adaptively. In Proc. of ICDM, 2005.

[7]Y. Chi, H. Wang, P. S. Yu and R. R. Muntz. Catch the Moment: Maintaining Closed Frequent Itemsets over a Data Stream Sliding Window. In KAIS, 10(3): 265-294, 2006.

[8]M. Zaki. Generating Non-Redundant Association Rules. In Proc. of KDD, 2000.

[9]. M. Zaki and C. J. Hsiao. CHARM: An Efficient Algorithm for Closed Itemset Mining. In Proc. of SDM, 2002.

[10]. J. Wang, J. Han, and J. Pei. CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. In Proc. of KDD, 2003.

[11]. K. Gouda and M. Zaki. Efficiently Mining Maximal Frequent Itemsets. In Proc. of ICDM, 2001.

[12]. T. Calders and B. Goethals. Mining All Nonderivable Frequent Itemsets. In Proc. Of PKDD, 2002.

[13]. J. F. Boulicaut, A. Bykowski and C. Rigotti. FreeSets: a Condensed Representation of Boolean Data for the Approximation of Frequency Queries. In DMKD, 7(1):5-22, 2003.

[14]. J. Pei, G. Dong, W. Zou, and J. Han. Mining Condensed Frequent-Pattern Bases. In KAIS, 6(5): 570-594, 2004.

[15]. D. Xin, J. Han, X. Yan, and H. Cheng. Mining Compressed Frequent-Pattern Sets. In Proc. of VLDB, 2005.

[16]. F. Bonchi and C. Lucchese. On Condensed Representations of Constrained Frequent Patterns. In KAIS, 9(2): 180-201, 2005.

[17]. J. Cheng, Y. Ke, and W. Ng. δ-Tolerance Closed Frequent Itemsets. To appear in Proc. of ICDM, 2006.

[18]. R. Jin and G. Agrawal. An Algorithm for In-Core Frequent Itemset Mining on Streaming Data. In Proc. of ICDM, 2005

[19]Cormode. G. (2007). Fundamentals of Analyzing and Mining Data Streams. WORKSHOP ON DATA STREAM ANALYSIS – San Leucio, Italy - March, 15-16.

[20]Chang J.H. and Lee WS. (2003). Finding recent frequent itemsets adaptively over online data streams. In:Getoor L, Senator T, Domingos P, Faloutsos C (eds) Proceedings of the Ninth ACM SIGKDD international conference on knowledge discovery and data mining, Washington, DC, pp 487–492

[21] Chang J.H and Lee WS. (2003). estWin: adaptively monitoring the recent change of frequent itemsets over online data streams. In: Proceedings of the 2003 ACM CIKM international conference on information and knowledge management, New Orleans, Louisiana, USA, November 2003, pp 536–539.

[22]Giannella, J. Han, J. Pei, X. Yan, and P.S. Yu. (2003). Mining Frequent Patterns in Data Streams at Multiple Time Granularities. in H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.), Next Generation Data Mining, AAAI/MIT.

[23]Hidber C. (1999). Online association rule mining. In: Delis A, Faloutsos C, Ghandeharizadeh S (eds) Proceedings of the ACM SIGMOD international conference on management of data, Philadelphia, Pennsylvania, pp 145–156.

[24] Karp R.M., Shenker S., and Papadimitriou C.H. (2003). A simple algorithm for finding frequent elements in streams and bags. ACM Transactions on Database Systems (TODS), 28(1):51–55.

[25] Kun Li., Yong-yan Wang, Manzoor Ellahi, Hong-an Wang. (2008). Mining Recent Frequent Itemsets in Data Streams. IEEE 2008.

[26] Li H. F., Chin-Chuan Ho, Man-Kwan Shan, and Suh-Yin Lee (2006). Efficient Maintenance and Mining of Frequent Itemsets over Online Data Streams with a Sliding Window. In IEEE SMC.

[27] Li H. F. and Lee S. Y. (2009). Mining frequent itemsets over data streams using efficient window sliding techniques. Sceince Direct, Expert Systems with Applications 36 (2009) 1466–1477.

[28] Lin C.H., Chiu D.Y., Wu Y.H. and Chen A.L.P. (2005). Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window. In Proceedings of 2005 SIAM International Conference on Data Mining.

[29] Manku G., and Motwani R( 2002). Approximate frequency counts over data streams. In Proc. 2002 Int. Conf. Very Large Data Bases (VLDB'02), 346.357.

[26] Naganth E.R. and Dhanaseelan F. R.(2008). Efficient Graph Structure for the Mining of Frequent Itemsets from data Streams. IJCSES International Journal of Computer Sciences and Engineering Systems, Vol.1, No.4.

[30] Pauray S. and Tsai M. (2009). Mining frequent itemsets in data streams using the weighted sliding window model. Elsevier, Expert Systems with Applications.

[31] Ren J. D. and Li K. (2008). Online data Stream mining of recent frequent Itemsets Based On Sliding Window Model. Proceedings of the Seventh International Conference on Machine Learning and Cybernetics, Kunming, 12-15.

[32]Jin R. and Agrawal. G. (2005). An algorithm for in-core frequent itemset mining on streaming data. To appear in ICDM'05.

[33]Chi Y, Wang H, Yu P, Muntz R (2004). Moment: maintaining closed frequent itemsets over a stream sliding window. In: Proceedings of the 4th IEEE international conference on data mining, Brighton, UK, , pp 59–66.

[34] Agrawal R. and Srikant R. (1994). Fast Algorithms for Mining Association Rules. In Proceedings of the 20th International Conference on Very Large Data Bases, pp. 487-499.

[35]Nabil, Hebatallah Mohamed, Ahmed Sharaf Eldin, Mohamed Abd, and El-Fattah Belal. 2013. "Mining Frequent Itemsets from Online Data Streams: Comparative Study." *IJACSA) International Journal of Advanced Computer Science and Applications* 4(7): 117–25. www.ijacsa.thesai.org

[36]F. Bonchi and C. Lucchese. On Condensed Representations of Constrained Frequent Patterns. In KAIS, 9(2): 180-201, 2005.

[37] Improved Association Mining Algorithm for Large Dataset Tannu Arora1, Rahul Yadav2 IJCEM International Journal of Computational Engineering & Management, Vol. 13, July 2011 ISSN (Online): 2230-7893

[38] An Algorithm for Frequent Pattern Mining Based On Apriori Goswami D.N. et. International Journal on Computer Science and Engineering Vol. 02, No. 04, 2010,

[39] An Improved Apriori-based Algorithm for Association Rules Mining 2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery

[40]Mining Dynamic Databases using Probability-Based Incremental Association Rule Discovery Algorithm Journal of Universal Computer Science, vol. 15, no. 12 (2009), 2409-2428 submitted: 15/12/08, accepted: 25/6/09, appeared: 28/6/09 J.UCS

[41]An Algorithm to Discover Calendar- based Temporal Association Rules with Item's Lifespan Restriction Computer Science Department, Graduate School of Engineering Federal University of Rio de Janeiro PO Box 68511, ZIP code: 21945-970 Rio de Janeiro – Brazil

[42]. DARM: Decremental Association Rules Mining Journal of Intelligent Learning Systems and Applications, 2011, Published Online August 2011

[43] Incremental Mining on Association Rules Wei-Guang Teng and Ming-Syan Chen Department of Electrical Engineering National Taiwan University Taipei, Taiwan,

[44] D.W. Cheung, J. Han, V.T. Ng, and C.Y. Wong. Maintenance of discovered association rules inlarge databases: anincremental updating technique. In Proc. ICDE 1996, pp. 106–114.

[45] D.W. Cheung, S. D. Lee, and B. Kao. A general incremental technique for maintaining discovered association rules. In Proc.DASFAA 1997, pp. 185–194.

[46]N.F. Ayan, A.U. Tansel, and E. Arkun. An efficient algorithm to update large itemsets with early pruning. In Proc. SIGKDD 1999, pp. 287–291

[47]J.Han, J.Pei, andY.Yin.Mining frequent patterns without candidate generation. In Proc. SIGMOD 2000, pp. 1–12.

[48] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Mining and Knowledge Discovery, 8(1), pp. 53–87, Jan. 2004.

[49] C.K.-S. Leung. Interactive constrained frequent-pattern mining system. In Proc. IDEAS 2004, pp. 49–58.

[50] Jaein Kim, Buhyun Hwang. Real-time stream data mining based on CanTree and GTree. Information Sciences 2016; 367 368: 512-528.

[51] Syed Khairuzzaman Tanbeer, Chowdhury Farhan Ahmed, Byeong-Soo Jeong, Young-Koo Lee. Sliding window-based frequent pattern mining over data streams. Information Sciences 2009; 179: 3843-3865.

[52]Carson Kai-Sang Leung, Quamrul I. Khan, Tariqul Hoque. CanTree: A Tree Structure for Efficient Incremental Mining of Frequent Patterns. Proceedings of the Fifth IEEE International Conference on Data Mining 2005.

[53]Jiawei Han, Jian Pei, Yiwen Yin, Runying Mao. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. Data Mining and Knowledge Discovery 2004; 8: 53–87

[54]J. Chang, W. Lee. A Sliding window method for finding recently frequent item sets over online data streams. J. Inf. Sci. Eng. 2004; 24 (4): 753–762.

[55] C.K. Leung, Q.I. Khan, Z. Li, T. Hoque. CanTree: a canonical order tree for incremental frequent pattern mining. Knowl. Inf. Syst. 2007; 287–311.

[56] Tri Thanh Nguyen. A Compact FP-tree for Fast Frequent Pattern Retrieval. PACLIC 2013; 27.

[57] W.Cheung andO.R.Za¨ıane. Incremental miningoffrequent patterns without candidate generation or support constraint. In Proc. IDEAS 2003, pp. 111–116.

[58]J.Han, J.Pei, andY.Yin.Mining frequent patterns without candidate generation. In Proc. SIGMOD 2000, pp. 1–12.

[59] J.-L. Koh and S.-F. Shieh. An efficient approach for maintaining association rules based on adjusting FP-tree structures. In Proc. DASFAA 2004, pp. 417–424.

[60] Leung, Carson Kai Sang, Quamrul I. Khan, and Tariqul Hoque. 2005. "CanTree: A Tree Structure for Efficient Incremental Mining of Frequent Patterns." *Proceedings – IEEE International Conference on Data Mining, ICDM*: 274–81.

[61]. Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In PODS, pages 1–16, 2002.