RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

FACULTÉ DES SCIENCES ET DE LA TECHNOLOGIE
DÉPARTEMENT DES MATHÉMATIQUES ET D'INFORMATIQUE

M É M O I R E

Pour obtenir le titre de

Master en Informatique

Spécialité : Systèmes Intelligents pour l'Extraction de Connaissances (SIEC)

Présenté par

Khadidja KHENINE **et** Meriem GHADA

Une implémentation Java de l'algorithme MAFIA

Soutenu publiquement le 27/06/2018

Jury :

Président :	Slimane BELLAOUAR	MC	Univ. Ghardaia
Examinateur :	Yousef MAHDJOUB	MA	Univ. Ghardaia
Examinateur :	Khald KCHIDA	MA	Univ. Ghardaia
Encadreur :	Slimane. OULAD-NAOUI	MC	Univ. Ghardaia

Remerciements

Nous aimerions tout d'abord remercier notre dieu qui nous a réconciliées à réaliser ce travail et nous lui sollicitons la réussite au futur (InchAllah).

La première personne que nous tenons à remercier est notre encadrant Mr.Slimane Ouled naoui, pour l'orientation, la confiance et la patience qui ont constitué un apport considérable sans lequel ce travail n'aurait pas pu être mené au bon port. Qu'il trouve dans ce travail un hommage vivant à sa haute personnalité.

Nous tenons à remercier également les membres du Jury pour avoir accepté d'examiner ce travail.

Nos remerciements s'étendent également à tous nos enseignants durant les années des études.

À monsieur K.Boumama pour ses conseils et Ses idées qui nous ont aidées à accomplir ce travail

Enfin, nous tenons à remercier tous ceux qui, de près ou de loin, ont contribué à la réalisation de ce travail.

Dédicaces

Je dédie ce travail à:

À mes parents .Aucun hommage ne pourrait être à la hauteur de l'amour
Dont ils ne cessent de me combler. Que dieu leur procure
Bonne santé et longue vie.

À mes frères et ma petite sœur Aicha.

À ma binôme *Meriem* et toute la famille GHADA.

À ma grande famille.

À mes proffésseurs.

À tous les membres de ma promotion2017/2018.

À toutes les personnes que je connais et que je n'ai pas citées.

KHENINE, Khadidja

Dédicace

Je dédie ce travail à:

À mes parents .Aucun hommage ne pourrait être à la hauteur de l'amour Dont ils ne cessent de me combler. Que dieu leur procure. Bonne santé et longue vie.

À mes chers frères et sœurs

A ma grand-mère et à mon grand-père

Aux petits poussins (Aboud, Ahmed, Iyad, Bouchra, Adam)

À ma grande famille.

À tous mes collègues de la promotion 2017/2018 sans exception,

A tous mes collègues de travail (CHF)

À toutes les personnes que je connais et que je n'ai pas citées.

GHADA. Meriem

À nos amies. Chacun avec son nom : Rachida, Romaissa, Kawtar, Fadila, Amal et Hannan.

Résumé

L'extraction des motifs fréquents est une tâche importante en fouille de données. Durant les deux dernières décennies de nombreux algorithmes ont été introduits pour résoudre ce problème très populaire. Toutefois, la nature combinatoire et exponentielle de ce problème rend l'énumération totale des motifs très difficile voire infaisable. Par conséquent, d'autres solutions visent l'extraction d'une représentation compacte de l'ensemble de motifs.

L'objectif de ce mémoire est d'étudier l'approche d'extraction des motifs fréquents maximaux, qui constitue une représentation compressée sans perte d'information et un moyen permettant de retrouver l'ensemble complet de motifs.

Une étude conceptuelle suivie d'une implémentation de l'algorithme MAFIA fera l'objet de ce rapport. l'expérience nous a montré que MAFIA inclut des multiples astuces et heuristiques intéressantes que nous n'avons pas pus les implémentés intégralement.

Mots clés :
Fouille de données, Motifs fréquents maximaux, MAFIA, Implémentation Java.

ملخص

استخراج الأنماط المتكررة من أهم المهام في مجال تنقيب البيانات. خلال العقدين الماضيين تم إدراج العديد من الخوارزميات لحل هذه المشكلة الشائعة . ومع ذلك، فإن الطبيعة الإندماجية والأسية لهذه المشكلة جعلت التعداد الكلي للأنماط صعبا للغاية أو حتى غير قابلة للتطبيق، لذلك تهدف حلول أخرى الى استخراج تمثيل مضغوط لمجموعة الأنماط.
الهدف من هذه المذكرة هو دراسة أسلوب إستخراج العناصر المتكررة القصوى، الذي يمثل تمثيلا مضغوطا دون فقدان المعلومات وهو وسيلة للعثور على المجموعة الكلية للأنماط.
موضوع هذا التقرير، ستكون الدراسة التصورية متبوعة بإنجاز خوارزمية مافيا . لقد أثبتت التجربة على ان مافيا تتضمن العديد من الأفكار والتجارب المثيرة للاهتمام التي لم ننفدها بالكامل .

كلمات مفتاحية

تنقيب البيانات، مجموعة العناصر المتكررة القصوى، مافيا، تنفيد جافا

Abstract

Extraction of frequent Itemset is a major task in Data mining. Ovrer the last two decades, a significant algorithms have been brought to solve this wide-spread problem. However, the combinatory and exponential nature of this problem makes the total enumerations of frequent Itemsets difficult or even infeasible. Other solutions, on the other hand, aim at extracting a compact representation of Itemset.

The aim of this research is to examine the extraction approach of the Maximal Frequent Itemsets that constitute a lossless compact representation, as well as a means that allows retrieving the full frequent itemsets.

The subject of this report will be a conceptual study followed by an implementation of the MAFIA algorithm. By experience, MAFIA algorithm included multiple hints and important heuristics that we are not able to fully implement.

Keywords :
Data Mining, Frequent Maximal Itemset, MAFIA, Implementation Java.

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Introduction générale

Avec le développement des domaines de la vie, nous sommes envahi de volumes colossaux de différents types de données. Cette croissance phénoménale est le résultat de l'informatisation de notre société et du développement rapide des outils de collecte, stockage, traitement et de transmission de données. Les informations jouent actuellement un rôle important dans la société, notamment dans les volets économique, industriel, et de recherche, etc. Le succès des bases des données et des technologies numériques en particulier les codes barre ont contribué à la génération d'ensembles de données de transactions d'achat gigantesques.

Les avancées technologiques introduites durant les dernières décennies ont pu maitrisé le volet stockage de données. Cependant, l'aspect exploitation et analyse des données accumulés demeure posé. La question est maintenant comment découvrir et extraire des connaissances afin de comprendre et valoriser les données recueillies. Il a été constaté que les moyens et méthodes disponibles jusque-là tel que la statistique sont limités pour la prise en charge de ces questions.

Pour faire face à la question de découverte de la connaissances à partir de données, il s'est développé depuis les année 90 un nouveau champ de recherche connu comme domaine de la fouille de données (communément appelé l'extraction des connaissances). Ce domaine concerne le développement des méthodes et des techniques pour comprendre et résoudre de nombreux problèmes. Il est le produit de plusieurs disciplines différentes, ce qui le rend plus efficace dans l'analyse et le traitement des données par rapport aux méthodes traditionnelles. La fouille de données est maintenant l'une des technologies les plus modernes.

La fouille de données inclut plusieurs techniques et tâches tels que la description, la classification, la prédiction, ainsi que les règles d'association. Cette dernières tâche est considérée comme une tâche de base qui a attiré l'attention des chercheurs. Elle vise la découvert de corrélations et dépendances entre des ensembles d'attributs ou d'items. Le processus d'extraction des règles d'association passe par deux phases :

1. Trouver les ensembles des motifs fréquents (le nombre d'occurrences dépasse un seuil fixé) dans la base de données.
2. Établir les règles d'association de ces ensemble fréquents.

Le processus d'établissement des règles d'association est un processus facile comparé à la recherche des ensembles des motifs fréquents. Ce dernier est un problème exponentiel en fonction du nombre d'items, ce qui rend toute énumération totale difficile et très coûteuse. Par conséquent, plusieurs algorithmes ont proposé d'extraire des représentations compactes des motifs fréquents.

Ce mémoire traite le sujet de l'extraction des motifs fréquents maximaux. Nous y développons un état de l'art de ces méthodes. Nous nous focalisons en particulier sur un algorithme efficace de cette classe de méthodes appelé MAFIA. Cet algorithme fera l'objet d'une présentation de ces principales idées ainsi qu'une implémentation en JAVA.

Ce mémoire est organisé comme suit. Le chapitre un s'intéresse à la définition des concepts de la fouille de données et discute ses principales tâches et techniques. Une étude des approches populaires d'extraction des motifs fréquents et des algorithmes les plus connus fera l'objet du chapitre deux. Le chapitre trois est réservé a l'étude de l'algorithme MAFIA et les détails de son implémentation. Le rapport est terminé par une conclusion.

Chapitre 1

Fouille de données

1.1 Introduction

Nous vivons dans un monde où d'énormes quantités de données sont collectées chaque jour, ce qui conduit à l'accumulation de données, et cela couvre différents domaines de la vie (économique, scientifique, industriel).

Cette croissance et accumulation massive de données sont le résultat de l'information de notre société et du développement rapide d'outils de collecte et de stockage de données.

Afin de bénéficier de ces données, il existe un besoin urgent d'outils puissants et d'utilisations multiples pour détecter automatiquement les informations importantes d'énormes quantités de données et les convertir à des connaissances pour l'organisation. Cela a conduit à la naissance de l'extraction de connaissance, ou ce que l'on appelle fouille de donnée.

Ce première chapitre présenté la définition de concept de la fouille de donnée, et les différentes étapes d'un processus d'extraction des connaissances à partir des données, Nous insistons sur les différentes approches et les technique des fouille de données les plus connus.

1.2 Définition de la fouille de donnée

Selon le Gartner Groupe « la fouille de donnée ou data mining est le procédé de découverte de corrélations significatives, de règles et de tendances en parcourant de grands volumes de données stockées dans des référentiels, en utilisant des technologies de reconnaissance de formes, mais également des techniques statistiques et mathématiques »[13]Il existe d'autres définitions.

— « la fouille de donnée est l'analyse d'un ensemble de donnée d'observations (souvent important) qui pour trouver des relations insoupçonnées et résumer les données d'une nouvelle manière, de façon qu'elles soient plus compréhensibles et utiles pour leurs détenteurs. » [12]

— « la fouille de donnée est un domaine pluridisciplinaire qui regrouper des techniques d'apprentissage automatique, de la reconnaissance de forme, des statistiques, des bases des données et de la visualisation pour apporter une réponse à l'extraction d'information provenant de base de donnée de grande taille. » [13]

1.3 Pourquoi la fouille de données ?

Le problème aujourd'hui n'est pas qu'il y ait pénurie de flux de donnée et d'information. En fait, nous sommes inondés des données dans Différents domaines, mais le problème est

comment extraire les connaissances de ces grandes quantités des données. Nous avons besoin de technologies qui aident dans l'amélioration du processus de recherche et développement, ce qu'on appelle les techniques d'intelligence artificielle, y compris la «la fouille de donnée ».[13]

Les facteurs qui ont conduit à L'exploration des données et la fouille de donnée sont :

— L'explosion des données.
— Les données sont collectées et stockées rapidement dans les entrepôts des données.
— L'énorme croissance de la puissance informatique et de la capacité de stockage

1.4 Processus d'extraction de connaissances (ECD)

La fouille de donnée n'est pas seulement un problème de l'extraction de modèles dans un ensemble de données, mais aussi une étape de processus ECD qui consiste à applique des algorithmes d'analyse et découvert de donnée.[6]

D'après [6] la définition d'ECD est « l'extraction de connaissances est un processus non trivial, qui consiste à identifier des modèles valides, nouveaux, potentiellement, utiles et surtout compréhensibles dans les donnée ».[6]

Le figure1.1 présente les quatre étape de processus ECD : nettoyage et intégration des données, pré-traitement des données, la fouille de donnée et l'évaluation et présentation des connaissances.

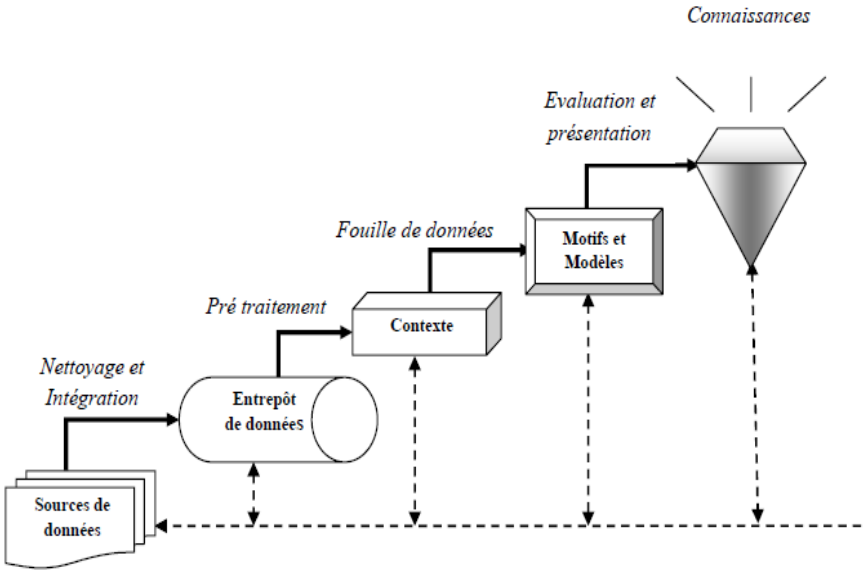

FIGURE 1.1 – processus d'extraction de connaissance à partir de donnée[6]

1.4.1 Nettoyage et intégration

L'opération de nettoyage consiste à traiter les données bruitées (doublons, information manquantes, erronées), soit en les supprimant, soit en les modifiant. L'intégration est le regroupement des données des différentes sources (base de données, sources externes, etc.) dans une seule structure.

Le but de ces deux opérations est de mettre en forme des entrepôts de donnée spécialisés contenant les données prétraitées pour faciliter la fouille.

1.4.2 Prétraitement de données

Cette étape permet de sélectionner les données objets de l'étude et pertinentes à l'analyse et les transformer de manière à les rendre exploitables par un outil de fouille de données. Le prétraitement de donnée est la plus longue étape elle occupe 80

1.4.3 Fouille de données

La fouille de données (data mining en anglais), est considérée comme le cœur du processus ECD car elle permet d'extraire la connaissance à partir de donnée, en applique les algorithmes et les technique de fouille de données selon un problème donné on recherche un modèle ou un motif intéressant. Cette étape est souvent difficile à mettre en œuvre et couteuse.

1.4.4 Évaluation et présentation des résultats

Dans La phase précédente on a extrait des connaissances utiles et intéressantes, et on a généré un modelé, Dans cette phase, dite d'évaluation ou de validation, l'objectif est de mesurer l'intérêt des modèles extraits, et de la présentation des résultats grâce à différentes techniques de visualisation pour aider l'utilisateur.

Deux approches sont communément utilisées dans la validation, la validation statistique et la validation par expertise.

•La validation statistique consiste à utiliser des méthodes de base de statistique descriptive. L'objectif est d'obtenir des informations qui permettront de juger le résultat obtenu, ou d'estimer la qualité ou les biais des données d'apprentissage.

•La validation par expertise est réalisée par un expert du domaine qui jugera de la pertinence des résultats produits.

1.5 Types de données

— Base données relationnelles : ce type de données est plus utilisé dans le regroupement d'un ensemble de données stockées dans une table.

— Entrepôt de données(Data warehouses) : contant une grande quantité de donnée stockée dans un schéma pour faciliter leurs exploitation futures, généralement modélisé par une structure de donnée multidimensionnelle appelle cube de donnée.[10]

— D'autre type de donnée : Il existe beaucoup d'autres types de données, des formes et des structures. Nous allons citer quelques uns : les données séquence, les données spatiales, les données multimédias(image, vidéos, audio) les données textuelles, les données orientées objet.

1.6 Tâches de la fouille de données

De nombreuses tâches peuvent être associées à la fouille de données

1.6.1 Classification

La classification une tache d'apprentissage supervise, est examiner les caractéristiques d'un objet et lui attribues une classe.

Plusieurs techniques sont applicables à la classification (réseaux de neurones, K plus proches voisins..).Des exemples de tache classification :

— attribuer ou non un prêt à un client.
— établir un diagnostic.

1.6.2 Estimation

L'estimation défini le lien entre un ensemble de prédicteurs et une variable cible catégorielle ou numérique. Donc pour estimer la valeur de variable cible on produit de nouvelle observation, et on se base sur les valeurs des prédicteurs.[13] Par exemple :

— Estimer les résultats du baccalauréat en fonction des résultats du brevet de collèges pour une population de lycéens.
— Estimer la pression sanguine à partir de l'âge, le sexe, le poids et le niveau de sodium dans le sang.

1.6.3 Prévision

La prévision est similaire à l'estimation et à la classification mise à part que pour la prévision, les résultats portent sur le futur. par exemple :

— Prévoir le prix d'action à trois mois dans le futur.
— Prévoir le gagnant du championnat de football, par rapport à une comparaison des résultats des équipes

1.6.4 Association

Cette tâche consiste à trouver quelles valeurs des variables sont corrélées ensemble.
Très répondue dans le monde du business ou des affaires elle est mieux connue en tant qu'analyse du panier de la ménagère, elle permet de recherche des associations pour mesure la relation entre deux attribués ou plusieurs.[13]
Généralement les règles d'association se forme « si <antécédent> alors <conséquent> » à travers des mesures de support et seuil de confiance de la règle [1] par exemple :

— la détermination des articles (le pain et le lait, la tomate, les carottes et les oignons) qui se retrouvent ensemble sur un même ticket de supermarché.
— Étudier quelle configuration contractuelle d'un abonné d'une compagnie de téléphone portable conduit plus facilement à un changement d'opérateur.

1.6.5 Segmentation

La segmentation ou (le clustring) consiste à forme des groupes homogènes appeler segmentation ou clustre, les algorithmes de clustring visent à segmenter la totalité des donnée en de sous-groupes en maximisant la similarité à l'intérieur de chaque groupes et en minimisant entre déférents groupes. Par exemple Placer un nouvel étudiant dans une filière particulière au regard de besoins spécifiques.

1.6.6 L'analyse d'exception et de déviation

L'analyse d'exception est de dégager et d'étudier des exceptions ou des surprises contenues dans les données, comme par exemple les objets ne pouvant être classés dans une classification .

Ces cas peuvent révéler des explications utiles dans certains domaines, ou indiquer des données bruitées ou erronées.

1.7 Techniques de la fouille de donnée

La fouille de donnée domaine multi-spécialité donc il adapter ses technique dans plusieurs domaine de recherche (statistiques, apprentissage automatique, système base de donnée). Pour tout jeu de donnée et un problème spécifique il existe plusieurs techniques. Parmi les techniques nous citons :

1.7.1 K-plus proche voisins

Est une méthode de classification la plus simple et la plus utilisée, elle est basée sur la distance dans lequel l'ensemble d'apprentissage est mémorisé, le principe de cette algorithme est :

- Déterminer le paramètre K plus proches voisins.
- Calculer la distance d(x,y) entre le nouvel objet avec tout l'objet de la base de données.

$$d(X,Y) = \sqrt{\left(\left[d_1(X_1,Y_2)^2 + ... + d_n(X_n,Y_n)^2\right]\right)} \tag{1.1}$$

- trier les distances de plus proche voisine croissante.
- Choisir la classe la plus proche.
- Affecter l'objet à la classe choisie.

1.7.2 Arbre de décision

arbre de décision est une représentation graphique d'une procédure de classification organisé de manière arborescente.
Cette technique est constituée d'un ensemble de nœuds des décisions connectés par des branches, en commençant au nœud racine jusque à se terminer par des nœuds feuilles.
la figure1.2 représente un exemple simple d'arbre de décision.[1]
Dans cet exemple, Il représente le concept achète l'ordinateur, c'est-à-dire, il prédit si un client est susceptible d'acheter un ordinateur ou non, Ils sont classés par L'âge de chaque client (jeunesse, âge moyen, Sénior)

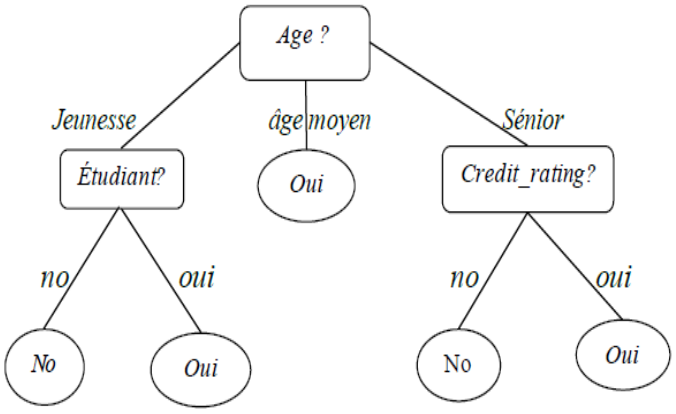

FIGURE 1.2 – Arbre de décision [10]

1.7.3 Réseaux de neurones

Un réseau de neurone est composé de plusieurs neurones inter-connectés, cette méthode simule le fonctionnement de réseau neuronal biologique de l'être humain.
C'est un outil très utilisé pour la classification, l'estimation et la segmentation.[13]
Les données d'entrée (x_i) dans la figure1.3 sont recueillies à partir des neurones et combinées à travers d'une fonction Σ, qui entrée dans fonction d'activation généralement non linéaire pour produire la résultat en sortie Y.

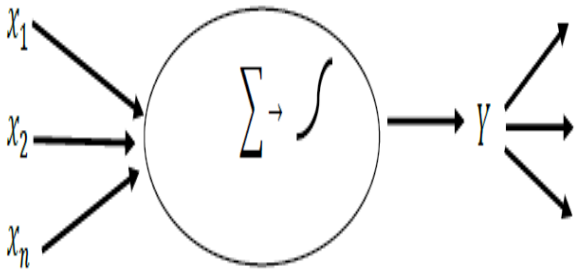

FIGURE 1.3 – Un modèle de neurone artificiel[10]

Étudions le réseau de neurones simple présenté dans la figure1.4 un réseau de neurones consiste en réseau de neurones artificiels il composé de plusieurs couches, les trois couches le plus connu est : une couche d'entré, une couche cachée, une couche de sortie.
Le réseau de neurones est complètement connecté, par ce que chaque nœud d'une couche de donnée connecté à chaque nœud suivante, et la connexion entre les nœuds c'est un poids (w)
Les nombre de nœuds en entrées dépondent à de variables de l'ensemble des données et du problème modélisé et le nombre de couche cachée déponde les besoin de l'utilisateur.
Exemple :

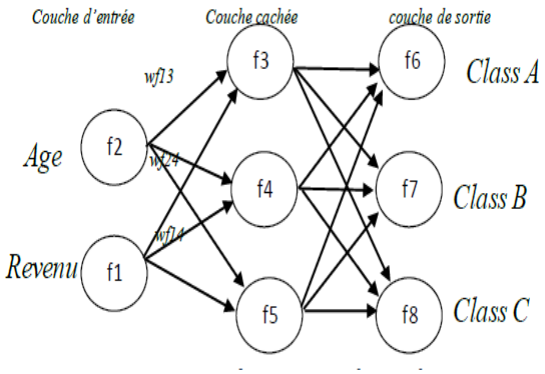

FIGURE 1.4 – Réseau de neurones multi-couches pour la classification [10]

1.7.4 k moyennes

L'algorithme k-moyennes ou k-means est une technique de segmentation définit le centroïde d'un cluster comme la valeur moyenne des poids dans le cluster, il procédé comme suit :

1. Cluster les données en K groupe, K est prédéfini.
2. Sélectionné aléatoirement le K centre de classe initiaux.
3. Trouver la classe le plus proche de chacun centres, en utilisant le calcul de distance.
4. Calculer les centroïdes ou la moyenne de tous les objets dans chaque groupe ou cluster.
5. Répéter l'étape 3 – 4 – 5 jusque stabilité les k centre. L'algorithme termine quand les controïde ne changent plus.

1.7.5 Autre techniques

Il y a plusieurs de technique que applique dans la fouille de donnée Nous mentionnons : régression linéaire (simple et multiple), le SVM (Support vecteur machine) que sépare les données, classification de bayes .

1.8 Domaines d'application

La fouille de données trouve de nombreuses applications qui ont décroché un succès impressionnant dans de nombreux domaines tels que : l'économie, les sciences, la gestion des affaires, médecine et sports.[10] On site quelques exemples :

- Marketing : C'est l'un des applications commerciales le plus réussie dans le domaine de l'extraction de connaissance.
  L'exploitation des bases de données des achats des clients précédents peut dérivée des habitudes d'achat, y compris la création de fichiers pour les clients qui peuvent être utilisés pour commercialiser plus efficacement.
- Médecine : Quelques exemples de l'usage médicaux des techniques de fouille de donnée pour l'analyse de bases de données médicales.
  - Prédiction de présence de maladies et/ou de complications.
  - Le choix d'un traitement pour une malade.
- Moteurs de recherche Web : Les moteurs de recherche Web sont de très grandes applications pour l'exploration de données.
  Différentes techniques d'extraction de données sont utilisées dans tous les aspects des moteurs de recherche, allant de l'exploration (par exemple, décider des pages à explorer et des fréquences d'exploration), l'indexation (par exemple, sélection des pages à indexer et détermination de la mesure dans laquelle l'index doit être construit), et la recherche dans le web.

1.9 Conclusion

Dans ce chapitre, nous avons évoqué les concepts de base de la fouille de donnée qui ont été discutés.Et la fouille de ces données peut aider à résoudre des problèmes d'analyse de données rencontrés par de nombreuses institutions et entreprises, ce qui le rend très important dans le présent et le futur.

Dans le chapitre suivant, L'étude sera centrée à propos le concept d'analyse de association et sa relation avec l'extraction des motifs fréquent et la énumération total à travers des algorithme et des approche, nous allons toucher les représentation découvertes.

Chapitre 2

Extraction des motifs fréquents

2.1 Introduction

La recherche de motifs fréquents et des règles d'association entre des objets a fait l'objet de nombreux travaux en fouille de donnée qui sont utilisés aujourd'hui dans de nombreux domaines de la vie.

Le concept de règle d'association est apparu pour la première fois à travers l'analyse du panier de manger (Market Basket Analysis),afin de mieux comprendre les habitudes d'achat des clients,pour une classification optimale des produits dans les centres commerciaux.

Dans ce chapitre, nous allons étudier la tâche d'analyse d' association qui est utilisée pour la découverte des associations ou des relations cachées dans les grandes bases de données. Les relations découvertes peuvent être représentées sous forme de règles d'association entre ensemble des motifs fréquents, Il y a deux étape clés qui doivent être considérés pour l'extraction des règles d'association.

premièrement, trouver tous les motifs fréquents : la recherche des ensembles dont la récurrence est supérieure ou égale à un certain seuil. Deuxièmement, établissement de règles d'association à partir des motifs fréquent trouvés dans l'étape précédents.

L'opération de l'extraction tous les motifs fréquent est une opération plus coûteux en termes de temps et d'espace de stockage,par conséquent, les chercheurs ont suggéré des approches et des algorithmes qui amélioreraient le coût de la recherche des ensemble de motifs fréquents dans un grand ensemble de données.

Dans ce chapitre, nous touchons les différentes types d'approches et les algorithmes utilisés pour trouver l'ensemble des motifs fréquents comme l'approche naïve, l'approche par niveau, l'approche verticale et l'approche projective.
Au début millénaire est apparu nouvelle direction que rechercher les possibilité d'extraction ces règles à partir de moins d'espace pour les ensemble des motifs fréquents, que sont des représentations compactes, qui considères un noyau de l'espace total des ensemble des motifs fréquents comme les représentations de motifs fréquents fermés et les représentation des motifs fréquent maximaux.

2.2 Motifs fréquents

Nous allons étudier une méthode connue sous le nom d'analyse des associations et qui est utilisée pour découvertes des associations ou des relations cachées dans les grandes bases de données. Par exemple, les grands magasins collectent énormément de données sur les achats des consommateurs via les tickets de caisses. Le tableau 2.1 donne une illustration de ce type de données. Chaque linge correspond à une transaction et reporte le numéro de ticket ainsi qu'une liste de produits achetés. Les commerçants sont intéressés par l'analyse de ce type de données pour mieux connaitre les comportements d'achat de leurs clients. Ces informations servent à bien mener les compagnes marketing mieux gérer les inventaires ou améliorer les relations clients [17].

TID	Motifs
1	$\{Pain, Lait\}$
2	$\{Pain, Couches, oeufs\}$
3	$\{Lait, Couches, Coca\}$
4	$\{Pain, Lait, Couches\}$
5	$\{Pain, Lait, Couches, Coca\}$

TABLE 2.1 – Panier de ménagère

Le tableau précédent représente les articles achetés qui sont les items, alors que les achats jouent le rôle des transactions. L'ensemble forme ce qui communément appelé une base de transactions.

— **Item** : un élément de l'ensemble non vide $I = \{i_1, i_2, \ldots, i_m\}$ de m items. ces items peuvent représenter des produits, des objets, des patients, des évènements, etc.

— **Motifs** : un motif ou un itemset X est un ensemble d'items. il est dit $(k - Motif)$ si son cardinal est K $(|X| = k)$

— **Transaction** : Un ensemble non vide d'items identifiée par un numéro unique i notée t_i.

— **Base de transactions** un ensemble D de n transactions $D = \{t_1, \ldots t_n\}$

— **support d'un motif** : est le nombre de occurrence de motif dans la base de transactions, c-à-d le nombre de transaction que contient le motif. formellement :

$$sup(x; D) =\mid \{t_k \in D \mid x \subseteq t_k\} \mid$$

le support peut aussi être exprimé en relatif comme une fraction entre 0 et 1 en le divisant par la taille de la base D

$$Rsup(X, D) = \frac{sup(X, D)}{|D|}$$

— **Motifs fréquents** : un motif est fréquent si son support dépasse un seuil minimal μ spécifie par l'utilisateur . formellement : X est fréquent ssi $sup(X, D) \geq \mu$

— **Fouille des motifs fréquents** : ce problème consiste à trouver l'ensemble $\mathcal{F}$ de tous les motifs fréquents dans une base D par rapport à un seuil μ

$$\mathcal{F}_D(s) = \{x \subseteq I \mid \sup(x) \geq \mu\}$$

2.3 Règles d'association

Une règle d'association est une implication de la forme $R : X \to Y$ où X et Y sont des motifs disjoints elle définit le lien entre deux motifs X et Y dans base de transactions.

Généralement le nombre de règle d'association très élevé, si pour cet raison qu'on impose d'autre critères (support et confiance) afin de ne retenir que celles jugées intéressantes.

— **Support d'une règle** : est représente le support de ses constituants X et Y [19].

$$sup(X \to Y) = sup(X \cup Y)$$

— **Confiance d'une règle** :exprime la force et la certitude de la règle comme une probabilité conditionnelle des transaction que contient conséquence sachant quelles contient aussi l'antécédent.

$$Conf(X \to Y) = \frac{sup(X \cup Y)}{sup(X)}$$

de façon similaire, une règle et retenir si ca confiance dépasse un seuil minimale de confiance λ. tel que λ est une paramètre fourni par l'utilisateur.

$$Conf(X \to Y) \geq \lambda$$

Transaction	Motifs
1	$ABDE$
2	BCE
3	$ABDE$
4	$ABCE$
5	$ABCDE$
6	BCD

TABLE 2.2 – base de transaction

Exemple Considérons la règle d'association $BC \to E$. Utilisation support de motif Valeurs indiquées dans le tableau 2.2, le support et la confiance de la règle sont les suivants :
$sup(BC \to E) = sup(BCE) = 3$
$conf(BC \to E) = sup(BCE)/sup(BC) = 3/4 = 0,75$

— **Génération de règle d'association**
La création de règles d'association est effectuée à partir de l'ensemble des motifs fréquents F, et calcul la confiance des règles peuvent être dérivées à partir d'un ensemble des motifs fréquents $Z \in F$, nous regardons tout sous-ensembles $X \subset Z$
. Pour calculer les règles du formulaire $X \to Y$, où $Y = Z/X$ où $Z/X = Z - X$. La règle doit être fréquente car $s = sup(XY) = sup(Z) \geq \mu$
Si le coefficient de confiance est supérieur ou égal au seuil minimum du coefficient de confiance, la règle d'association une règle forte. Si $conf(X \to Y) < c$ (voir algorithme1) [19].

Algorithme 1 GÉNÉRATION DES RÈGLES D'ASSOCIATION

Entrée : L'ensemble des motifs fréquents $\mathcal{F}$
Sortie : Les règles d'association fortes

1. Soit $\mathcal{F}$ l'ensemble de motifs fréquents trouvés dans la phase une,
2. Itérer en considérant à chaque étape un motif $I \in \mathcal{F}$ avec $|I| \geq 2$
   (a) Choisir X parmi les sous motifs propres de I. Soit Y son complément à I, *i.e.*, $Y = I \setminus X$. (Pour bénéficier de l'optimisation en (b), il est préférable de commencer par les sous motifs ayant les cardinalités maximales)
   (b) Former la règle $X \Rightarrow Y$ et calculer sa confiance, puis l'éditer si la condition de confiance est remplie. Autrement, la négliger et ignorer également tous les sous motifs X' de X dans la suite des itérations (les règles qui en résultent ne peuvent être fortes car sprt(X′) ≥ sprt(X))
   (c) S'il existe des sous motifs propres non traités : aller à (a)
3. Si $\mathcal{F}$ non épuisé : aller à (2)

Exemple : Un exemple d'une base de donnée de tableau 2.2 $I = \{A, B, C, D, E\}$ considérer les motifs fréquentes $ABDE(3)$ à partir de tableau 2.1, dont le support est indiqué dans le parenthèses. supposer que $minconf = 0.9$. pour générer une forte règles d'association nous initialiser l'ensemble des antécédents à un
$I =$ { ABD (3), ABE (4), ADE (3), BDE (3), AB (3), AD (4), AE (4), BD (4), BE (5), DE (3), D (4), B (6), D (4), E (5) }
le premier sous-ensemble est $X = ABD$, et la confiance des $ABD \rightarrow E$ est $3/3 = 1.0$, nous avons donc de sortie $X = ABE$, mais le correspondant règle $ABE \rightarrow D$ ne sont pas forte depuis $conf(ABE \rightarrow D) = 3/4 = 0.75$.
nous pouvons donc retirer à partir d'une tous les sous-ensembles de ABE, la mise à jour ensemble de antécédents est :
$A =$ { ADE (3), BDE (3), AD (4), BD (4), DE (3), D (4) }
Ensuite, nous sélectionnez $X = ADE$, ce qui donne une forte règle, et ainsi de faire $X = BDE$ et $X = AD$. cependant, quand nous processus $X = BD$, nous trouvons que $conf($ $BD \rightarrow AE$ $) = 3/4 = 0.75$, et donc nous pouvons prune tous les sous-ensembles de BD à partir d'un, pour donner un
$A =$ { DE (3) }
la dernière règle d'être est $DE \rightarrow AB$ qui aussi forte [19]. l'ensemble final de la forte règles comme suit :
$ABD \rightarrow E, conf = 1.0$
$ADE \rightarrow B, conf = 1.0$
$BDE \rightarrow A, conf = 1.0$
$AD \rightarrow BE, conf = 1.0$
$DE \rightarrow AB, conf = 1.0$

2.4 Énumération totale

Pour extraire un ensemble des motifs fréquents dans une base de transaction il existe plusieurs types d'approches. nous présentons les principale approches dans les sections suivant :

2.4.1 Approche naïve

Un algorithme naïf ou brute force énumère tous les éléments possibles qui s'appelle les motifs candidat et calcule le support pour chaque élément et détermine s'il est fréquent ou non, son pseudo-code est montré au dessous [19].

Algorithme 2 BRUTEFORCE(D, I, μ)

Entrée : Une base de donnée $\mathcal{D}$, tout les ensemble des item $\mathcal{I}$,minsup μ
Sortie : L'ensemble des motifs fréquents $\mathcal{F}$

```
F ⟵ ∅
pour X ⊆ I faire
    sup(X) ⟵ COMPUTE-SUPPORT(X,D)
    si sup(X) ≥ μ alors
        F ⟵ F ∪ (X, sup(X))
    fin si
fin pour
retourner 𝔽

COMPUTE-SUPPORT(X,D)
sup(X) ⟵ 0
pour t_i ∈ D faire
    si X ⊆ t(i) alors
        sup(X) ⟵ sup(X) + 1
    fin si
fin pour
```

— la génération de candidats : l'espace de recherche des motifs candidats de l'ensemble I est exponentiel en fonction du nombre d'items $|I|$ donc la complexité est $O(2^{|I|})$
— Le calcul de support : la complexité de calculer de support dans le pire des cas $O(|I||D|)$ par ce que la détermination des éléments fréquents nécessite l'analyse complète de la base de donnée(D) [19].

Donc la complexité de l'algorithme Brute-force $O(|I||D|2^{|I|})$.
Ainsi, l'approche naïve est non pratique même dans les petits ensembles de données.

2.4.2 Approches par niveaux

Le premier algorithme dans cette approche est l'algorithme Apriori proposé par Agrawal en 1994 [3]. Il utilise une exploration horizontale de l'espace de recherche ; ainsi, il calcule les sanglotants fréquent puis a partir de ces derniers trouve les paires fréquents, pour par la suite extraire les triplets fréquents, etc. Apriori optimise son travaille d'une part, en adaptant une alternance entre génération de candidats et formation de motifs fréquents. D'autre part, il introduit une heuristique dite la propriété Apriori stipulant que chaque ce motifs d'un motifs fréquent est aussi fréquent est parallèlement tout sur-motifs d'un motifs infréquent ne peut être fréquent. Le déroulement de l'algorithme Apriori (Algorithme 3).

Algorithme 3 APRIORI(D, μ)

Entrée : Une base de transactions D, et un seuil de support μ
Sortie : L'ensemble $\mathcal{F}$ des motifs fréquents

$k \leftarrow 1$
$\mathbb{F}_1 \leftarrow$ {Les sigletons fréquents}
tant que $\mathbb{F}_k \neq \emptyset$ **faire**
  Générer $\mathbb{C}_{k+1}$ par jointure des motifs de $\mathbb{F}_k$
  Éliminer de $\mathbb{C}_{k+1}$ les éléments qui violent la propriété Apriori
  Déterminer $\mathbb{F}_{k+1}$ par calcul des supports des candidats retenus
  $k \leftarrow k + 1$
fin tant queretourner $\cup_i \mathbb{F}_i$

Exemple : Considérons l'exemple de la base de données de la table 2.2; $\mu = 3$ $\mathbb{C}_1 = \{A, B, C, D, E\}$
$\Rightarrow \mathbb{F}_1 = \{A(4), B(6), C(4), D(4), E(5)\}$
$\mathbb{C}_2 = \mathbb{F}_1 \times \mathbb{F}_1 = \{AB(4), AC(2), AD(3), AE(4), BC(4), BD(4), BE(5), CD(2), CE(3), DE(3)\}$
$\Rightarrow \mathbb{F}_2 = \{AB(4), \cancel{AC(2)}, AD(3), AE(4), BC(4), BD(4), BE(5), \cancel{CD(2)}, CE(3), DE(3)\}$
$\mathbb{C}_3 = \mathbb{F}_2 \times \mathbb{F}_2 = \{ABD(3), ABE(4), ADE(3), BCE(3), BCD(2), BDE(3)\}$
$\Rightarrow \mathbb{F}_3 = \{ABD(3), ABE(4), ADE(3), BCE(3), \cancel{BCD(2)}, BDE(3)\}$
$\mathbb{C}_4 = \mathbb{F}_3 \times \mathbb{F}_3 = \{ABDE(3), BCDE(1)\}$
$\Rightarrow \mathbb{F}_4 = \{ABDE(3), \cancel{BCDE(1)}\}$
$\mathcal{F} = \mathbb{F}_1 \cup \mathbb{F}_2 \cup \mathbb{F}_3 \cup \mathbb{F}_4$ exemple prédicant explique le mécanisme de travail cet L'algorithme et représenter comme arbre dans le figure 2.1

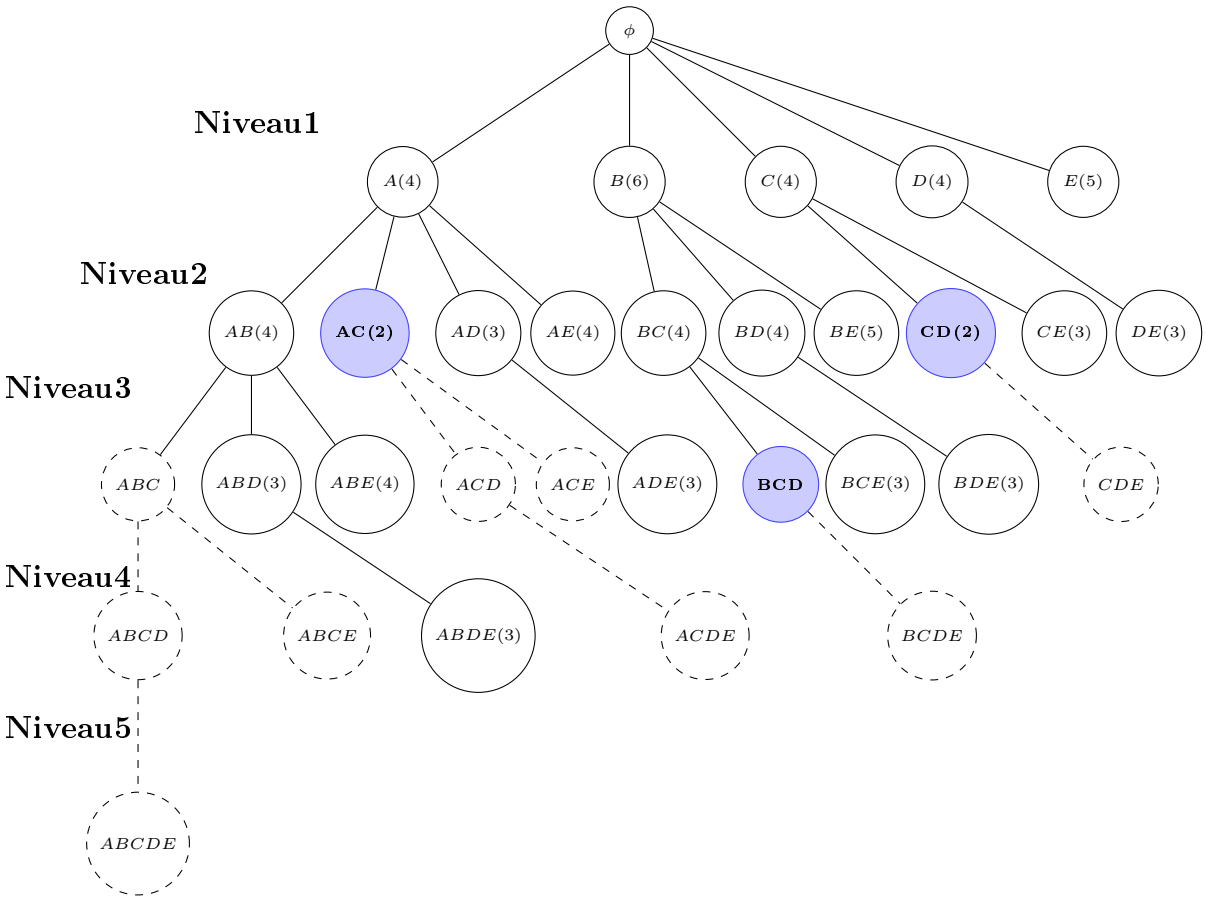

FIGURE 2.1 – Apriori : arbre de recherche de préfixe et effet d'élagage. Les nœuds ombrés indiquent des ensembles d'items peu fréquents,tandis que les nœuds et les lignes pointillés indiquent tous les nœuds et toutes les branches élagués. Les lignes pleines indiquent des motifs fréquents [19].

La complexité de l'algorithme d'Apriori dans le pire des cas $O(\mid I \mid\mid D \mid 2^{|I|})$, car tous les motifs pouvons être fréquents. En pratique, en raison de l'élagage de l'espace de recherche, le coût est beaucoup plus bas. Cependant, Apriori nécessite $O(|I|)$ parcoure de la base de données. Malgré qu'en pratique il ne nécessite que l scans où l est taille de motifs le plus long motif dans la base [19].
En dépit, de la popularité et la simplicité de l'algorithme Apriori ce darnais soufre de deux inconvénients majeurs :

- la génération d'un nombre considérable de candidats ce qui dégrade significativement ses performances
- Le surcoût causé par les multiples et couteuses opérations d'entrée/sortie nécessaires pour le calcul des supports.

2.4.3 Approches verticales

Cette approche utilise une représentation verticale, où nous pouvons regarder la base de transactions comme une structure inversée : a chaque item est associée sa listes de transaction ou il apparait appelé tidlist de l'item considéré [19].
L'idée de base dans cet approche est le calcul des supports via des intersections de tidlist, et le parcours de l'espace du problème en profondeur (depth-first traversal (DFS)).

Le premier algorithme qui utilise cette approche est l'algorithme Eclat proposé par Zaki et al [20]. Eclat (Equivalence CLAss Transformation) est un algorithme récursif, basé sur deux principes : l'organisation verticale de la base de donnée, et la maniéré de calcul des supports par des intersections d'un coté. D'un autre coté, une stratégie de décomposition de l'espace de recherche en parties à travers la définition de classes d'équivalence, afin de réduire l'espace mémoire requis lors de l'exploration. Le pseudo code de l'algorithme Eclat ainsi le déroulement sur l'exemple de référence sont fournis ci-après.

Algorithme 4 ECLAT$(L, \mu, \mathcal{F})$

Entrée : Une base de données transactionnelle D, et un seuil de support μ
Sortie : L'ensemble $\mathcal{F}$ des motifs fréquents

```
F ← ∅
L ← {les singletons fréquents}
pour tout P_i ∈ L faire
    F ← F ∪ P_i
    FP_i ← ∅
    pour tout P_j ∈ L | j > i faire
        si |tidlist(P_i) ∩ tidlist(P_j)| ≥ μ alors
            FP_i ← FP_i ∪ {P_i ∪ P_j}
        fin si
    fin pour
    si FP_i ≠ ∅ alors
        ECLAT(FP_i, μ, F)
    fin si
fin pour
```

Pour utiliser l'algorithme Eclat, la base de transaction de tableau 2.2 a été converti a la base verticale tableau 2.3, avec $\mu = 3$.

x	A	B	C	D	E
	1	1	2	1	1
	3	2	4	3	2
T(x)	4	3	5	5	3
	5	4	6	6	4
		5			5
		6			

TABLE 2.3 – Représentation verticale

La classe d'équivalence de préfixe initial est :

$$P = \{< A, 1345 >, < B, 123456 >, < C, 2456 >, < D, 1356 >, < E, 12345 >\}$$

l'algorithme Eclat calcule l'intersection de tidlist $t(A)$ de l'item A avec les autre tidlist $t(B), t(C), t(D), t(E)$ pour construire les tidlist des ensembles AB, AC, AD, AE ensuite, il est calculé le support pour tous les ensemble et déterminer les fréquent pour ajoute dans la classe d'équivalence P_A. Nous avons calcule les tidlist pour les items :
$t(AB) = t(A) \cap t(B) = 1345 \cap 123456 = 1345$

$t(AC) = t(A) \cap t(C) = 1345 \cap 2456 = 45$
$t(AD) = t(A) \cap t(E) = 1345 \cap 1356 = 135$
$t(AE) = t(A) \cap t(D) = 1345 \cap 12345 = 1345$
Puis déterminer le support :
$sup(AB) = |1345| = 4$
$sup(AC) = |45| = 2$
$sup(AD) = |135| = 3$
$sup(AE) = |1345| = 4$
Donc en comparer les support de chaque tidsets par a pour le $minsup = 3$, les items qui dépasse le seuil est AB, AD, AE, la classe équivalence de P_A est :
$\{P_A =< AB, 1345 >, < AD, 135 >, < AE, 1345 >\}$
Et d'une manière récursive en terminer le calcule de tous les item .

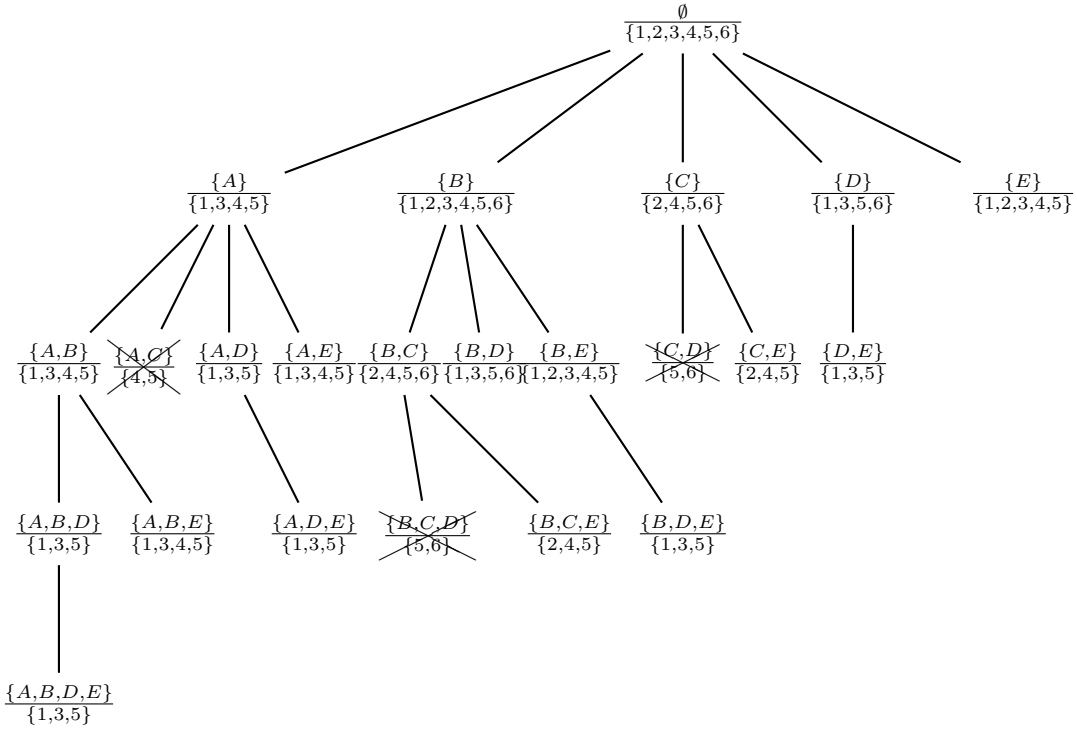

FIGURE 2.2 – Eclat déroulé sur l'exemple de référence s. Les tidlists résultants des différentes intersections sont en parties basses des nœuds

La complexité d'Eclat est $O(\mid D \mid 2^{|I|})$ dans le pire des cas, puisque on peut avoir $2^{|I|}$ motifs fréquents. Est l'intersection de deux tidlists est bornée par la taille de la base [19].

Il a été trouvé qu'Eclat se comporte très bien devant des contextes de fouille pour des bases de données sparses qui impliquent souvent des intersections légères. Cependant, dans le cas des bases denses les résultats intermédiaires générés par Eclat deviennent énormes où il soufre alors de manque de mémoire.

2.4.4 Approches projectives

Afin de compresser la base de transactions et éviter ses parcours répétés . Han, Pie, Yin [11] ont proposer la approche différente qui ne nécessite pas une phase de génération de candidats. les auteurs introduit l'algorithme FP-Growch (Frequent pattern growch) qui compacte la base dans une structure réduite appelée FP-Tree (Ferquent Pattern Tree) qui est une sort d'arbre

de préfixes augmenté par les information de support. Ainsi les deux élément essentiels qui composent la structure de FP-Tree sont :

1. une structure sous forme d'un arbre avec une racine étiquetée "NULL" et un ensemble de nœuds, chaque nœud contient un item et stock le support de motif formé des items trouvés sur le chemin depuis la racine de à ce nœuds.
2. une table entête avec des pointeurs intra-nœuds qui jeu le rôle d'un index pour faciliter l'accès aux déférents nœud de l'arbre (les nœud ayant le même item sont relier entre eux)[18].

FP-Growch adopte une exploration récursive en profondeur de l'espace de recherche, il nécessité deux passes de la base de donnée. Dans la première passe, il détermine les items fréquents en fonction du support minimum, ces items seront triés par l'ordre décroissant de leurs supports(les items infréquents sont supprimés). Dans la deuxième passe l'arbre FP-Tree est construire : chaque transaction est alors triée selon l'ordre des items, le nœud racine de l'arbre est d'abord crée (Null) durant ce même parcours, chaque transaction réécrite selon le nouveau ensemble et l'ordre d'item qui retenus puis insérée dans l'arbre, mais des transaction ayant un même préfixe partagent les même nœuds.

La réécriture des transactions de la base selon les items fréquents et la représentation par l'ordre décroissant de support a permis de réduire considérablement la taille de la base, car cette structure inclut uniquement des items fréquents, où les plus fréquents sont regroupés dans les niveaux supérieurs de l'arbre .

La base de données de la table 2.1. Nous ajoutons chaque transaction dans l'arbre FP, et garder une trace du compte à chaque nœud. Pour notre exemple de base de données l'ordre des éléments triés est $B(6), E(5), A(4), C(4), D(4)$. Prochain, chaque transaction est réorganisée dans le même ordre ; par exemple, $< 1, ABDE >$ *devient* $< 1, BEAD >$. La Figure 2.3 illustre la construction pas à pas de l'arbre FP comme trié transaction est ajouté à celui-ci. L'arborescence FP finale de la base de données est montré dans la Figure 2.3. Pour déterminée l'arbre FP en utilise la base de transaction de tableau 2.4 a été ordonné décroissant.

Tid	Motifs	L'ordre des motifs fréquents
1	ABDE	BEAD
2	BCE	BEC
3	ABDE	BEAD
4	ABCE	BEAC
5	ABCDE	BEACD
6	BCD	BCD

TABLE 2.4 – L'ordre des motifs fréquents

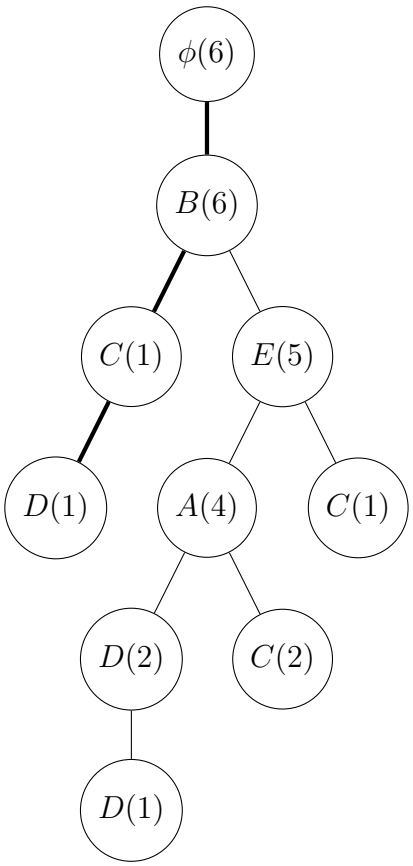

FIGURE 2.3 – FPGrowth[19]

Après la construction du FP-Tree de la base, l'exploration est faite directement sur lui car il contient toute l'information nécessaire. Le code suivant donne l'algorithme FP-Growth qui reçoit l'arbre FP-Tree, un seuil de support μ, et le motif courant initialement vide.

Algorithme 5 FPGrowth(FPT, μ, P)

Entrée : FPT l'arbre compact d'une base de données transactionnelle D, min support μ et le suffixe courant P

Sortie : L'ensemble $\mathcal{F}$ des motifs fréquents

```
si FPT est un chemin simple alors
    pour tout combinaison C de nœuds de FPT faire
        Écrire (C ∪ P)
    fin pour
sinon
    pour tout item a dans FPT selon l'ordre croissant du support faire
        Q = {a} ∪ P
        F ← F ∪ Q
        Temp ← ∅
        pour tout chemin d depuis la racine à a faire
            d1 ← d tronqué de a
            insérer d1 dans l'arbre Temp
        fin pour
        si Temp ≠ ∅ alors
            FPGrowth(Temp, μ, Q)
        fin si
    fin pour
fin si
```

Nous illustrons la méthode FPGrowth sur le FP-tree construit en Exemple présidant, comme le montre la Figure 2.3. Soit $\mu = 3$ Le préfixe initial est $P = \phi$ et l'ensemble des éléments fréquents i dans FP-tree est $B(6), E(5), A(4), C(4), D(4)$. FPGrowth crée un arbre FP projeté pour chaque items, FP-tree montré dans la Figure 2.4.

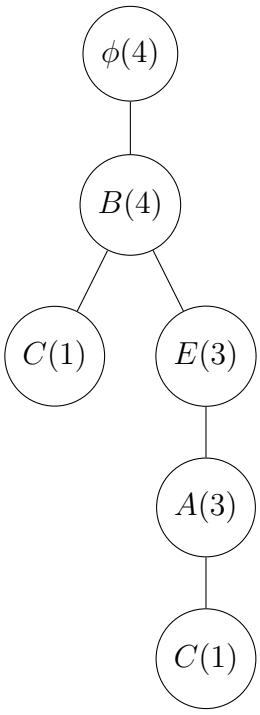

Figure 2.4 – Arborescence de motifs fréquents projetée pour D (BEAD,cnt = 2) [19]

La complexité asymptotique de l'algorithme demeure, quant à elle, toujours de l'ordre de $O(2^{|I|}|D|)$au pire des cas.

2.5 Énumération abrégée

Le plus grand défi est d'extraire des motifs fréquents du grand ensemble de données où l'espace de recherche pour ces objets est très grand et croît exponentiellement avec un nombre croissant d'items, en particulier lorsque le support(μ) est faible, Pour surmonter ce problème,il est courant d'utiliser des représentations compactes d'ensembles de motifs fréquents pour réduire les besoins de calcul et de stockage et faciliter le processus d'analyse des connaissances obtenues [19].
Deux types de représentation compacte seront décrit dans cette partie :
La représentation de motifs fréquents fermés (MFF) et la représentation de motifs fréquents maximaux (MFM). Il existe une relation d'inclusion entre ces différentes représentation. celle-ci est dans la figure 2.5.

$$(MFM \subseteq MFF \subseteq F)$$

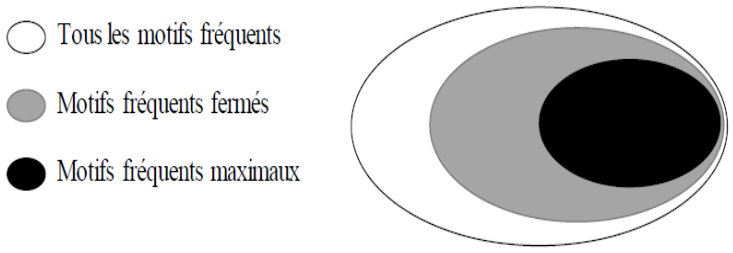

FIGURE 2.5 – Représentation Compactes [16]

les représentations compactes permettent de récupérer l'ensemble de tous les motifs fréquents originaux. dans la suite, nous donne la définition des motifs fréquents fermes (MFF), pour se focaliser en suite sur les motifs fréquent maximaux (MFM).

2.5.1 Motifs fréquents fermés

Un motif x est fréquent fermé dans une base de donnée D, s'il est fréquent et il n'existe pas un sur-motif y ayant le même support que x.
Formellement : un motif x est fréquent fermé s'est x est fréquent et

$$\nexists y \mid y \supset x \wedge \sup(y) = \sup(x)$$

En d'autres termes, x est fermé si tous les ensemble supérieur est de x ont strictement moins de support, c'est-à-dire $sup(X) > sup(Y)$,pour tout $Y \supset X$.
Exemple : Considère la base de donnée de tableau 2.2 on utilise le seuil ou le min-sup=3 pour extraire des motifs fréquent fermé. Sont montré le résultat dans le figure 2.6.
les éléments sont 5 donc tous les éléments possible si $2^5 - 1 = 31$ et les ensembles des motifs fréquent est 19, par exemple les ensembles d'éléments $\{AD,\ DE,\ ABD,\ ADE,\ BDE,\ ABDE\}$

sont trouver dans les même trois transaction « 135 », et ainsi constituer une classe d'équivalence,le plus grand ensemble des motifs parmi ceux-ci : ABDE est l'ensemble des motifs fermé. Notre exemple $\{A,\ AB,\ AE,\ ABE\}$ sont trouver dans la même transaction « 1345 » et l'ensemble des motifs fermé de cette groupe est ABE.
Tableau 2.5 de tous les ensembles des motifs fréquents fermés :

Tidset	C= l'ensemble fermé
1345	ABE
123456	B
1356	BD
12345	BE
2456	BC
135	ABDE
245	BCE

TABLE 2.5 – les ensemble fermé

2.5.2 Motifs fréquents maximaux

Un motif x est fréquent maximal dans une base de donnée D, s'est-il x fréquent par rapport un seuil μ. fréquent :c'est-à-dire la valeur de support est supérieure ou égale le seuil μ, et il n'existe pas un sur-motif y est aussi fréquent, d'autre façon tous les sur-motifs il est infréquent. Formellement : un motif x est fréquent maximal c'est x fréquent et

$$\nexists Y \mid Y \supset X \wedge sup(y) \geq \mu$$

Exemple :

Considérer la base de données donné à le tableau 2.2, en utilise un algorithme pour extraire les ensemble fréquent maximum et donné le $minsup = 3$ après le traitement il obtient les éléments fréquenté sur le tableau 2.6.

sup	les sous-ensemble des Motifs
6	B
5	E,BE
4	ABE, AB, BC, BD, AE, D, A, C
3	ABDE, ABD, ADE, BCE, BDE, CE, DE, AD

TABLE 2.6 – les motifs fréquent (min-sup=3)

Après le résultat de cette base de données, il y a deux ensembles fréquents maximal : $ABDE$ et BCE, par ce que tous les ensemble des motifs fréquent doit être un sous-ensemble de l'un des l'ensemble fréquent maximum.
Par exemple on peut déterminer que ABE est fréquent, puisque $ABE \subset ABDE$ et on peut

déterminer le support que $sup(ABE) \geq sup(ABDE) = 3$ puisque l'apparence de l'élément ABE dans l'ensemble $ABDE$ égale 3.
le figure 2.6 afficher le résulta de MFM et MFF.

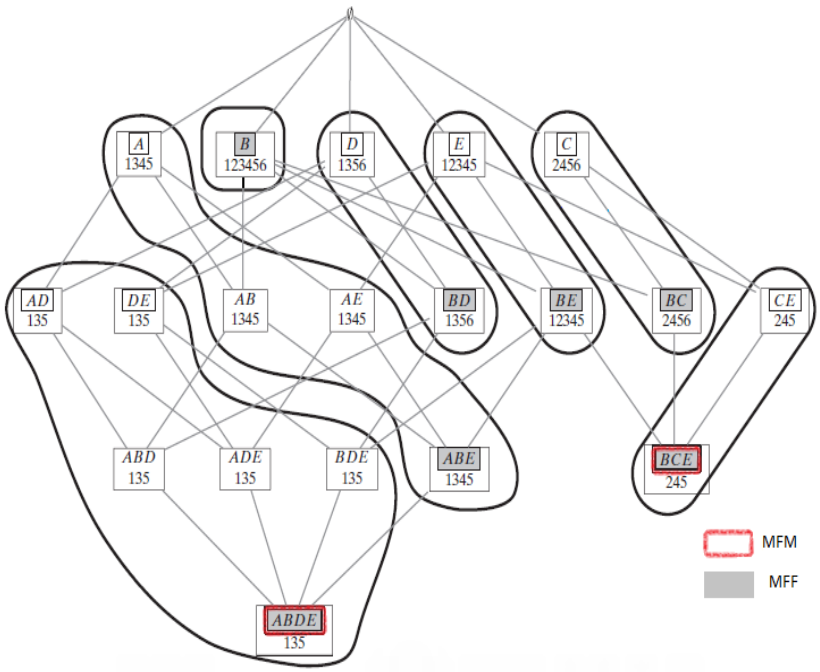

FIGURE 2.6 – les résultats des motifs fréquent fermé et maximal [19]

2.6 Algorithmes d'extraction de motifs fréquents maximaux

Nous décrivons brièvement, dans ce qui suit quelque algorithmes représentatifs de découverte de motifs fréquent maximaux :

2.6.1 Pincer-search

L'algorithme pincer-search proposé par Dao-I Lin et Zvi M. Kedem de l'université de New-York en 1997 [15]. Cet algorithme fait partie de la classe d'algorithmes d'extraction de motifs fréquents maximaux. Dans sa stratégie d'exploration de l'espace de recherche, il combine un parcours ascendant (Botton-up) et descendant (top-down), malgré que la direction principale de la recherche est toujours ascendante.
Le résultat est construire en observant la propriété Apriori lise dans le traitement des motifs fréquents maximaux de grande longueur.

2.6.2 Max-Miner

Max Miner est un algorithme pour extraire les ensembles des motifs fréquents maximaux présenté par Roberto J, Byardo en 1998 [4]. Pour ce faire, Max-Miner adapte deux heuristiques principales pour l'élagage de l'espace de recherche. Premièrement, en se basant sur la propriété Apriori, tout motif x qui inclut un sous-motif infréquent x' sera ignoré. En suite cet algorithme utilise une technique d'anticipation (Look A Head) en examinant touts, les branches dont les sur-motifs sont fréquents. En effet, si les derniers forment un ensemble Y ($X \cup Y$ est fréquent)

alors l'exploration des branches $y \in Y$ sera suspende en considérant que tous les sous-ensemble de $X \cup Y$ seront fréquents.
En départ de ces deux techniques réduit considérablement l'espace de recherche, Max-Miner exige toujours plusieurs passes sur l'ensemble de données ce qui de mense un inconvénient [9].

2.6.3 Depth-Project

Depth-Project proposé par Agrawal, Aggrawal et Prasad en 2000 [1] recherche l'ensemble des motifs fréquent maximaux en combinant deux techniques de recherche verticale et horizontale. Dans cet algorithme, les deux méthodes d'élagage des sous-motifs infréquents et les sur-motifs fréquents sont également appliquées. La base de donné est représentée par un bitmap : chaque ligne du bitmap est un vecteur de bits reflétant la présence ou non d'un item dans cette transaction ; de même, une colonne est le vecteur de bits qui code les occurrences de l'item en question dans les déférentes transactions de la base.
Les résultat expérimentaux ont montré que Depth-Project dépasse Max-Miner grâce à cette représentation par bitmap qui a permet une accélération dans le calcul des support des motifs [9].

2.6.4 GenMax

GenMax proposé par Gouda Karam et Mohammed J.Zaki en 2005 [8] est un algorithme basé sur une recherche récursive pour extraire les ensembles des motifs fréquents maximaux. Cet algorithme utilise la représentation verticale de la base de données issue de l'algorithme ECLAT proposé par la même équipe.
GenMax a profité et exploité les optimisations de ses prédécesseurs MAX-Miner et MAFIA (voir chapitre suivant). De plus, l'algorithme GenMax propose d'autres techniques pour réduire le temps de recherche des motifs maximaux : la propagation des *diffsets* et la *progressive focusing* qui ont contribué a l'amélioration de l'efficacité de l'algorithme. Pour plus de détail veuillez ce référer à l'article [9].

2.6.5 MAFIA

Doug Burdick, Manuel Calimlim, et Johannes Gehrke ont proposé l'algorithme MAFIA en 2001 [5] à l'université de Cornell pour extraire les motifs fréquent maximaux. La description complète de l'algorithme MAFIA est présentée dans le chapitre suivant.

2.7 Conclusion

Dans ce chapitre, nous avons défini la problématique de la fouille de motifs fréquents et les différents concepts liés à ce problème. Nous avons aussi présenté les principales approches et algorithmes d'énumération de motifs fréquents d'une base de transactions. Nous avons vu qui l'ensemble des travaux sont classés dans trois approches : horizontales, verticales ou projectives. Une section a été réservé aux algorithmes visant l'extraction de représentations compactes des motifs fréquents (fermés et maximaux). En fin du chapitre, nous avons dressé des descriptions succinctes de quelque algorithme d'extraction de motifs fréquents maximaux comme préalable a notre étude.
Le chapitre suivant sera consacré a l'étude et l'implémentation de MAFIA un algorithme célèbre d'extraction des motifs fréquents maximaux.

Chapitre 3

MAFIA et son implémentation

3.1 Introduction

L'extraction des motifs fréquents maximaux est une représentation compressée qui a été implémentée dans de nombreux algorithmes. Il sont fondés sur les approches de base que nous avons étudié, à laquelle s'ajoutent d'autre optimisations pour accélérer le processus de fouille.

Dans ce chapitre, nous allons étudié un algorithme pour extraire des motifs fréquents maximaux appelée MAFIA de façons détaillée afin de comprendre les optimisations et les structures utilisées, puis nous décrivons une implémentation partielle de cet algorithme par le langage JAVA.

3.2 MAFIA

L'algorithme MAFIA (MAximal Frequent Itemset Algorithm)est un algorithme pour extraire des motifs fréquents maximaux à partir d'une base de données transactionnelle, proposé par Doug Burdick, Manuel Calimlim, Johannes Gehrke en 2001 [5]. cet algorithme est efficace particulièrement lorsque les motifs de la base sont trop longs.

MAFIA adopte une stratégie efficace qui combine une exploration en profondeur et des mécanismes d'élagage de l'espace du problème. Pour la représentation de la base de données, il utilise la représentation verticale sous forme de bitmaps compressés.
Dans la suite nous expliquons les principes sur lesquels se base l'algorithme MAFIA [5].
✓ L'ensemble d'items $I = \{1, ...N\}$ est supposé totalement ordonné selon l'ordre lexicographique.
✓ L'ordre sur l'ensemble I sera utilisé dans l'énumération de la collection de motifs qui peut être représentée par un arbre. La racine de cet arbre est le motif null et chaque nœud inclut un motif appelé sa tête(Head abrégé H) et un ensemble d'extensions possible de nœud appelé queue(Tail abrégé T). Considérons le nœud P sur la figure 3.1, la tête de P est $\{A\}$ et la queue est l'ensemble $\{B, C, D, E\}$. Notons que l'union de la tête avec la queue (HUT) d'un nœud constitue les items qui peuvent apparaitre dans le sous arbre enraciné par ce nœud. Dans l'exemple précédent HUT de P est $\{A, B, C, D, E\}$[5].

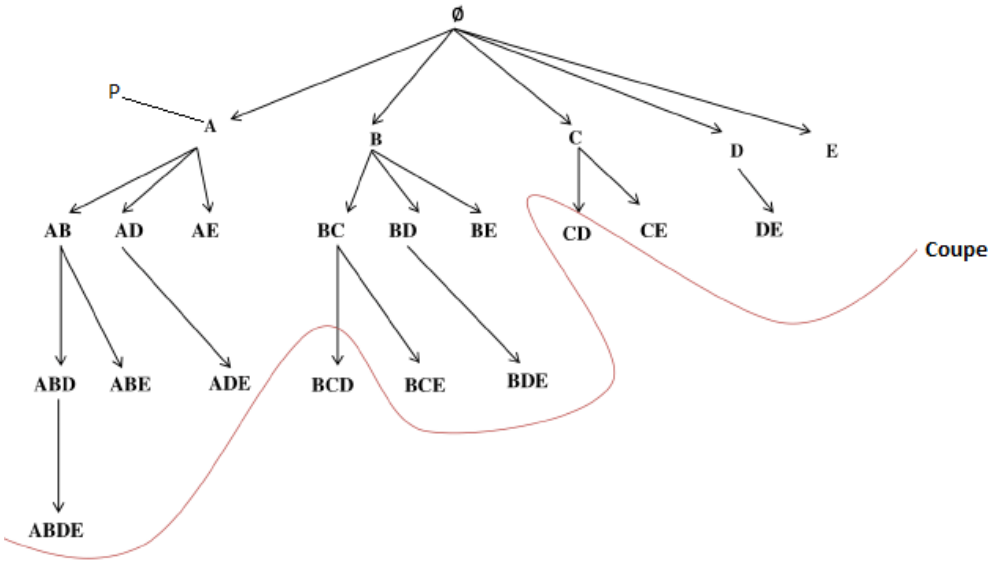

FIGURE 3.1 – La représentation de l'exemple dans un arbre

✓ problème de l'extraction des motifs fréquent peut être vu comme la détermination d'une coupe dans l'arbre, tel que tous les motifs au-dessus sont fréquents (AB), et ceux au-dessous sont infréquents (BCD) comme le montre la figure 3.1

3.3 Représentation des données

Contrairement à la plupart des algorithmes de fouille de motifs fréquents qui ont utilisé une disposition horizontale, MAFIA utilise la représentation verticale par des bitmaps de la base(la présence d'un item est marqué par un 1 et sont absence par un 0), afin d'accéléré les calculs des supports [5]. Pour notre exemple de référence, cette représentation est schématisée dans la table 3.1 et la table 3.2.

x	A	B	C	D	E
T(x)	1	1	2	1	1
	3	2	4	3	2
	4	3	5	5	3
	5	4	6	6	4
		5			5
		6			

TABLE 3.1 – Représentation verticale

T	A	B	C	D	E
1	1	1	0	1	1
2	0	1	1	0	1
3	1	1	0	1	1
4	1	1	1	0	1
5	1	1	1	1	1
6	0	1	1	1	0

TABLE 3.2 – Représentation Bitmap

3.4 Calcul des supports

Les transactions étant représentées par des vecteurs de bits, les supports sont calculé en effectuant le *Et* logique (*And*) entre les bitmaps. La valeur du support est alors le nombre de bits à 1 du résultat [5].

Par exemple le vecteur de bits des items A, B, C de tableau 3.2 sont : 101110,111111 et 010111. Pour calculer le support de motif AB on applique l'opération 101110 et 111111 = 101110 donc le support(AB) = 4. Le motif AC est infréquent car son vecteur de bits résultat est 101110 et 010111 = 000110 donc le support(AC) = 2.

3.4.1 Compression et Bitmaps projetés

Le point faible d'une représentation vertical pour représenter le Bitmap en particulier aux niveaux de support les plus bas, il y a beaucoup de zéros puisque l'absence et la présence de motif dans une transaction doivent être représentées [5].
Cette série de zéros est une source d'inefficacité car nous allons effectuons des opérations gaspillées sur des zones contenant des informations inutiles.
Notez que nous avons seulement besoin d'informations sur les transactions contenant les motifs X pour calculer le support de nœud n, Si la transaction T ne contient pas le motif X il ne fournira pas informations utiles pour compter les supports des enfants de n. Donc, conceptuelle ment, nous pouvons supprimer le bit pour la transaction T de X et tous les éléments dans la queue de n, C'est une forme de compression des bitmaps verticaux pour accélérer les calculs [5].
Le pseudo-code décrivant le processus peut être trouvé dans l'algorithme6.

Algorithme 6 PROJECT$(X, TAIL)$

Entrée : Bitmap X, et node's $TAIL$
Sortie : ensemble de bitmaps projetées I' et X'

```
pour item i ∈ TAIL faire
    Créer un bitmap vide I'
    pour c fairehaque transaction T
        si bit T de X est On alors
            Ajouter le bit T de I à I'
        fin si
    fin pour
fin pour
Créer X'- un bitmap rempli avec 1 et la taille du support(X)
    retourner l'ensemble des bitmaps projetés I' et X'
```

L'opération de projection est effectuée lorsque le support de l'ensemble d'éléments X tombe en dessous d'un certain seuil de reconstruction (rebuilding-threshold) exprimé en pourcentage de la taille globale de bitmap X (en bits).

3.4.2 Optimisations et élagage

Dans cette section, nous décrivons les différents composants de l'algorithme MAFIA dons le pseudo code est montré dans Algorithme 7 et les méthodes d'élagage utilisées pour réduire l'espace de recherche.

Algorithme 7 MAFIA($C, MFI, IsHUT$)

Entrée : Nœud actuel C, et l'ensemble des motifs fréquents maximaux MFI, $BooleanIsHUT$
Sortie : L'ensemble MFI des motifs fréquents maximaux.

$HUT = C.head \cup C.tail$
si $HUT \in MFI$ **alors**
Arrêtez la génération d'enfants et retour Calcul tous les enfants, utilisez le PEP pour couper la queue et réordonnez en augmentant le support
fin si
pour chaque item i dans $C.trimmed_tail$ **faire**
$IsHUT$ = si je suis l'enfant le plus à gauche dans la queue
$newNode \leftarrow C \cup i$
MAFIA($newNode, MFI, IsHUT$)
fin pour
si $IsHUT$ et toutes les extensions sont fréquentes **alors**
Arrêtez la recherche et revenez en haut de sous-arbre
fin si
si C est une feuille et $C.head$ n'est pas dans MFI **alors**
Ajouter $C.head$ à MFI
fin si

Afin d'illustrer les effets des techniques d'optimisation utilisées par MAFIA, commençons par décrire un parcours en profondeur simple (DFS).

1. **DFS simple**

L'algorithme 8 est un parcours DFS simple et récursif avec retour arrière (backtracking). Il traverse l'arbre lexicographique en profondeur, ou un nœud n est étendu par les items de sa queue.Si une extension est fréquente le parcours continue l'exploration d'autre extensions fréquente ; dans le cas contraire, l'exploration s'arrête (propriété Apriori). Si aucune des extensions d'un nœud n'est fréquente alors ce nœud est une feuille de l'arbre et peut être considéré comme un motif fréquent maximal candidat. Il est inséré dans le résultat si aucun de ses sur-motifs de ce candidat n'existe pas déjà dans l'ensemble MFI [5].

Algorithme 8 SIMPLE(C, MFI)

Entrée : Nœud actuel C, et l'ensemble des motifs fréquents maximaux MFI
Sortie : L'ensemble MFI des motifs fréquents maximaux

pour item $i \in C.tail$ **faire**
$newNode \leftarrow C \cup i$
si newNode est fréquent **alors**
SIMPLE($newNode, MFI$)
fin si
fin pour
si C est une feuille et $C.head$ n'est pas dans MFI **alors**
Ajouter $C.head$ à MFI
fin si

Exemple :

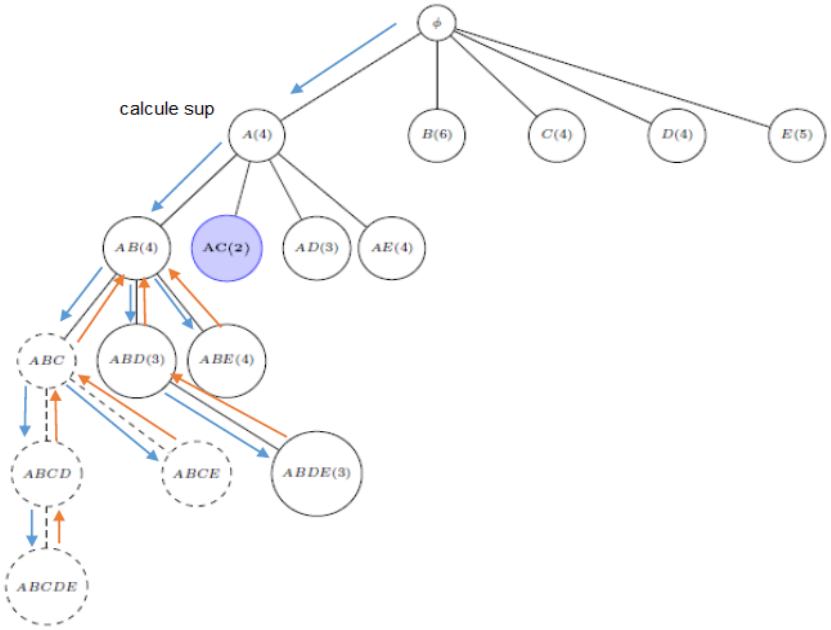

FIGURE 3.2 – Déroulement de DFS pour l'exemple référencé

2. **Élagage d'équivalence des parents (PEP)**

Une méthode d'élagage consiste à comparer les supports de chaque nœud avec ses fils. Soit x la tête du nœud n et y un élément de sa queue. Si $t(x) \subseteq t(y)$, alors toute les transactions contenant x contient également y. Cela garantit que n'importe quel ensemble les motifs fréquents z contenant x mais non y possède $(z \cup y)$ comme sur-motifs fréquent[5]. Puisque nous cherchons des motifs fréquents maximaux, il n'est pas nécessaire de considérer les motifs contenant x et non y. Par conséquent, nous déplaçons l'élément y de la queue à la tête pour le nœud n comme cela est illustré dans l'algorithme 9

Algorithme 9 PEP(C, MFI)

Entrée : Nœud actuel C, et l'ensemble des motifs fréquents maximaux MFI
Sortie : L'ensemble MFI des motifs fréquents maximaux

```
pour item i ∈ C.tail faire
    newNode ← C ∪ i
    si newNode.support == C.support alors
        Déplacer i de C.tail à C.head
    sinon
        si newNode is frquent alors
            PEP(newNode, MFI)
        fin si
    fin si
fin pour
si C est une feuille et C.head n'est pas dans MFI alors
    Ajouter C.head à MFI
fin si
```

Exemple : On considéré la base de donnée T de l'exemple précisent, pour explique cette

méthode de l'élagage dans le figure 3.3 suivant :
✓ calcul le support de A et leur extension AE, AB, AD. Si le support de A égale le support de l'un des extensions donc arrête et déplacé à la tête, par exemple : $sup(A) = sup(AB) = sup(AE) = 4$ donc déplacé à la tête $\{ABE\}$ et continué par le noeud AD

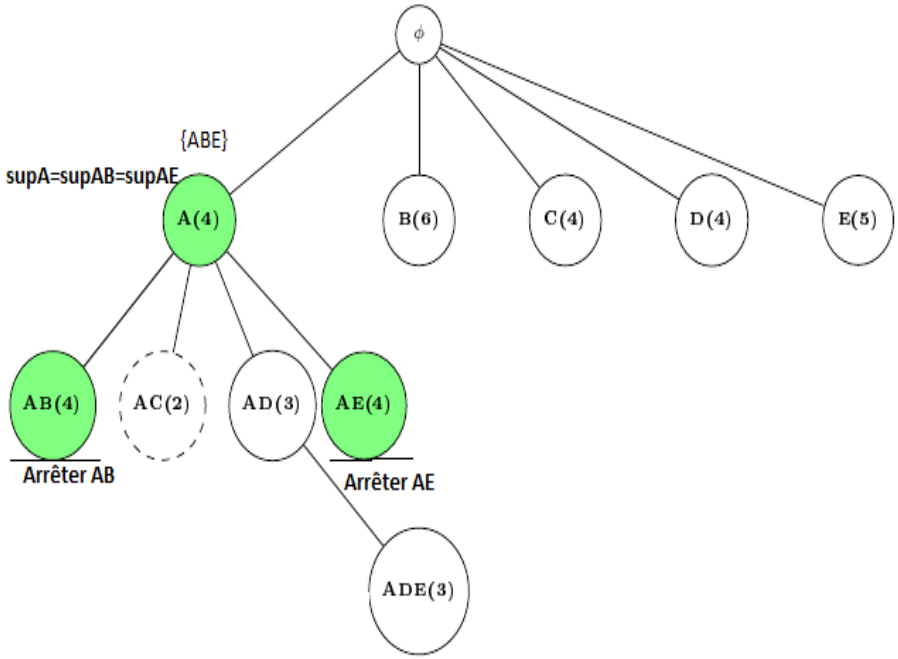

FIGURE 3.3 – L'heuristique PEP

3. **Élagage basé sur la fréquence de la tête union la queue (FHUT)**

Si pour un nœud n donné on peut observé que l'union des items formant sa tête et sa queue (HUT) et fréquente alors il n'est plus nécessaire d'explorer le sous arbre enraciné à n car cette union est le motif le plus long qui peut être généré (en explorant toujours le fils le plus à gauche) à partir de n [5]. Voir l'algorithme 10.

Algorithme 10 FHUT(C, MFI)

Entrée : Nœud C, et l'ensemble des motifs fréquents maximaux $\mathcal{MFI}$,$BooleanIsHUT$
Sortie : L'ensemble MFI des motifs fréquents maximaux

```
pour item i ∈ C.tail faire
    newNode ← C ∪ i
    IsHUT == si je suis l'enfant le plus à gauche dans la queue
    si newNode is frquent alors
        FHUT(newNode, MFI, IsHUT)
    fin si
fin pour
si C est une feuille et C.head n'est pas dans MFI alors
    Ajouter C.head à MFI
fin si
si IsHUT et toutes les extensions sont fréquentes alors
    Arrêtez d'explorer cette sous-arborescence et remontez
    l'arborescence lorsque IsHUT a été remplacé par True.
fin si
```

Exemple :

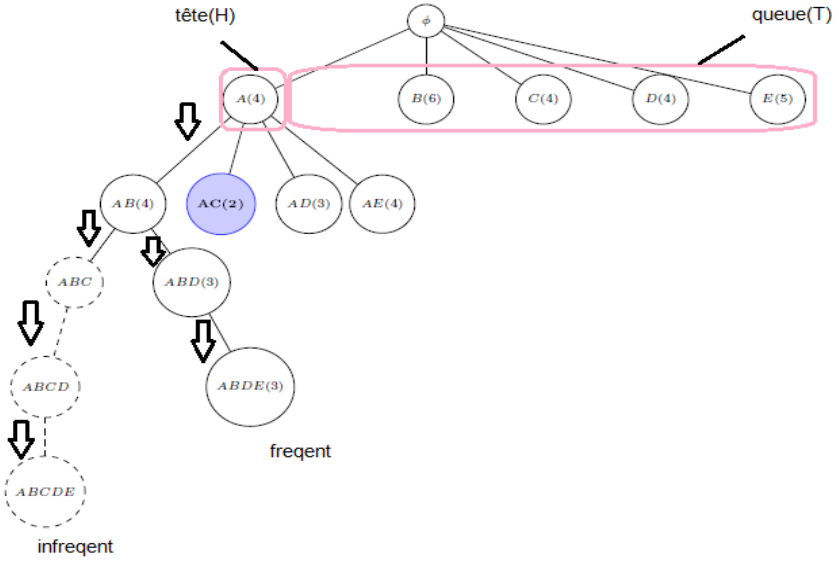

FIGURE 3.4 – FHUT

4. **Élagage basé sur la fréquence de sur-motif (HUTMFI)**

   Pour déterminer si un motif est fréquent ou non il existe deux méthode : calculer directement son support ou bien vérifier si un sur-motif est déjà déclaré fréquent. Dans FHUT on utilise la première méthode ; ici on utilise le test de sur-motif. Autrement dit, si le HUT d'un nœud n est un sous motif d'un autre motif déjà dans MFI, donc tout l'arbre enraciné dans n doit être élagué [5]. Cette heuristique est appelée HUTMFI est montrée dans l'algorithme 11.

Algorithme 11 HUTMFI(C, MFI)

Entrée : Nœud C, et l'ensemble des motifs fréquents maximaux $\mathcal{MFI}$
Sortie : L'ensemble MFI des motifs fréquents maximaux

```
nom HUT == C.head ∪ C.tail
si HUT ∈ MFI alors
    Arrêtez de chercher et revenir
fin si
pour item i ∈ C.tail faire
    newNode ← C ∪ i
    si newNode is frquent alors
        HUTMFI(newNode, MFI, IsHUT)
    fin si
fin pour
si C.head n'est pas dans MFI alors
    Ajouter C.head à MFI
fin si
```

Exemple :

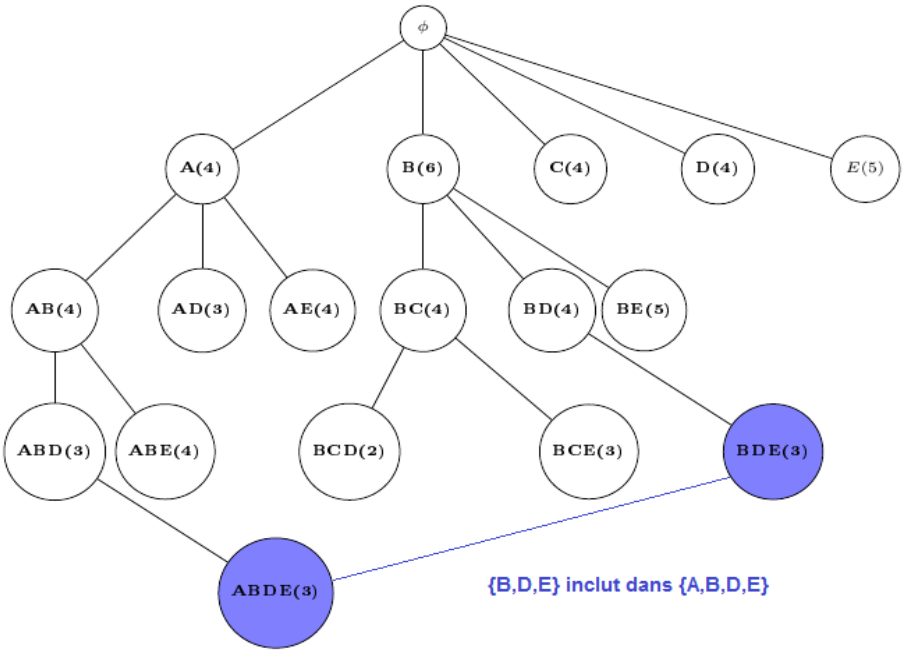

FIGURE 3.5 – L'heuristique HUTMFI

5. **Réordonne-ment dynamique**

   Pour améliorer l'efficacité de l'algorithme il est constaté qui l'ordre lexicographique pure est lourd, puisque la queue d'un nœud peut inclure des items infréquents qui peuvent augmenter inutilement le temps de réponse de l'algorithme. Aussi, la façons de prise en charge des fils d'un nœud est primordial. MAFIA combine élimination des items infréquents et déplacement des items de la queue vérifiant l'élagage de PEP et tri des fils selon l'ordre croissant de leur supports comme noté par Bayardo [4].

3.5 Implémentation java

Dans cette section, nous décrivons notre implémentation de l'algorithme MAFIA. Nous avons choisi le langage JAVA au vu de ses propriétés intéressantes (Multi plateformes, Orienté Objets...) d'une part. Ce langage utilisé dans la bibliothèque open source SPMF

3.5.1 Environnement

Les codes de notre implémentation sont écrits en JAVA en utilisant l'EDI NetBeans version 8.0.2. Les tests sont effectués sur des ensembles des données aussi bien réels que synthétiques disponibles sur multiples source des données : le site FIMI [14], la plateforme SPMF [7], etc. Nos expérimentations ont été réalisés sur une machine HP avec processeur Intel(R) core (i3) 2.10 GHZ et une mémoire RAM 4 GO, fonctionnant sous le système d'exploitation $Windows8$ $64bits$.

3.5.2 Préparation de donnée

Nous devons importé un fichier de base de donnée par l'instruction *"FileToPatch"* pour faire l'exécution de l'algorithme et un fichier Output pour afficher les résultat.
Déclaration une variable "minsupport" le seuil minimal de support pour testé les items il est

fréquent ou non.
Création un collection des données HashMap pour stocker les données, le HashMap est une implémentation de collection Map que associe une clé(unique) à une valeur. Donc l'utilisation de cette collection dans l'algorithme qui facilité la récupération des items comme un clé et les transaction que apparait de ce item comme une valeur.
Nous avons aussi besoin dans cet algorithme pour définir la collection de donnée ArrayListe, est une tableau qui se redimensionnement automatiquement, il accepte tout les type d'objets.
Déclaration de deux variables pour calculer le tempes d'exécution de l'algorithme (*StartTimp*, *EndTimp*) sont utilisé dans *"currentTimeMillis"*.

3.5.3 Classes principales

1. **Classe Main** qui contient les instructions correspondant aux actions que l'on veut exécuter.
2. **Classe Node** est une enregistrement contient deux *Arraylist*, liste pour les items (*itemlist*) et liste pour les tidliste contient ce item (*tidsetlist*) et un variable entre pour les support de chaque item.
3. **Classe TransactionDataBase** est une classe contient les méthode (AddTransaction, LaodFile, Size, etc) principale pour lire la base de donnée, cette classe nous l'avons pris de bibliothèque SPMF nous modifier pour lire les base de donnée que contient les mots et les lettres.
4. **Classe Bitmap** est une enregistrement contient deux attribué *transactionIndex* et *isFound*, le premier pour indexé les transaction de la base de donnée dans l'enregistrement *Bitamp*, et le deuxième est une variable contient le nombre 0 changer par 1 qu'il item apparaît dans le transaction . Crée cette classe pour utilisé dans la méthode qui retourner la base de transaction comme bitmap.
5. **Classe Algo-Mafia** cette classe contient tout les méthodes qui manipuler l'algorithme, nous mentionnons chaque méthode à sa place.

3.5.4 Représentation verticale et bitmap

Dans cet algorithme nous avons besoin d'utiliser la représentation verticale pour faciliter le calcul de support donc nous avons organisé la base de donnée de manière verticale. Création une HashMap "*mapItemCount*" contient les item dans la clé et un set contient les transaction, pour placé chaque item et leur transaction il fout déterminé le support de chaque item qui appariée les transaction par la fonction *calculteSupportSingleItems* qui calcule le support des sanglotant items, voir algorithme 12.

Algorithme 12 CALCULATESUPPORTSINGLEITEMS($database, mapItemCount$)

Entrée : Base de transaction *databse*, et le HashMap *mapItemCount*
Sortie : Variable *maxItemId*

```
maxItemId ⟵ 0
n ⟵ database.size()
pour i = 0; i < n; i++ faire
    pour item : database.getTransaction().get(i) faire
        set < Intger > set = mapItemCount.get(item)
        si set == NULL alors
            mapItemCount.put(item, set)
        fin si
        set.add(i)
    fin pour
fin pour
    retourner maxItemId
```

Donc la sortie de cet algorithme qui remplit le HashMap par les items comme une clé et les transactions dont apparaît cet item comme set. l'affichage de HashMap va être conçu par les instructions spéciales de la collection Map aide de l'enregistrement de classe Node qui affiche la liste d'items et la liste de transaction. L'initialisation de HashMap par 0 donc en considérons les transaction commencer par 0.

```
==============  ALGO_MAFIA ==============
 Transactions count from database : 6
--> A |0|2|3|4

--> B |0|1|2|3|4|5

--> C |1|3|4|5

--> D |0|2|4|5

--> E |0|1|2|3|4
```

FIGURE 3.6 – La représentation Verticale

Nous avons aussi besoin à la représentation Bitmap qui représenter l'apparence de l'item dans les transaction, en déclaré la fonction *binaire* quels sont utilisés les attribue de classe *Bitmap* voir algorithme 13

Algorithme 13 BINAIRE($mapItemCount$)

Entrée : Une liste de item fpliste
Sortie : Une liste contient les valeur 0 ou 1 représenter la transaction

```
Kset : récupérer les clé de HashMap
pour S : Kset faire
    Création une ArrayListe : listRecordInformation
    Création une Object : itemsetBuffer contient les tidlist de chaque item
    pour i = 0; i < databse.size(); i++ faire
        Création une enregistrement recordInformation
        recordInformation.transactionIndex ⟵ i + 1
        pour j = 0; j < itemsetBuffer.length; j++ faire
            si i == itemsetBuffer[j] alors
                isfound = 1
                Break
            fin si
        fin pour
        Ajouter le résultat de recordInformation à listRecordInformation
    fin pour
    Remplier le HashMap binaryRecord par les item(clé) et listRecordInformation
fin pour
    retourner maxItemId
```

Le résultat de cette fonction est une représentation bitmap avec une liste contient 0 ou 1 qui représenter l'item il est trouvé dans les transaction ou non.

3.5.5 DFS

Cet algorithme utilisé le parcours en profondeur simple (voir l'algorithme 14) c'est à dire sons méthode d'élagage. L'implémentation de cette fonction basé sur ArrayListe et le min support, donc nous allons crée fp-liste de type Node qui contient tout les liste d'item et leur transaction.

Algorithme 14 DFS($fpList, \mu$)

Entrée : La liste $fpList$, et le min-support μ
Sortie : Les motifs fréquent $fpListArc$

```
pour i = 0; i < fplist.size; i++ faire
    Création une ArrayListe : fpListArc
    pour j = i + 1; j < fpList.size; j++ faire
        Création une enregistrement fPNode
        UNION (itemlist(i), itemlist(j)) et ajouté dans fPNode
        INTERSECTION (tidsetList(i), tidsetList(j)) et ajouté dans fPNode
        Récupéré le support de chaque item supportTIDlist
        si supportTIDliste ≥ support alors
            Ajouté fPNode dans fpListArc
            Ajouté fPNode dans maxFPList // pour le test de maximalité
        fin si
    fin pour
    DFS(fpListArc, μ)
fin pour
```

La fonction DFS utilisé deux méthode l'union et l'intersection, le premier l'union entre deux liste $fp(i)$ et $fp(j)$ et le deuxième entre les tidliste de deux items. le pseudo code de deux méthode il montré dans l'algorithme 15 et l'algorithme 16.
En même temps de l'exécution de la fonction DFS on utilise une condition qui teste les motifs de maximal dans l'ensemble des motifs fréquents qui générée par la recherche en profondeur.

Algorithme 15 UNION(fpI, fpJ)

Entrée : Deux liste fpI et fpJ
Sortie : Une liste contient l'union de deux liste *unionItemList*

```
Création une liste contient le résultat de l'union unionItemList
isfound ⟵ 0
pour i = 0; i < fpI.size; i++ faire
    Ajouté fpI(i) dans unionItemList
fin pour
pour j = 0; j < fpJ.size; j++ faire
    isfound ⟵ 0
    pour i = 0; i < fpI.size; i++ faire
        si fpI(i) == fpJ(j) alors
            isfound ⟵ 1
        fin si
    fin pour
    si isfound == 0 alors
        Ajouté fpJ(j) dans unionItemList
    fin si
fin pour
    retourner unionItemList
```

Le fonctionnement de la méthode union est :
✓ En ajout tous les élément de la liste fpI dans la liste *unionItemList*.
✓ Parcours la liste fpJ Par rapport le premier liste fpI s'il existe la valeur $isfound$ changé à 1 donc n'ajout pas dans la liste *unionItemList*, contrairement s'il ne existe pas la valeur $isfound$ reste 0 et en ajouté dans la liste *unionItemList*.
✓ À la fin de la déroulement de cette méthode cela devient notre une liste contient l'union entre deux items. Par exemple : item AB union L'item AC égale ABC.

Algorithme 16 INTERSECTION(fpI, fpJ)

Entrée : Deux liste fpI et fpJ
Sortie : Une liste contient l'intersection entre deux tidliste *tidsetList*

```
Création une liste contient le résultat de l'intersection tidsetList
pour j = 0; i < fpJ.size; j++ faire
    pour i = 0; j < fpI.size; i++ faire
        si fpI(i) == fpJ(j) alors
            Ajouté fpJ(j) dans la liste tidsetList
        fin si
    fin pour
fin pour
    retourner tidsetList
```

Le fonctionnement de la méthode intersection est :
✓ Parcours la liste de fpJ Par rapport à la liste fpI s'il existe le tidliste de fpJ dans la liste

de tidliste de fpI donc en ajouté dans la liste de résultat $tidsetList$.
✓ À la fin de déroulement la fonction retourné une liste contient les tidliste qui partagé deux item. Par exemple l'item $A = 1345$ et l'item $B = 123456$ donc l'intersection égal $AB = 1345$.

3.5.6 Test de maximalité

Puisque l'algorithme MAFIA est une algorithme de motifs fréquent maximaux il est nécessaire de rechercher le motifs maximale. Nous avant applique le test dans la fonction DFS.
Nous avons besoin de créer une liste static $maxFPList$ de type string que rempli par les candidats si il existe, en même temps de travaille de la fonction DFS. Crée la liste candidat de type node qui rempli par l'item Père de la liste $fpliste$ (voir l'algorithme 14).
Ensuite, le test de maximalité est réalisé dans d'autres fonctions booléenne qui retournée les motifs s'il inclut dans le sur-motifs ou non.

Algorithme 17 GetMaxFPList($maxfpList$)

Entrée : Une liste $maxfpList$
Sortie : Liste contient les motifs maximal

```
Création une liste contient les motifs maximal newMaxfpList
isfound ⟵ 0
pour i = 0; i < maxfpList.size; i++ faire
    isfound ⟵ 0
    pour j = 0; j < maxfpList.size; j++ faire
        si j! = i alors
            item1 ⟵ maxfpList(i).itemList
            item2 ⟵ maxfpList(j).itemList
            si ItemIsMax (item1, item2)== True  alors
                isfound ⟵ 1
                Break
            fin si
        fin si
    fin pour
    si isfound == 0 alors
        Ajouté maxfpList(i) dans newMaxfpList
    fin si
fin pour
vidé la liste maxFPList
maxFPList ⟵ newMaxfpList
```

Le résultat de cet algorithme pour notre exemple de référence est montré dans la figure 3.7

```
============  DFS ============
ABDE inexistant dans MFM
BCE inexistant dans MFM
L'ens. de MFM sont : 2 [ABDE, BCE]
```

Figure 3.7 – le résultat de l'algorithme avec le test de maximal

3.6 Conclusion

Dans ce chapitre, nous avons décrit les différentes optimisations et nous avons fait la description et la analyse de l'algorithme MAFIA.
En suite, nous introduisons une implémentation partielle qui inclut la représentation verticale et bitmap, la recherche des motifs fréquents en profondeur contient une test de maximalté pour prendre seulement les motifs fréquent maximaux de la base de données.

Conclusion générale

Aujourd'hui, les données sont devenues énorme d'une façon continue en raison du développement technologie que nous vivons maintenant. Afin de tirer profit de ces volumes importants de données, nous devons les analyser.

Au fil des années, de nombreuses techniques et méthodes ont émergé pour analyser ces données. Cependant, la majorité des méthodes se voie confrontées au problème relatif aux tailles de données et les échelles souvent très considérables. Dès lors, le recours aux approches intelligentes pour extraire les connaissances à partir de ces données représente actuellement une solution et un champ de recherche très actif.

Dans notre étude, nous nous sommes concentrés sur l'une des tâches les plus célèbres de la fouille de données à savoir l'extraction des règles d'associations, dont le but est de trouver des relations entre deux variables ou plus. L'accent été mis sur l'extraction des ensemble des motifs fréquents qui sont considérés comme l'étape la plus importante dans cette tâche.

La première recherche été dirigée sur l'extraction totale des ensembles des motifs fréquents à travers la description plusieurs approches et algorithmes, qui sont représentés dans les algorithmes d'approche des niveaux et de l'approche verticale, en plus l'approche par projection.

Au fil du temps, l'attention s'est orientée sur comment extraire des règles d'association pour les ensembles fréquents à partir d'un espace plus petit. Donc, il est apparu une nouvelle approche ce qui est connu les représentations compactes, qui sont divisée en deux groupes : les ensemble des motifs fréquents fermés (MFF) et les ensemble des motifs fréquents maximaux (MFM) et qui nous permet d'éviter l'énumération totale des éléments trouvés dans la première recherches, ce qui permet de réduire le coût de la consommation de stockage et le temps d'exécution.

Dans ce mémoire, nous avons mis en lumière les algorithmes de fouille de motifs fréquents maximaux. La dernière partie a été consacrée en particulier à l'algorithme MAFIA. D'abord, les différentes idées, structures de données et heuristiques de l'algorithme MAFIA ont fait l'objet de description et analyse.

Ensuite, une implémentation partielle a été introduite. Cette dernières inclut notamment : la représentation des données sous forme verticale, le calcul de support via des intersections et une exploration en profondeur simple. Le résultat final est obtenu en effectuant un test de maximalité pour certain motifs candidats.

Le travail que nous avons réalisé dans ce mémoire nous a confirmé que de multiple idées et astuces existent pour le problème de la fouille de motifs. Ces heuristiques sont partagés et communs à de nombreux algorithmes dont MAFIA. Il serait intéressant de répertorier dans une recherche à part l'ensemble de ces heuristiques et étudier leur remparts sur l'extraction

de tous les motifs fréquents, fréquents fermés ou fréquents maximaux. L'analyse isolera chaque heuristique et l'examinera en profondeur sur les données très volumineuse réelles ou synthétiques afin de les maîtriser.

Bibliographie

[1] Ramesh C Agarwal, Charu C Aggarwal, and VVV Prasad. Depth first generation of long patterns. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 108–118. ACM, 2000. 25

[2] Charu C Aggarwal and Jiawei Han. *Frequent pattern mining*. Springer, 2014.

[3] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994. 14

[4] Roberto J Bayardo Jr. Efficiently mining long patterns from databases. *ACM Sigmod Record*, 27(2) :85–93, 1998. 24, 33

[5] Douglas Burdick, Manuel Calimlim, and Johannes Gehrke. Mafia : A maximal frequent itemset algorithm for transactional databases. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, pages 443–452. IEEE, 2001. 25, 26, 27, 28, 29, 30, 31, 32

[6] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3) :37, 1996. vi, 4

[7] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Antonio Gomariz, Ted Gueniche, Azadeh Soltani, Zhihong Deng, and Hoang Thanh Lam. The spmf open-source data mining library version 2. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 36–40. Springer, 2016. 33

[8] Karam Gouda and Mohammed J Zaki. Genmax : An efficient algorithm for mining maximal frequent itemsets. *Data Mining and Knowledge Discovery*, 11(3) :223–242, 2005. 25

[9] Gösta Grahne and Jianfei Zhu. High performance mining of maximal frequent itemsets. In *6th International Workshop on High Performance Data Mining*, volume 16, page 34, 2003. 25

[10] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining : concepts and techniques*. Elsevier, 2011. vi, 5, 7, 8, 9

[11] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation : A frequent-pattern tree approach. *Data mining and knowledge discovery*, 8(1) :53–87, 2004. 18

[12] D.J. Hand, D.J.H.H.M.P. Smyth, H. Mannila, P.D.S.D.J. Hand, P. Smyth, and F. Bach. *Principles of Data Mining*. A Bradford book. CogNet, 2001. 3

[13] Daniel T. Larose and Thierry Vallaud. *Des données à la connaissance*. Bases de données. Vuibert informatique, Paris, impr. 2005. 3, 4, 6, 8

[14] Moshe Lichman et al. Uci machine learning repository, 2013. 33

[15] Dao-I Lin and Zvi M Kedem. Pincer-search : an efficient algorithm for discovering the maximum frequent set. *IEEE Transactions on Knowledge and Data Engineering*, 14(3) :553–566, 2002. 24

[16] Laszlo Szathmary. *Symbolic Data Mining Methods with the Coron Platform*. Theses, Université Henri Poincaré - Nancy 1, November 2006. I have implemented all the algorithms presented in my thesis in a unified software platform called Coron. The software is available at http ://coron.loria.fr/ . :tel-01754284

[17] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005. :books/aw/TanSK2005

[18] Khaled Tannir. *L'algorithme FP-Growth*. http://blog.khaledtannir.net/2012/07/lalgorithmefp-growth-les-bases-13/. :2013

[19] Mohammed J. Zaki and Jr. Wagner Meira. *Data Mining and Analysis : Fundamental Concepts and Algorithms*. Cambridge University Press, May 2014. vi, 12, 13, 14, 16, 18, 20, 21, 22, 24

[20] Mohammed Javeed Zaki. Scalable algorithms for association mining. *IEEE transactions on knowledge and data engineering*, 12(3) :372–390, 2000. 17