
Projet STL

Extraction de motifs graduels fermés fréquents
sous contrainte de temporalité

Composition du trinôme :
MUHANNAD ALABDULLAH
LEA EL ABBOUD
ESMA HOCINI

Encadrants :
Mehdi NAIMA
Lionel TABOURIER

Table des matières

1	Introduction générale	1
1.1	Présentation du projet	1
1.2	Définition	1
2	Algorithme d'extraction des itemsets fréquents	2
2.1	Algorithme Apriori	2
2.1.1	Principe de base	2
2.1.2	Pseudo-code	2
2.1.3	Exemple	3
2.2	Les structures de données	4
2.3	Fonctions et Opérations	5
2.3.1	Opérations sur les Itemsets	5
2.4	Algorithme AprioriTID	5
2.4.1	Principe de base	5
2.4.2	Pseudo-code	5
2.4.3	Exemple	5
3	Etude expérimentale	7
3.1	Paramètres	7
3.2	Génération de données artificiels	7
3.3	Temps d'exécution	8
3.3.1	Variation de la probabilité	8
3.3.2	Variation du support minimum	8
3.4	Consommation en mémoire	8
4	Bilan	9

1 Introduction générale

1.1 Présentation du projet

Pendant leurs opérations quotidiennes, de nombreuses entreprises commerciales collectent une quantité considérable d'informations pour étudier les comportements de leurs clients. Les supermarchés, par exemple, accumulent un nombre important de données sur les achats des consommateurs en évaluant le contenu de leurs paniers, ce qui les aide à améliorer les stocks et la disposition des produits en rayon, ainsi que de créer un système de recommandations pour leur magasin en ligne.

Dans le cadre de notre projet, nous visons ainsi à développer des algorithmes efficace pour l'extraction de motifs graduels fermés fréquents. En priorité, on recherche des motifs fermés, ce qui signifie qu'il n'existe pas de motif plus grand ayant la même valeur du support.

Ce travail nécessitera la mise en place de structures de données et d'algorithmes spécifiques pour traiter efficacement les ensembles de données de grande taille et complexité.

1.2 Définition

Avant de détailler notre approche, il est primordial de se familiariser avec certaines définitions fondamentales.

Definition 1.2.1. Un item fait référence à tout objet appartenant à un ensemble fini d'éléments distincts $\mathcal{I} = \{x_1, x_2, \dots, x_m\}$.

Definition 1.2.2. Un ensemble d'éléments de \mathcal{I} est désigné comme un itemset. Plus précisément, un itemset composé de k éléments est nommé un k -itemset.

Definition 1.2.3. Une transaction est un itemset identifié par un identificateur unique Tid.

Definition 1.2.4. La fréquence (Support) d'un itemset désigne le nombre de fois que cet itemset apparaît dans un ensemble de transactions donnée.

Definition 1.2.5. Le support minimum représente la fréquence minimale requise pour qu'un itemset soit considéré comme fréquent.

Definition 1.2.6. Un itemset est dit fréquent si son support est au-delà du support minimum fixé à priori.

2 Algorithme d'extraction des itemsets fréquents

2.1 Algorithme Apriori

Pour réaliser ce type de problème, nous allons nous appuyer sur l'algorithme Apriori proposé par Agrawal et Srikant en 1994 [1]. Il est reconnu pour sa capacité à identifier des propriétés qui reviennent fréquemment dans un ensemble de données et d'en déduire une catégorisation.

2.1.1 Principe de base

L'algorithme Apriori implémente une approche itérative pour trouver les itemsets fréquents. Tout d'abord, il détermine le support des 1-itemsets en effectuant une première passe sur la base de données. Les itemsets dont le support ne rencontre pas le seuil minimum défini sont qualifiés de non fréquents, Par la suite, un nouvel ensemble des 2-itemsets, désignés comme les itemsets candidats, est formé. Après un autre passage sur la base de données, les supports de ces 2-itemsets candidats sont calculés. Ceux jugés non fréquents sont rejetés. Le processus décrit précédemment est maintenu jusqu'à ce que l'on ne puisse plus générer les itemsets fréquents.

La fonction Apriori-gen(L_{k-1}) est utilisée pour générer des candidats et prend L_{k-1} comme paramètre, produisant ainsi un ensemble de tous les k-itemsets candidats. Cette opération se déroule en deux étapes : d'abord la jonction de L_{k-1} avec L_{k-1} . Ensuite, lors de l'étape d'élagage, tout itemsets c dans C_k tels qu'il existe un sous-ensemble de c de taille $k-1$ qui n'est pas dans L_{k-1} sont supprimés.

2.1.2 Pseudo-code

TABLE 2.1 – Notation

L_k	Ensemble des k -itemsets fréquents de taille k (avec support minimum). Chaque membre de cette ensemble possède deux attributs : i) itemset et ii) support count.
C_k	Ensemble de candidats k -itemsets. Chaque membre de cette ensemble possède deux attributs : i) itemset et ii) support count.

Algorithm 1 Apriori

```
1: Entrées : Base de transaction  $D$ , Support minimum  $S$ 
2: Sorties : Ensemble  $L_k$  des itemsets fréquents
3:  $L_1 \leftarrow \{1\text{-itemsets fréquents}\}$ 
4: for ( $k \leftarrow 2, L_{k-1} \neq \emptyset; k \leftarrow k + 1$ ) do
5:    $C_k \leftarrow \text{Apriori-Gen}(L_{k-1});$  // nouveaux candidats
6:   for all transaction  $t \in D$  do
7:      $C_t \leftarrow \text{Subset}(C_k, t);$ 
8:     for all candidat  $c \in C_t$  do
9:        $c.\text{support} \leftarrow c.\text{support} + 1;$ 
10:    end for
11:  end for
12:   $L_k \leftarrow \{c \in C_k | c.\text{support} \geq S\};$ 
13: end for
14: fin
15: retourner  $\bigcup_k L_k$ 
```

Algorithm 2 Apriori-Gen

```
1: Entrées : Ensemble  $L_{k-1}$  de  $(k - 1)$ -itemsets fréquents
2: Sorties : Ensemble  $C_k$   $k$ -itemsets candidats
3: insérer dans  $C_k$ ;
4: sélectionner  $p[1], p[2], \dots, p[k - 1], q[k - 1];$ 
5: de  $L_{k-1}, L_{k-1};$ 
6: où  $p[1] = q[1], \dots, p[k - 2] = q[k - 2], p[k - 1] < q[k - 1];$ 
7: for all itemset candidat  $c \in C_k$  do
8:   for all sous-ensemble  $s$  de  $c$  de taille  $k - 1$  do
9:     if  $s \notin L_{k-1}$  then
10:       supprimer  $c$  de  $C_k$ ;
11:     end if
12:   end for
13: end for
14: fin
15: retourner  $C_k$ 
```

2.1.3 Exemple

Pour démontrer les étapes successives de l'algorithme Apriori, nous exécuterons un petit exemple en choisissant un support minimum $S = 2$.

Dans la première étape, chaque item de transaction T est un 1-itemset de C_1 . Un premier passage sur la base de donnée D permet d'obtenir le support de chaque 1-itemset, ceux qui atteignent ou dépassent le support minimum sont considérés comme fréquents et conservés dans L_1 . Ensuite, une jointure de $L_1 \times L_1$ est réalisée pour déterminer l'ensemble C_2 . Un deuxième parcours de D est effectué pour trouver le support de 2-itemsets candidats. Les 2-itemsets jugés fréquents sont retenus dans C_2 et on utilise une jointure de $L_2 \times L_2$ pour obtenir les candidats C_3 . Une troisième passe sur D permet de trouver les 3-itemsets fréquents. Les itemsets n'ayant pas le support supérieur ou égal au support minimum sont supprimés pour construire L_3 . De nouveau, on réalise une jointure de $L_3 \times L_3$ pour trouver les candidats C_4 . Cet ensemble est vide, car on n'a plus qu'un seul élément de taille 3.

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

TABLE 2.2 – Base de données des transactions

C_1		L_1	
Itemsets	Support	Itemsets	Support
{1}	2	{1}	2
{2}	3	{2}	3
{3}	3	{3}	3
{4}	1	{5}	3
{5}	3		

TABLE 2.3 – Premier passage sur la base de données pour trouver des ensembles d'éléments uniques

C_2		L_2	
Itemsets	Support	Itemsets	Support
{1 2}	1	{1 3}	2
{1 3}	2	{2 3}	2
{1 5}	1	{2 5}	3
{2 3}	2	{3 5}	2
{2 5}	3		
{3 5}	2		

TABLE 2.4 – Deuxième parcours pour trouver des ensembles d'éléments de taille 2

C_3		L_3	
Itemsets	Support	Itemsets	Support
{2 3 5}	2	{2 3 5}	2

TABLE 2.5 – Troisième parcours pour trouver des ensembles d'éléments de taille 3

2.2 Les structures de données

Pour optimiser l'extraction de motifs fréquents à partir de grandes bases de données transactionnelles, nous avons conçu des structures de données spécifiques pour soutenir les algorithmes que nous avons implémentés.

— Structure des Itemsets (`item_set`)

La structure `item_set` est utilisée pour représenter les itemsets identifiés lors du processus de fouille de données. Elle contient un tableau dynamique pour stocker les items de cet ensemble, leur nombre (k) et un compteur d'occurrences (*count*) utilisé pour enregistrer le nombre d'apparition de cet itemset dans l'ensemble de données.

— Table de hachage

Afin de gérer efficacement les grandes quantités d'itemsets et de faciliter les opérations fréquentes, nous avons intégré une structure de table de hachage, une composante clé pour notre algorithme.

— Structure et fonctionnement de la table de hachage

La table de hachage est composée d'un tableau de pointeurs `pointers`, où chaque entrée pointe vers un `hash_node`, une structure de donnée qui sert de conteneur pour les `item_set`, offrant ainsi un accès aux itemsets et leur données associées.

— Gestion des collisions

La gestion des collisions est un aspect important de la performance de la table de hachage. Lors de l'insertion d'un nouvel itemset, la fonction de hachage calcule un indice qui détermine

où le `hash_node` correspondant devrait être placé dans le tableau de pointeurs. L'indice calculé sera également conservé dans le `hash_node` pour une récupération efficace. En cas de collision, quand plusieurs itemsets sont attribués le même indice de hachage, les `hash_node` sont reliés en liste chaînée grâce au pointeur `next`. Chaque nœud ajouté est connecté au suivant, permettant ainsi de résoudre les collisions tout en préservant l'accès rapide aux itemsets.

— *Avantages de la table de hachage*

L'utilisation de la table de hachage offre une amélioration des performances de notre algorithme, surtout lorsqu'il est appliqué à de grandes bases de données transactionnelles. Cette structure de données facilite les opérations fréquentes, comme les insertions et les recherches, grâce à sa complexité temporelle moyenne qui est constante $\mathcal{O}(1)$.

2.3 Fonctions et Opérations

2.3.1 Opérations sur les Itemsets

Plusieurs opérations sont implémentées afin de manipuler les itemsets dans l'algorithme. Les fonctions clés incluent l'initialisation d'itemsets en créant un nouvel itemset vide (`init_item_set`), l'ajout d'éléments à un itemset existant tout en assurant qu'il n'y a pas de doublons (`add_item_to_item_set`), et la vérification de la présence d'un item dans un itemset (`item_in_item_set`) ou dans la table de hachage (`item_set_in_hash_table`).

2.4 Algorithme AprioriTID

N.B. : L'algorithme AprioriTID est une optimisation de l'algorithme Apriori standard. Son implémentation est en cours de développement et sera donc détaillée dans une version prochaine de ce rapport.

2.4.1 Principe de base

Afin d'optimiser les performances de l'algorithme standard, les mêmes auteurs ont proposé l'algorithme AprioriTID. Basé sur un principe identique à celui d'Apriori, dans AprioriTID à partir de la deuxième passe, la base de données n'est plus utilisée pour calculer les supports des itemsets candidats. Un ensemble $\overline{C_k}$ composé des identifiants de transaction (TID) et des k -itemsets candidats présents dans cette transaction est utilisé. Pour $k = 1$, C_1 correspond à la base de transaction D . Par rapport à l'algorithme Apriori, le principal avantage de AprioriTID réside dans sa capacité à mémoriser les identifiants de transactions qui contiennent les itemsets fréquents au sein de l'ensemble $\overline{C_k}$, permettant ainsi de réduire le nombre de parcours de la base D .

2.4.2 Pseudo-code

Notation 2.4.1. $\overline{C_k}$. Contient des paires de l'identifiant de transaction et du k -itemset candidat présent dans cette transaction, c'est à dire (Tid, C_k)

2.4.3 Exemple

Considérons la base de données dans la figure ci-dessous et supposons que le support minimum S est de 2. L'appel de Apriori-Gen avec L_1 à l'étape 4 donne les itemsets de candidats C_2 . Dans les étapes 6 à 10, nous comptons le support des candidats dans C_2 en itérant sur les entrées dans $\overline{C_1}$ et en générant $\overline{C_2}$. La première entrée dans $\overline{C_1}$ est $\{ \{1\}, \{3\}, \{4\} \}$, correspondant à la transaction 100. Le C_t à l'étape 7 correspondant à cette entrée t est $\{ \{1,3\} \}$ parce que $\{1,3\}$ est un membre de C_2 et à la fois $\{ \{1,3\}, \{1\} \}$ et $\{ \{1,3\}, \{3\} \}$ sont des membres de $t.set-of-itemsets$. L'appel de Apriori-Gen avec L_2 donne C_3 . Faire un passage sur les données avec $\overline{C_2}$ et C_3 génère $\overline{C_3}$. Notons qu'il n'y a pas d'entrée dans $\overline{C_3}$ pour les transactions avec les TID 100 et 400, puisqu'elles ne contiennent aucun des itemsets dans C_3 . Le candidat $\{ \{2\}, \{3\}, \{5\} \}$ dans C_3 s'avère être large et est le seul membre de L_3 . Lorsque nous générons C_4 en utilisant L_3 , il s'avère être vide, et nous terminons.

Algorithm 3 AprioriTID

```
1:  $L_1 \leftarrow \{1\text{-itemsets fréquents}\}$ 
2:  $C_1 \leftarrow \text{Base de donnée } D;$ 
3: for ( $k \leftarrow 2; L_{k-1} \neq \emptyset; k++$ ) do
4:    $C_k \leftarrow \text{Apriori-Gen}(L_{k-1});$  // Nouveaux candidats
5:    $\overline{C}_k \leftarrow \emptyset;$ 
6:   for all entrée  $t \in \overline{C}_{k-1}$  do
7:      $\overline{C}_t \leftarrow \{c \in C_k | (c - c[k]) \in t.\text{set-of-itemsets} \wedge$ 
8:        $(c - c[k-1]) \in t.\text{set-of-itemsets}\};$ 
9:     for all candidats  $c \in \overline{C}_t$  do
10:       $c.\text{count}++;$ 
11:      if ( $\overline{C}_t \neq \emptyset$ ) then
12:         $\overline{C}_k \leftarrow \overline{C}_k \cup \{t.TID, \overline{C}_t\};$ 
13:      end if
14:    end for
15:  end for
16:   $L_k \leftarrow \{c \in \overline{C}_k | c.\text{count} \geq S\}$ 
17: end for
18:  $\text{Answer} \leftarrow \bigcup_k L_k;$ 
```

Base de données

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

 \overline{C}_1

TID	Set-of-Itemsets
100	{ {1}, {3}, {4} }
200	{ {2}, {3}, {5} }
300	{ {1}, {2}, {3}, {5} }
400	{ {2}, {5} }

 L_1

Itemset	Support
{1}	2
{2}	3
{3}	3
{5}	3

 C_2

Itemset
{1, 2}
{1, 3}
{1, 5}
{2, 3}
{2, 5}
{3, 5}

 \overline{C}_2

TID	Set-of-Itemsets
100	{ {1, 3} }
200	{ {2, 3}, {2, 5}, {3, 5} }
300	{ {1, 2}, {1, 3}, {1, 5}, {2, 3}, {2, 5}, {3, 5} }
400	{ {2, 5} }

 L_2

Itemset	Support
{1, 3}	2
{2, 3}	2
{2, 5}	3
{3, 5}	2

 C_3

Itemset
{2, 3, 5}

 \overline{C}_3

TID	Set-of-Itemsets
200	{ {2, 3, 5} }
300	{ {2, 3, 5} }

 L_3

Itemset	Support
{2, 3, 5}	2

FIGURE 2.1 – Application d'AprioriTID sur une base de données

3 Etude expérimentale

Bien que théoriquement la complexité de l'algorithme Apriori puisse atteindre une forme exponentielle dans le pire des cas, son application pratique sur des bases transactionnelles réelles est souvent bien plus gérable. Cette distinction s'explique par l'hétérogénéité des ensembles de données dans des situations réelles, où certains items ou combinaisons d'items tendent à apparaître avec une fréquence nettement plus élevée que d'autres. Ceci permet à l'algorithme de se concentrer sur les combinaisons d'items les plus pertinentes, optimisant ainsi la complexité.

Pour une évaluation approfondie de la complexité et de la consommation en mémoire de l'algorithme Apriori, l'analyse de courbes générées à partir d'un ensemble de données crée aléatoirement se révèle particulièrement utile.

3.1 Paramètres

Définissons d'abord quelques paramètres utiles pour notre étude :

Symbole	Description
N	Nombre totale d'items
$ D $	Nombre de transactions
$ T $	Taille moyenne des transactions
$P(i)$	Probabilité d'appartenance d'un item i dans une transaction

TABLE 3.1 – Paramètres

3.2 Génération de données artificiels

Dans le cadre de notre projet, la génération de données artificiels de transactions est essentielle pour analyser la performance de notre algorithme Apriori¹ sur un large ensemble de données. Cette approche est réalisée par la fonction `generate_baskets_randomly(D,N,P(i))`.

Pour chaque ensemble de données généré, nous avons appliqué l'algorithme Apriori pour identifier les motifs fréquents. Le temps d'exécution de l'algorithme pour chaque ensemble a été enregistré.

Probabilité d'appartenance des items La probabilité $P(i)$ joue un rôle essentiel dans la détermination de l'absence ou de la présence d'un item dans une transaction donnée. La fonction `generate_0_or_1` décide, en fonction d'une probabilité donnée, si un item spécifique est inclus (représenté par 1) ou non (représenté par 0) dans une transaction, en générant un nombre aléatoire et en le comparant à cette probabilité. Cette logique est appliquée à travers la fonction `generate_baskets_randomly`.

Fichier obtenu. Ces données synthétiques sont enregistrés dans un fichier CSV "DataRand.csv" facilitant le traitement ultérieur. Chaque ligne du fichier représente un item présent dans une transaction, identifié par un identifiant unique (**Transaction ID**). Le tableau 3.2 présente un extrait de notre fichier DataRand.csv

1. Dans cette version du rapport nous nous concentrerons sur la performance de l'algorithme Apriori standard

Transaction ID	Items
1	1
1	7
1	12
...	...
10	97

TABLE 3.2 – Exemple tiré de notre fichier DataRand.csv

3.3 Temps d'exécution

Pour évaluer la performance de l'algorithme Apriori, nous adoptons la méthode suivante : un seul paramètre de notre générateur de données est varié à la fois tout en fixant les autres. Cela nous aide à comprendre comment chaque paramètre affecte la performance de l'algorithme.

Les résultats de notre étude sont représentés sous forme graphique pour visualiser la variation du temps d'exécution de l'algorithme Apriori en fonction du paramètre varié.

3.3.1 Variation de la probabilité

En variant le paramètre de probabilité, nous pouvons simuler des scénarios de transactions plus ou moins denses, ce qui nous permet de tester l'efficacité de l'algorithme Apriori.

FIGURE 3.1 – Variation du temps d'exécution de l'algorithme Apriori en fonction de la probabilité d'inclusion des items.

Analyse du graphe.

3.3.2 Variation du support minimum

Cette section sera détaillée dans le prochain rapport.

3.4 Consommation en mémoire

Cette section sera détaillée dans le prochain rapport.

FIGURE 3.2 – Caption of the graph.

4 Bilan

Notre perspective et notre point de vue sur cette étude seront abordés dans la dernière version du rapport. Pour la première phase de l'étude, nous nous sommes concentrés sur l'algorithme Apriori.

Pour la suite, nous incluons une comparaison avec AprioriTID et mènerons davantage d'expériences pertinentes pour enrichir notre analyse.

Bibliographie

- [1] Rakesh Agrawal and Ramakrishnan Srikant. *Fast Algorithms for Mining Association Rules*, 1994.