

UNIVERSITÉ DES SCIENCES ET DE LA  
TECHNOLOGIE HOUARI BOUMEDIENE



DATA MINING

---

**Rapport**  
**Algorithme Apriori et PF-growth**

---

*Rédaction:*

MOULAI HASSINA SAFAA

Matricule : 201400007564

HOUACINE NAILA AZIZA

Matricule : 201400007594

M2 SII Groupe:3

*Professeur*

Mme. BABA ALI

November 10, 2018

# Contents

<b>1</b>	<b>Généralités</b>	<b>2</b>
<b>2</b>	<b>Apriori</b>	<b>3</b>
2.1	Concepts de base pour apriori . . . . .	3
2.2	Principe de fonctionnement . . . . .	4
2.3	Pseudo-code . . . . .	5
2.4	Déroulement sur un exemple . . . . .	5
<b>3</b>	<b>FP-Growth</b>	<b>7</b>
3.1	Concepts de base pour FP-Growth . . . . .	7
3.2	Principe de fonctionnement . . . . .	8
3.3	Pseudo-code . . . . .	9
3.4	Déroulement sur un exemple . . . . .	11

# Chapter 1

## Généralités

### Concepts généraux pour la détection de motifs fréquents

Quelques notions relatives à la détection de motifs fréquents doivent préalablement être définies:

#### **un itemset**

un item set (ensemble d'items) est un ensemble comportant des items ou des éléments qui se produisent ensemble. par exemple : un itemset de transactions  $T = (T1=\text{lait,café}, T2=\text{yaourt, crème glacée}, T3=\text{couche bébé} \dots)$

#### **le support**

$\text{supp}(X)$  d'un jeu d'éléments  $X$  est le rapport entre les transactions dans lesquelles un jeu d'éléments apparaît et le nombre total de transactions.

#### **frequent itemset**

Un ensemble d'éléments fréquent est un ensemble d'éléments dont le support est supérieure à une prise en charge minimale spécifiée par l'utilisateur (notée  $L_k$ , où  $k$  est la taille de l'ensemble d'éléments)

# Chapter 2

## Apriori

A priori est un algorithme fondamental proposé par R. Agrawal et R. Srikant en 1994 pour d'éléments fréquents pour les règles d'association booléennes [AS94b]. Le nom de l'algorithme est basé sur le fait que l'algorithme utilise connaissance préalable des éléments fréquents comme nous le verrons plus tard. c'est un algorithme facile à comprendre et très utilisé .

### 2.1 Concepts de base pour apriori

Tout d'abord avant de plonger directement dans le principe de fonctionnement globale de l'algorithme on a à définir quel que notion :

#### **un itemset**

un item set (ensemble d'items) est un ensemble comportant des items ou des éléments qui se produisent ensemble . par exemple : un itemset de transactions  $T=(T1=\text{lait,café},T2=\text{yaourt,crème glacée},T3=\text{couche bébé....})$

#### **le support**

supp (X) d'un jeu d'éléments X est le rapport entre les transactions dans lesquelles un jeu d'éléments apparaît et le nombre total de transactions.

#### **frequent itemset**

Un ensemble d'éléments fréquent est un ensemble d'éléments dont le support est supérieure à une prise en charge minimale spécifiée par l'utilisateur (notée  $L_k$ , où k est la taille de l'ensemble d'éléments)

#### **candidate itemset**

Un groupe d'éléments candidat est un groupe d'éléments potentiellement fréquent (noté  $C_k$ , où k est la taille du groupe d'éléments)

## Apriori propriété

chaque subset (sous-ensemble d'item) d'un fréquent itemset doit être fréquent (condition du support minimale vérifié).

## Opération JOIN (jointure)

Pour trouver le  $L_k$  (frequent itemset de items), on utilise un ensemble de candidate itemset qui sont générés grâce à la jointure de  $L_{k-1}$  avec  $L_{k-1}$  (produit cartésien).

## 2.2 Principe de fonctionnement

Apriori emploie un approche itérative tel que chaque k-itemsets est utilisé pour explorer les (k+1)-itemsets .

les différentes étapes :

1. Exploration de la base de données pour avoir le support de chaque 1-itemset (ensemble d'un seul item).
2. Comparer le support(fréquence) avec le **min\_supp** .
3. Supprimer les 1-itemsets ayant un support inférieur au **min\_supp** générer alors  $L_1$ .
4. Faire une jointure de  $L_{k-1}$  avec  $L_{k-1}$  pour générer les ensembles de candidate k-itemsets.
5. Verifier la propriété APRIORI pour élaguer les k-itemsets qui ne sont pas fréquents.
6. Exploration de la base de données pour avoir le support de chaque candidate k-itemset vérifiant la propriété apriori.
7. Comparer le support de chaque candidate k-itemset avec **min\_supp**
8. Garder que l'ensemble des k-itemsets vérifiant la condition de **min\_supp** et on aura ainsi  $L_k$
9. si  $L_k$  est vide alors pour chaque frequent itemset 1 générer les subsets non vide de 1, et pour chaque s subsets non vide de 1, écrire la règle "s implique(1-s)" si la confiance C de la règle "s implique 1-s" satisfait le **support min de confiance**
10. sinon aller à 4

## 2.3 Pseudo-code

---

**Algorithm 1** APRIORI
 

---

**Input:** D, base de données de transactions, support minimum

**Output:** L, itemset fréquent dans D

```

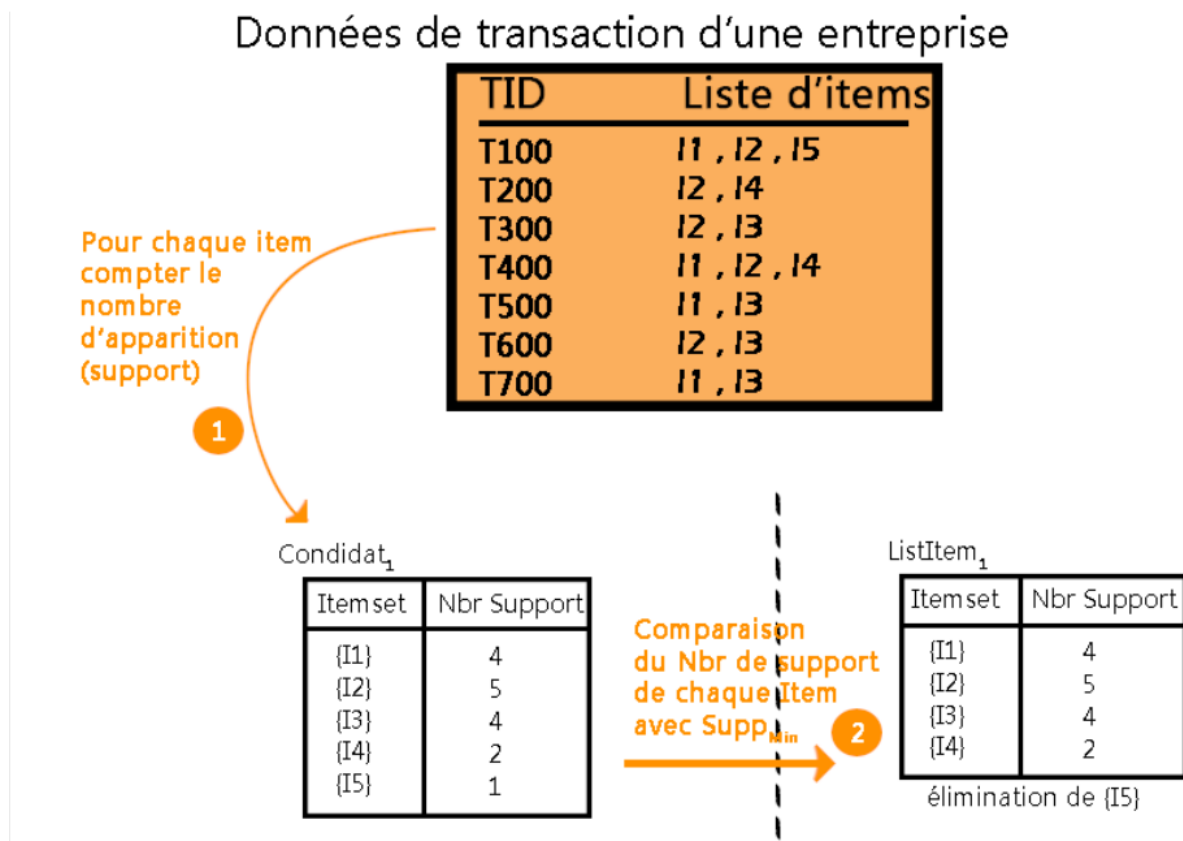
1 L1 = find frequent 1-itemsets(D);
2 for  $k \leftarrow 2; L_{k-1} \neq \emptyset; k++$  do
3    $C_k \leftarrow \text{apriori\_gen}(L_{k-1})$ 
4   for transaction  $t \in D$  do
5     scan D for counts
6      $C_t = \text{subset}(C_k, t)$ ; générer les subsets de t qui sont candidats
7     for candidate  $c \in C_t$  do
8        $c.\text{count}++$ ;
9        $L_k = \{c \in C_k, c.\text{count} > \text{support minimum}\}$ 
9 return  $L = \bigcup_k L_k$ 
  
```

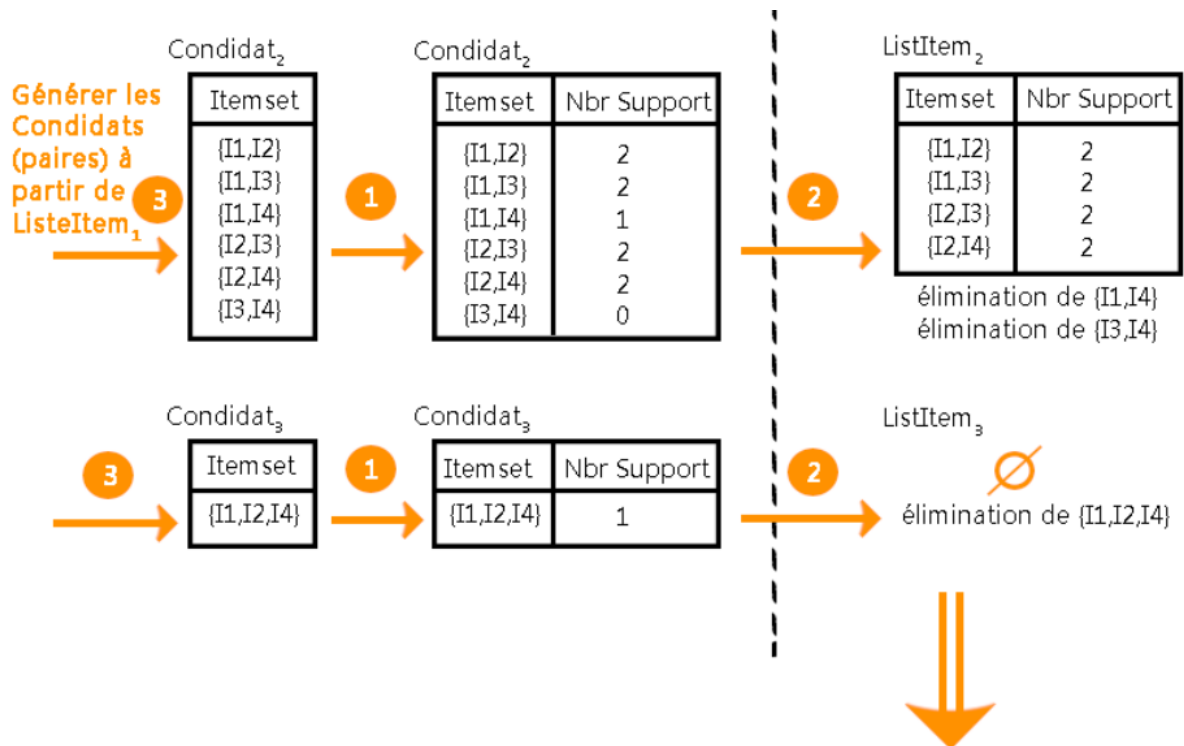
---

## 2.4 Déroulement sur un exemple

Nous prenons comme exemple applicatif les données de transaction d'une entreprise.

Nous y appliquons l'algorithme apriori pour retrouver les motifs fréquents, comme suit:





**Motifs fréquents = Union des  $ListItem_i$**

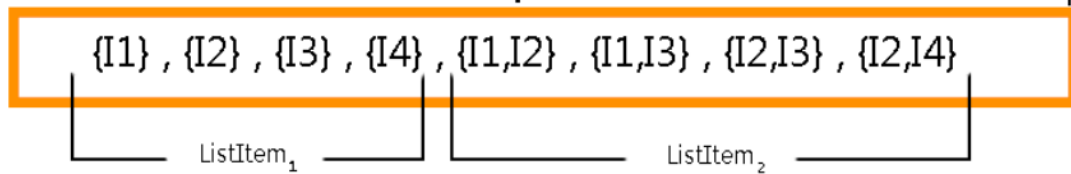


Figure 2.1: Déroulement de l'algorithme Apriori sur un exemple.

# Chapter 3

## FP-Growth

FP-Growth est un algorithme proposé par Han en 2000, c'est une méthode efficace et évolutive pour l'exploitation minière. Utilisation d'une structure d'arborescence de préfixes étendue pour le stockage d'informations compressées et cruciales sur les modèles fréquents nommés arborescence de modèles fréquents (FP-tree).

### 3.1 Concepts de base pour FP-Growth

Tout d'abord avant de plonger directement dans le principe de fonctionnement globale de l'algorithme on a à définir quel que notion :

#### Tree (Arbre)

Un arbre est une structure constituée de nœuds, qui peuvent avoir des enfants (qui sont des nœuds à leur tour).

Dans cet algorithme nos noeuds sont des paires (Item:freq)

Deux types de nœuds ont un statut particulier : les nœuds qui n'ont aucun enfant, qu'on appelle des feuilles, et un nœud qui n'est l'enfant d'aucun autre nœud, qu'on appelle la racine, dans notre FP-Tree c'est la racine null.

#### Path (Chemin)

C'est une suite de nœuds allant de la racine vers une feuille, pour cet algorithme il s'agit d'une suite d'Items constituant une transaction, ces Items sont ordonné des plus fréquents aux moins fréquents.



## 3.2 Principe de fonctionnement

Les différentes étapes :

1. Exploration de la base de données pour avoir le support de chaque 1-itemset (ensemble d'un seul item).
2. Comparer le support(fréquence) avec le **min\_supp** .
3. Supprimer les 1-itemsets ayant un support inférieur au **min\_supp** , et ordonner les autres Items dans l'ordre décroissant de nombre de support, générer alors  $L$ .
4. Initialiser notre  $FB_{Tree}$  a *null*
5. Ordonner les Item de chaque transaction selon l'ordre obtenu dans  $L$
6. Construction de  $FB_{Tree}$  de manière itérative:  
 Pour chaque Transaction insérer les items la formant dans le  $FB_{Tree}$   
 tel que soit on ajoute un nœuds (si celui si n'existe pas), soit on incrémente la fréquence du nœuds (dans le cas contraire).
7. Pour chaque Item de  $L$  toujours dans l'ordre décroissant de fréquence et en utilisant une stratégie de profondeur d'abord:  
 On détermine l'arbre de cet Item (tous les chemins allant de la racine  $\{\}$  a cet Item)
8. si l'on trouve un seul chemin  $P$  constitué d'un ensemble d'Items, alors on génère toutes les combinaisons  $\beta$  de ces Items
9. On récupère les combinaison  $\beta$  ayant un support  $\geq$  **min\_supp** que l'on ajoute a notre liste de motifs fréquents  $\alpha \cup \beta$
10. Sinon (plusieurs chemins) générer les motifs de support  $\geq$  **min\_supp**
11. Construire la base de motif conditionnelle à partir de  $\beta$
12. une fois toutes les combinaisons récupéré, on fait un appel récursif de FP-Growth avec la nouvelle liste  $\beta$  (retour à 8)

### 3.3 Pseudo-code

---

**Algorithm 2** FP-GROWTH
 

---

**Input:** D:base de données de transations , $Supp_{min}$  :support minimum

**Output:** L: itemset fréquent dans D

```

10 L= find frequent 1-itemsets(D);
11 //Create the root of an FP-tree, and label it as "null."
12  $FP_{tree} = \{null\}$ 
13 foreach  $Trans$  in  $D$  do
14     | Sort Items of ( $Trans$ ) according to the order of ( $L$ )
      | foreach  $Item$  in  $Trans$  do
15     | | //  $FP_{tree}$  is the first element
      | | //  $P$  is the remaining list
      | | Insert_tree( $[FP_{tree}|P]$ ,  $Item$ )
16 L = FP-growth-Const( $FP_{tree}, null$ )
17 return  $L$ 
  
```

---



---

**Algorithm 3** PROCEDURE INSERT\_TREE
 

---

**Procedure :** Insert\_tree( $[p—P]$ ,  $Item$ )

```

if  $Item.has\_a\_child(N)$  And  $N.item-name = p.item-name$  then
  | N.freq ++
end
else
  | Create_Node( $N$ )
  | N.freq = 1
  | Link_To_Parent( $Item, N$ )
end
if  $P.empty() == False$  then
  | insert_tree( $P, N$ )
end
return  $P$ 
  
```

---

---

**Algorithm 4** PROCEDURE FP-GROWTH-CONST

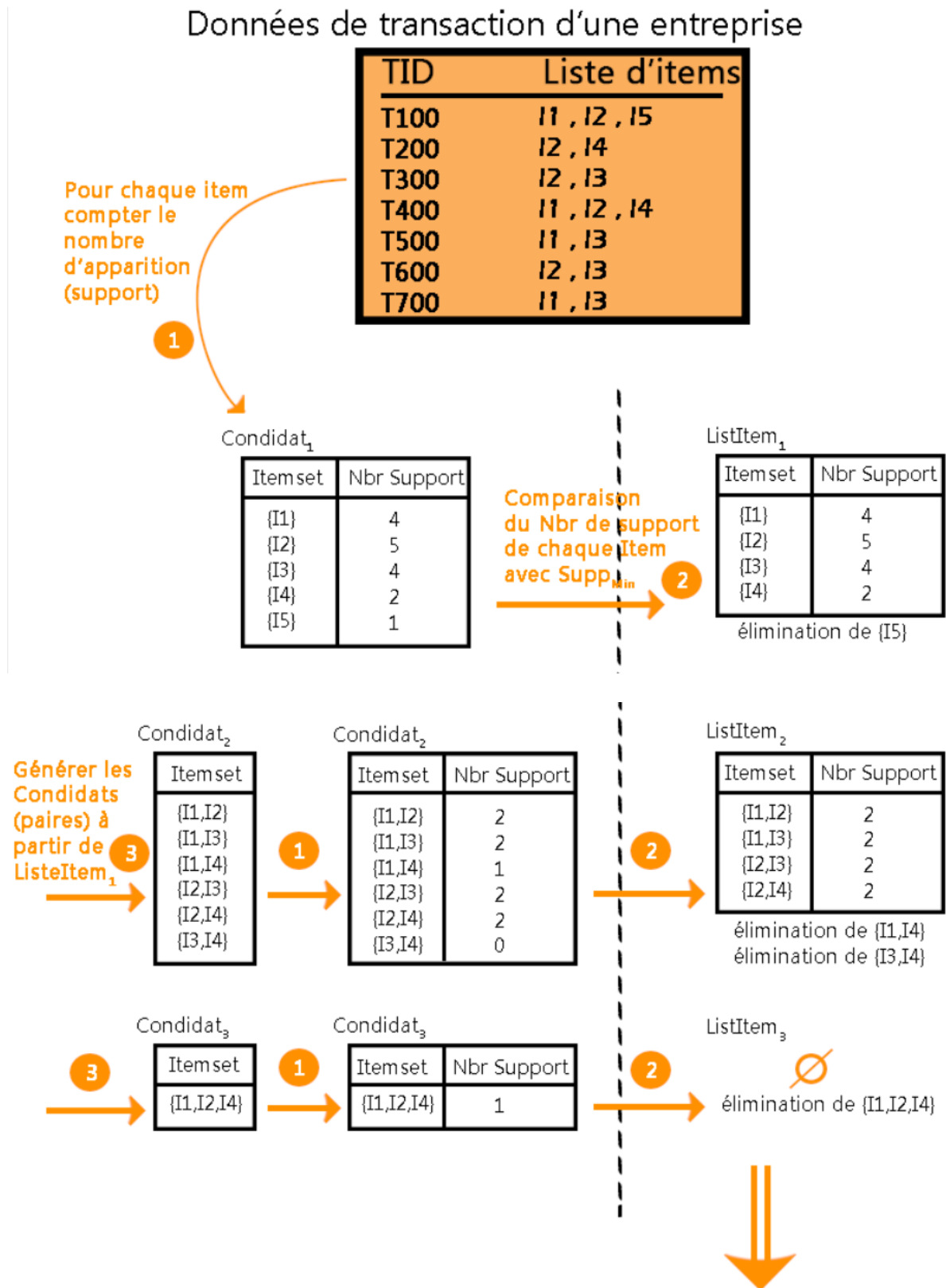
---

**procedure** FP\_growth-Const(Tree,  $\alpha$ )**if** Tree contains a single path  $P$  **then**    **foreach** combination  $\beta$  of the nodes in the path  $P$  **do**        generate pattern  $\beta \cup \alpha$  with support\_count = minimum support count of nodes in  $\beta$ ;**else**    **foreach**  $a_i$  in  $Q$  /\*the header of Tree\*/ **do**        generate pattern  $\beta = a_i \cup \alpha$  with support\_count =  $a_i$ .support\_count        construct  $\beta$  conditional pattern base and then  $\beta$  conditional FP\_tree  $Tree_\beta$         **if**  $Tree_\beta = \emptyset$  **then**            FP\_growth( $Tree_\beta$ ,  $\beta$ )        set( $Q$ ) is the set of patterns generated**return**  $\alpha$ 

---

### 3.4 Déroulement sur un exemple

Nous prenons comme exemple applicatif les données de transaction d'une entreprise.  
Nous y appliquons l'algorithme apriori pour retrouver les motifs fréquents, comme suit:



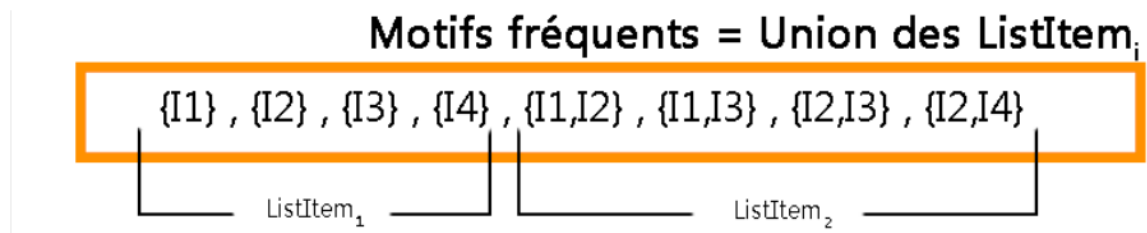


Figure 3.1: Déroulement de l'algorithme Apriori sur un exemple.