Sorbonne Université

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

27 avril 2024

Table des matières

1 Introduction générale **1**
1.1 Présentation du projet . . . 1
1.2 Définitions . . . 1

2 Algorithme d'extraction des itemsets fréquents **3**
2.1 Algorithme Apriori . . . 3
2.1.1 Principe de base . . . 3
2.1.2 Pseudo-code . . . 5
2.1.3 Exemple . . . 6
2.2 Les structures de données . . . 7
2.3 Fonctions et Opérations . . . 8
2.3.1 Opérations sur les Itemsets . . . 8
2.4 Complexité . . . 8
2.5 Algorithme AprioriTID . . . 8
2.5.1 Principe de base . . . 8
2.5.2 Pseudo-code . . . 8
2.5.3 Exemple . . . 9

3 Étude expérimentale **11**
3.1 Paramètres . . . 11
3.2 Génération de données artificielles . . . 11
3.3 Temps d'exécution . . . 11
3.3.1 Variation de la probabilité . . . 12
3.3.2 Variation du support minimum . . . 12
3.3.3 Variation du nombre d'items . . . 12
3.3.4 Variation du nombre de transactions . . . 12
3.4 Consommation en mémoire . . . 12
3.5 Perspectives . . . 13

4 Bilan **14**

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, en particulier, accumulent une des données sur les achats en évaluant le contenu des paniers des consommateurs. Cette démarche vise non seulement à améliorer les stocks et à optimiser la disposition des produits en rayon mais également de développer un système de recommandations pour les boutiques en ligne.

Par exemple, si une analyse des paniers de clients révèle que les couches et les bières sont souvent achetées ensemble, cela peut guider les supermarchés dans l'agencement de leurs produits. Placer ces articles côte à côte pourrait augmenter les ventes.

Dans le cadre de notre projet, nous visons ainsi à développer des algorithmes efficaces pour l'extraction de motifs graduels fermés fréquents tels que l'algorithme Apriori et Apriori TID.
En fouille de données, un motif fréquent désigne un ensemble d'éléments d'un jeu de données qui apparaissent en même temps de façon cohérente et dont le nombre d'occurrences (le support) dépasse un seuil donné. Un motif graduel identifie des relations telles que "plus/moins X, plus/moins Y", capturant la manière dont les variables évoluent ensemble dans le temps. Un motif fermé, quant à lui, est un groupe d'éléments qui apparaissent fréquemment ensemble dans les données et qui ne peut pas être agrandie sans réduire sa fréquence d'apparition.

Ce travail, réalisé en langage C, nécessitera la mise en place de structures de données pour traiter efficacement les ensembles de données de grande taille et complexité. Notre démarche sera structurée en plusieurs étapes clés. Tout d'abord, nous définirons les concepts et termes essentiels pour établir une base solide de compréhension. Puis, nous aborderons le développement des algorithmes Apriori et Apriori TID, tout en discutant de leur complexité et des structures de données ainsi que des fonctions utilisées. Enfin, nous procéderons à une étude expérimentale, générant des données artificielles et traçant des courbes expérimentales, afin de comparer l'efficacité de ces algorithmes.

1.2 Définitions

Avant de détailler notre approche, il est primordial de se familiariser avec certaines définitions fondamentales. Nos exemples seront issus de la base de données D suivante :

TID	Items
10	1 2 5
20	1 2
30	3 4 6
40	1 2 3 5
50	3 6

TABLE 1.1 – Base de données D des transactions

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

Les articles en vente dans un magasin sont des items. Dans la base $\mathcal{D}$, les élements de $\mathcal{I} = \{1, 2, 3, 4, 5, 6\}$ sont des items.

Définition 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.*
Une combinaison d'items. Un exemple de $\mathcal{D}$ est {1, 2, 5}, qui est un 3-itemset de la transaction TID 40.

Définition 1.2.3. *Une transaction est un ensemble de données identifié par un identificateur unique Tid.*
Un ensemble d'items achetés ensemble. Un panier au supermarché. Par exemple, l'ensemble d'items {1, 2} avec TID 20 est une transaction.

Définition 1.2.4. *La fréquence (Support, $s(itemset)$) d'un itemset désigne le nombre de fois que cet itemset apparaît dans un ensemble de transactions donnée.*
Si nous prenons l'item 1 dans notre base de données $\mathcal{D}$, il a un support de 3 car il apparaît dans trois transactions : TID 10, 20 et 40.

Définition 1.2.5. *Le support minimum S représente la fréquence minimale requise pour qu'un itemset soit considéré comme fréquent.*
Si le support minimum était de 2 dans notre base de données $\mathcal{D}$, alors l'itemset {1, 2}, apparaissant dans les transactions TID 10 et 40, serait considéré comme fréquent.

Définition 1.2.6. *Un itemset est dit fréquent si son support est au-delà du support minimum fixé à priori.*
Dans $\mathcal{D}$, en prenant un support minimum de 2, l'item 1 serait considéré comme faisant partie d'un itemset fréquent car il apparaît dans trois transactions.

2 Algorithme d'extraction des itemsets fréquents

2.1 Algorithme Apriori

Pour résoudre 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.

2.1.1 Principe de base

Dans le contexte de l'extraction des itemsets fréquents, la propriété d'antimonotonie joue un rôle essentiel dans l'efficacité de la recherche.

Définition 2.1.1. Pour tout itemset I qui est un sous-ensemble de J (noté $I \subseteq J$), $s(I) \geq s(J)$, indépendamment de la base de données T [2].

En d'autres termes, lorsqu'un itemset est étendu en ajoutant un autre élément, son support ne peut augmenter. Cela constitue la base de la propriété Apriori, qui précise que si $s(I) < S$, alors tout sur-ensemble J de I sera également peu fréquent, $s(J) < S$. Par conséquent, on peut éliminer immédiatement le sous-graphe des sur-ensembles de tout itemset non fréquent, réduisant ainsi l'espace de recherche de manière significative. Cette stratégie de réduction est connue sous le nom d'élagage basé sur le support. Prenons la figure 2.1. Par exemple, si l'itemset {c, d, e} est fréquent, toute transaction le contenant inclura également ses sous-ensembles {c, d}, {c, e}, et {d, e} comme le montre la figure 2.2. En revanche, pour un itemset comme a, b qui n'est pas fréquent, tous ses sur-ensembles seront également non fréquents, permettant ainsi l'élagage de l'intégralité du sous-graphe qui les contient, comme l'indique la figure 2.3.

Table 2.1 – Notation

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

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. Grâce à la propriété d'antimonotonie, l'algorithme élimine les itemsets non fréquents à chaque étape itérative. Par la suite, un nouvel ensemble des 2-itemset, désignés comme les itemsets candidats, est formé. Après un autre passage sur la base de données, les supports de ces 2-itemset candidats sont calculés. Ceux 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}) de l'algorithme général 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 trois étapes : l'extraction de tous les éléments possibles de l'ensemble L_{k-1}, puis parcourir cet ensemble d'éléments extraits et L_{k-1} et construire toutes les combinaisons possibles de taille k en essayant d'ajouter à chaque itération, un élément extrait à un ensemble de taille $k-1$ dans L_{k-1}. Ensuite, lors de l'étape d'élagage, tous les 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.

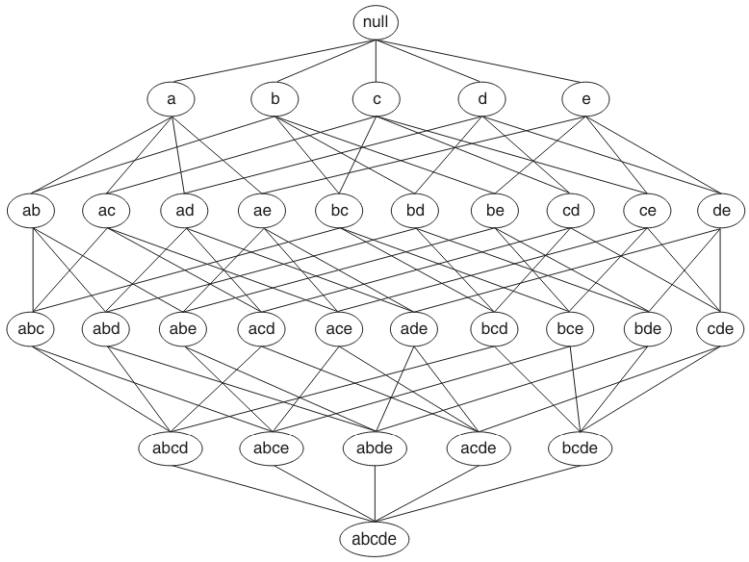

FIGURE 2.1 – Etant donné N items, il existe 2^N candidats itemsets possibles.

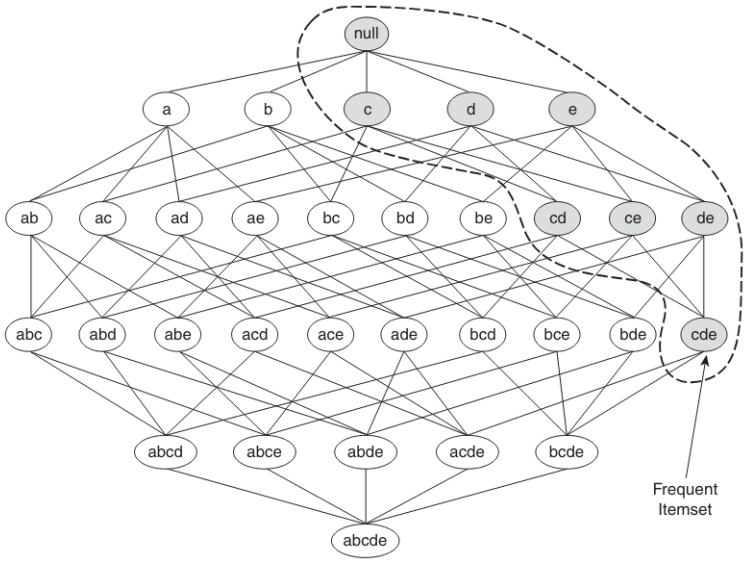

FIGURE 2.2 – Illustration du principe d'Apriori. Si {c,d,e} est fréquent, alors tout sous-ensembles de cet itemset est fréquent.

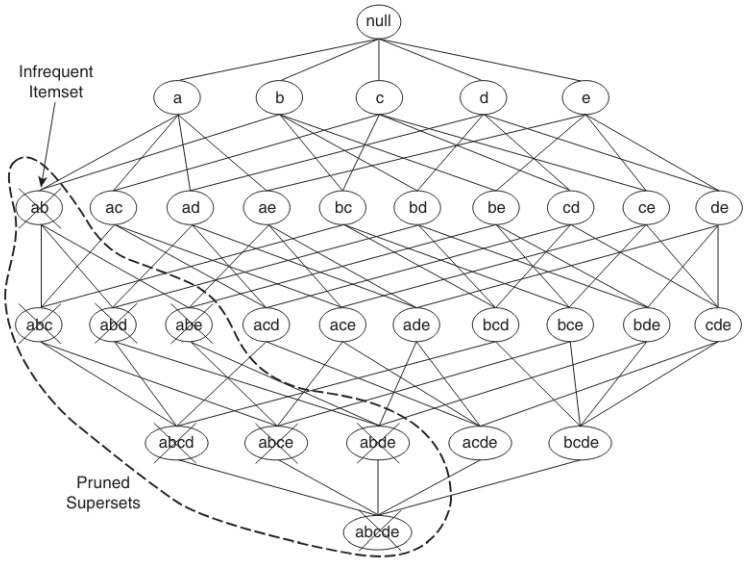

FIGURE 2.3 – Illustration de l'élagage basé sur le support minimum. Si {a,b} n'est pas fréquent, alors tout les sur-ensembles de cet itemset ne sont pas fréquents.

2.1.2 Pseudo-code

Algorithm 1 Apriori

Entrées : Base de transaction $\mathcal{D}$, Support minimum S
Sorties : Ensemble L_k des itemsets fréquents
$L_1 \leftarrow$ {1–itemsets fréquents}
for $(k \leftarrow 2, L_{k-1} \neq \emptyset; k \leftarrow k+1)$ **do**
    $C_k \leftarrow$ Apriori-Gen(L_{k-1}); ▷ nouveaux candidats
    for all transaction $t \in D$ **do**
        $C_t \leftarrow$ Subset(C_k, t);
        for all candidat $c \in C_t$ **do**
            c.support $\leftarrow$ c.support $+ 1$;
        end for
    end for
    $L_k \leftarrow \{c \in C_k | c.\text{support} \geq \text{S}\}$;
end for
fin
retourner $\bigcup_k L_k$

Change the SQL style of the code

Algorithm 2 Apriori-Gen, fonction de l'algorithme général

Entrées : Ensemble L_{k-1} de $(k-1)$-itemsets fréquents
Sorties : Ensemble C_k k-itemsets candidats
for all items $i \in L_{k-1}$ **do**
end for
for all itemset candidat $c \in C_k$ **do**
    for all sous-ensemble s de c de taille $k-1$ **do**
        if $s \notin L_{k-1}$ **then**
            supprimer c de C_k ;
        end if
    end for
end for
fin
retourner C_k

2.1.3 Exemple

Pour décrire les étapes successives de l'algorithme, nous exécuterons Apriori sur l'exemple de la Figure 1.1 en choisissant un support minimum $S = 2$.

Étape 1.
Nous commençons par une première passe sur la base de données pour identifier les candidats C_1, qui sont les 1-itemsets, et calculons leur support. Les items {1}, {2}, {3}, {5}, et {6} se qualifient comme itemsets fréquents formant l'ensemble L_1, puisqu'ils apparaissent au moins deux fois. Nous éliminons l'item {4}, car son support est inférieur à S.

C_1

Itemsets	Support
{1}	3
{2}	3
{3}	3
{4}	1
{5}	2
{6}	2

L_1

Itemsets	Support
{1}	3
{2}	3
{3}	2
{5}	2
{6}	2

TABLE 2.2 – Premier passage sur la base de données pour trouver des ensembles d'éléments uniques

Étape 2.
Avec L_1 établi, nous faisons une deuxième passe pour générer C_2, les 2-itemsets candidats, et calculons à nouveau le support. De cette étape, les paires {1 2}, {1 5}, {2 5}, et {3 6} s'avèrent fréquentes, constituant ainsi l'ensemble L_2.

C_2

Itemsets	Support
{1 2}	3
{1 3}	1
{1 5}	2
{1 6}	0
{2 3}	1
{2 5}	2
{2 6}	0
{3 5}	1
{3 6}	2
{5 6}	0

L_2

Itemsets	Support
{1 2}	3
{1 5}	2
{2 5}	2
{3 6}	2

TABLE 2.3 – Deuxième parcours pour trouver des ensembles d'éléments de taille 2

Étape 3.
En progressant vers C_3, les 3-itemsets candidats, une troisième passe sur la base de données révèle que seul l'itemset {1 2 5} a un support qui satisfait S, ce qui nous donne l'ensemble L_3.

C_3	
Itemsets	Support
{1 2 5}	2

L_3	
Itemsets	Support
{1 2 5}	2

TABLE 2.4 – Troisième parcours pour trouver des ensembles d'éléments de taille 3

Puisqu'aucun itemset de taille supérieure à trois ne peut être formé avec un support suffisant, le processus itératif s'achève, laissant comme résultat final l'union des ensembles L_1 : {{1}, {2}, {3}, {5}, {6}}, L_2 : {{1 2}, {1 5}, {2 5}, {3 6}}, et L_3 : {{1 2 5}}.

Ainsi, le résultat final obtenu est le suivant :
{{1}, {2}, {3}, {5}, {6}, {1 2}, {1 5}, {2 5}, {3 6}, {1 2 5}}

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.

La table de hachage consiste à contenir des itemsets, alors elle représente les ensembles de type L_k et C_k.

— *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.

— *Fonctions de hachage*

Dans notre algorithme, il y a deux fonctions de hachage essentielles, une fonction qui donne une valeur de hachage à un élément `hash_item_id(int id)`, et une autre fonction qui donne une valeur de hachage à un itemset `hash_item_set(item_set s)`. Soit `val(item i)` une fonction qui prend un élément en paramètre et qui renvoie un entier v qui correspond à cet élément, la fonction `hash_item_id(id)` renvoie id mod 100 et la fonction `hash_item_set(s)` renvoie $\left(\sum_{i \in s} \texttt{val(i)}\right)$ mod 100. Notez que dans notre implémentation nous considérons que les éléments sont par défaut représentés par des entiers.

— *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, grace 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`)

Opération	**Complexité**
`init_item_set`	
`add_item_to_item_set`	
`item_in_item_set`	
`item_set_in_hash_table`	

TABLE 2.5 – Complexité des opérations

2.4 Complexité

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, car le pire des cas est très éloigné de la réalité.

2.5 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.

2.5.1 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.

2.5.2 Pseudo-code

Notation 2.5.1. $\overline{C_k}$. Contient des paires de l'identifiant de transaction et du k-itemset candidat présent dans cette transaction, c'est à dire (Tid, C_k)

Algorithm 3 AprioriTID

1: $L_1 \leftarrow$ {1-itemsets fréquents}
2: $C_1 \leftarrow$ Base de donnée D;
3: **for** $(k \leftarrow 2; L_{k-1} \neq \emptyset; k++)$ **do**
4: $C_k \leftarrow$ 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: $L_k \leftarrow \{c \in \overline{C}_k | c.\text{count} \geq \text{S}\}$
17: **end for**
18: Answer $\leftarrow \bigcup_k L_k$;

2.5.3 Exemple

Change the example

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.

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 2.4 – Application d'AprioriTID sur une base de données

3 Étude expérimentale

Pour une évaluation approfondie du temps de calcul 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

TABLE 3.1 – Paramètres

3.2 Génération de données artificielles

Dans le cadre de notre projet, la génération de données artificiels de transactions est essentielle pour analyser la performance de notre algorithme Apriori [1]sur un large ensemble de données. Cette approche est réalisée par la fonction `generate_baskets_randomly(D,N,P(i))` avec $P(i)$ la probabilité d'appartenance d'un item i dans une transaction.

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`).

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.

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

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.

Paramètres utilisées :

— Support = 80

— $|D| = 50$

— $|N| = 200$

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

3.3.2 Variation du support minimum

Le support minimum constitue un critère important pour l'efficacité de l'algorithme Apriori. Dans cette section, nous étudierons l'influence de la variation de ce paramètre sur l'exécution de l'algorithme.

Paramètres utilisées :

— $P(i) = 0.9$

— $|D| = 50$

— $|N| = 200$

FIGURE 3.2 – Variation du temps d'exécution en secondes de l'algorithme Apriori en fonction du support minimum.

3.3.3 Variation du nombre d'items

Paramètres utilisées :

— Support = 80

— $P(i) = 0.9$

— $|D| = 50$

FIGURE 3.3 – Variation du temps d'exécution en secondes de l'algorithme Apriori en fonction du nombre d'items.

3.3.4 Variation du nombre de transactions

Paramètres utilisées :

— Support = 80

— $P(i) = 0.9$

— $|N| = 200$

FIGURE 3.4 – Variation du temps d'exécution en secondes de l'algorithme Apriori en fonction du nombre de transaction.

3.4 Consommation en mémoire

Cette section sera détaillée dans le prochain rapport.

3.5 Perspectives

Le modèle de génération de données aléatoires est considéré comme irréaliste car il ignore deux aspects importants : d'abord, il suppose que la probabilité qu'un article soit présent dans une transaction est indépendante de celle des autres, ce qui est contraire à la réalité commerciale où les articles sont souvent achetés en fonction de leurs associations dues aux préférences des clients ou d'autre raisons. Ensuite, ce modèle échoue à refléter le fait que la probabilité d'achat varie d'un article à l'autre, étant influencée par la popularité de l'article, et d'autres facteurs. Ainsi, ces simplifications empêchent le modèle de refléter fidèlement la complexité des données transactionnelles dans le monde réel.

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 inclurons une comparaison avec AprioriTID et mènerons davantage d'expériences pertinentes pour enrichir notre analyse. Parmi les tâches restantes à réaliser, nous prévoyons de :

1. Comparer des ensembles de données réels à des ensembles de données artificiels, tout en s'assurant que le nombre d'items et le nombre de paniers soient identiques dans les deux cas, pour une évaluation précise de la performance de nos algorithmes.
2. Étudier la complexité de nos algorithmes en analysant les facteurs influençant les performances, tels que le support. Nous envisageons de calculer le nombre de combinaisons possibles à partir d'un nombre de candidats C connu pour un niveau k donné (itemsets fréquents C_k), afin de déterminer la limite supérieure de combinaisons et d'évaluer le nombre d'opérations nécessaires pour passer de C_k à C_{k+1}
3. Nous prévoyons d'utiliser le forage de données pour détecter des séquences temporelles dans de vastes ensembles de données, une méthode fréquemment employée en data mining. Notre objectif est d'identifier des motifs temporels récurrents dans des données environnementales, en mettant en lumière des variations corrélées. Nous nous concentrons sur la recherche de motifs fermés, ce qui nécessite une adaptation de l'algorithme initial.

Bibliographie

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

[2] Christian Borgelt. *Frequent item set mining*, 2012.