UNIVERSITE MONTPELLIER II

Fouille de données : Règles séquentielles

Encadrantes :

Mme. LAURENT Anne
Mme. TEISSEIRE Maguelonne

Etudiants

Mr. ALLIA Mohamed Rachid
Mlle. BOUADI Tassadit
Mr. El MOUTAOUKIL Sami
Mr. KEIRA Mamadou

Table des matières

Remerciements

Ce TER a été réalisé en collaboration avec l'équipe fouille de données du Laboratoire Informatique Robotique et Micro-électronique de Montpellier (LIRMM).

Nous tenons avant tout à exprimer nos vifs remerciements à **Lisa DI JORIO** pour toute l'aide qu'elle a apportée à la compréhension de notre sujet, sa disponibilité, toutes ses explications éclairées, ainsi que son investissement et enthousiasme concernant nos travaux. Elle a su, tout au long de ce projet, nous motiver afin que l'on fournisse les meilleurs résultats possibles.

Nous remercions Madame **Anne LAURENT** et Madame **Maguelonne TEISSEIRE,** nos deux encadrantes, pour leur implication plus que dévouée au bon déroulement de ce projet.
Elles nous ont apporté leur rigueur, leur participation à la rédaction et à la correction de ce rapport. Leur esprit critique ainsi que leurs conseils avisés sur la présentation de nos travaux nous ont été très utiles et nous ont permis de contenir notre enthousiasme afin de ne pas nous éloigner du sujet.

1 Introduction :

Dans le cadre de notre Master Informatique Unifiée Professionnel et Recherche première année, nous avons choisi au second semestre le TER intitulé : « Fouille de données : Règles séquentielles », dont le but était d'arriver à extraire toutes les règles séquentielles à partir d'un ensemble de séquences fréquentes générées par l'algorithme d'extraction de motifs séquentiels VPSP (Aurélien Serra, 2006), développé par l'équipe TATOO au sein du LIRMM.

Durant ces dernières années, les quantités de données collectées, dans divers domaines d'application, deviennent de plus en plus importantes.

Ces quantités ont suscité le besoin d'analyse et d'interprétation afin d'en extraire des connaissances utiles. Dans cette situation, la fouille de données se propose de donner les outils et les techniques nécessaires pour l'extraction de ces connaissances.

Deux classes de motifs se sont alors avérées très utiles et simultanément utilisées dans la pratique, à savoir :

- Les itemsets fréquents
- Les motifs séquentiels fréquents

L'extraction des motifs séquentiels permet la détection et l'analyse des comportements fréquents de différents acteurs lors de multiples évènements, exemple :

- 60% des gens qui achètent une télévision achètent un magnétoscope plus tard

Par contre, la notion de causalité n'apparaît pas dans les motifs séquentiels.

D'où l'intérêt et le but du travail qui nous a été assigné et qui est d'étendre ces méthodes au contexte des règles séquentielles où la relation de causalité est accompagnée par une relation temporelle.

Notre travail consiste dans un premier temps à étudier et à comprendre le fonctionnement de l'algorithme d'extraction de motifs séquentiels VPSP, dans un deuxième temps, à introduire la notion de confiance et l'implémenter dans VPSP tout en générant les règles séquentielles qui en découlent.

Par exemple : On extraira les règles du type « Les internautes ayant été intéressés par cet article ont plus tard été intéressés par tel autre » et grâce au concept de confiance on pourra avoir l'information suivante :

- 60 % des gens qui achètent un magnétoscope ont acheté auparavant une télévision ;
- 70 % des gens qui achètent une carte mémoire ont acheté auparavant un téléphone portable.

Dans la première partie de ce rapport, nous aborderons les concepts généraux afin de mieux appréhender le sujet ;
Dans la deuxième partie, nous ferons le point sur les différents algorithmes d'extraction de motifs séquentiels existants;
Dans la troisième partie, nous expliquerons le déroulement de l'algorithme VPSP ;
Dans la dernière partie, nous détaillerons l'implémentation de la solution proposée ;
Et pour finir, nous conclurons tout en proposant des perspectives.

2 Gestion du projet :

Afin de mener à bien notre projet nous avons choisi de promouvoir le travail collaboratif et le travail en équipe c'est-à-dire à quatre.

Différentes étapes ont été nécessaires à la bonne conduite de notre projet.

Ces dernières sont décrites dans ce qui suit:

- **Tâche 1 : Documentation**
  Description : recherche de documentation concernant notre sujet et des différents concepts appliqués à la fouille de données.
  Résultats : cette étape nous a permis de bien cerner les concepts théoriques nécessaires à la bonne compréhension de notre sujet.

- **Tâche 2 : Compréhension de VPSP**
  Description : étude du fonctionnement de l'algorithme VPSP.
  Résultats : bonne compréhension de VPSP qui nous permettra d'intégrer l'implémentation de notre solution.

- **Tâche 3 : Recherche de solutions**
  Description : cette étape consiste à étudier différentes approches répondant à notre problématique.
  Résultats : obtention d'une solution optimale et répondant à nos attentes.

- **Tâche 4 : Etude de la solution retenue**
  Description : étude de la faisabilité de la solution.
  Résultats : solution et algorithme prêt à être implémentés.

- **Tâche 5 : Implémentation**
  Description : implémentation et mise en œuvre de la solution
  Résultats : objectifs atteints, algorithme fonctionnel.

- **Tâche 6 : Rédaction du rapport**
  Description : rédaction et correction du rapport
  Résultats : rapport du TER

Tâches	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Documentation	■	■	■							
Compréhension de VPSP				■	■					
Présentation au LIRMM					■	■				
Recherche des solutions						■				
Etude de la solution retenue							■			
Implémentation								■	■	
Rédaction du rapport										■

Tableau 1 Tâches du projet

Concepts théoriques

3 Concepts généraux

Dans cette section, nous parlerons des concepts ou termes souvent employés dans ce rapport, on expliquera les différentes étapes de la recherche d'une règle d'association, celle des règles séquentielles en passant par les motifs séquentiels.

1.1. Définitions

3.1.1 Item et itemset :

Un item peut être défini comme un article et un itemset un ensemble d'articles.

3.1.2 Transaction :

Une transaction est un ensemble d'items achetés par un client C à une date précise. Dans une base de données une transaction est représentée par trois attributs : idClient (identifiant d'un client), idDate (un identifiant pour une date), itemset (un ensemble d'items non vide).

Clients	Dates	Itemsets
Client1	02/03/2008 03/04/2008	Pain, TV beurre
Client2	10/02/2008 11/02/2008	Lecteur DVD Dattes, pain
Client3	12/02/2008 13/02/2008	Pain Beurre

Tableau 2 Un exemple de base de données contenant 4 transactions

3.1.3 Support minimal :

Le support minimal est le nombre minimum d'occurrence d'un motif séquentiel pour être considéré comme fréquent. L'occurrence n'est prise en compte q 'une fois dans la séquence. C'est un seuil choisi par l'utilisateur.

3.2 Recherche de règles d'association :

Une règle d'association est une relation d'implication X→Y entre deux ensembles d'articles X et Y.
Cette règle indique que les transactions qui contiennent les articles de l'ensemble X ont tendance à contenir les articles de l'ensemble Y.
X est appelé *condition* ou *prémisse* et Y *résultat* ou *conclusion.*

L'extraction des règles d'association est l'un des principaux problèmes de l'ECD (Extraction de Connaissances à partir de Données). Ce problème fut développé à l'origine pour l'analyse de base de données de transactions de ventes. Chaque transaction est constituée d'une liste d'articles achetés, afin d'identifier les groupes d'articles vendus le plus fréquemment ensemble.

Ces règles sont intuitivement faciles à interpréter car elles montrent comment des produits ou des services se situent les uns par rapport aux autres. Ces règles sont particulièrement utiles en marketing. Les règles d'association produites par la méthode peuvent être facilement utilisées dans le système d'information de l'entreprise. Cependant, il faut noter que la méthode, si elle peut produire des règles intéressantes, peut aussi produire des règles triviales (déjà bien connues des intervenants du domaine) ou inutiles (provenant de particularités de l'ensemble d'apprentissage).

La recherche de règles d'association est une méthode non supervisée car on ne dispose en entrée que de la description des achats.
On peut dire donc qu'une *règle d'association* est une règle de la forme : Si **condition** alors **résultat**. Dans la pratique, on se limite, en général, à des règles où la *condition est une conjonction d'apparition d'articles* et le *résultat est constitué d'un seul article* (ABDELALI Mouad, 2003). Par exemple, une règle à trois articles sera de la forme : Si X et Y alors Z ; règle dont la sémantique peu être énoncée : Si les articles X et Y apparaissent simultanément dans un achat alors l'article Z apparaît.
L'extraction des règles d'association peut être décomposée en quatre étapes qu'illustre la Figure 1 Les étapes d'extraction de règles d'association suivante :

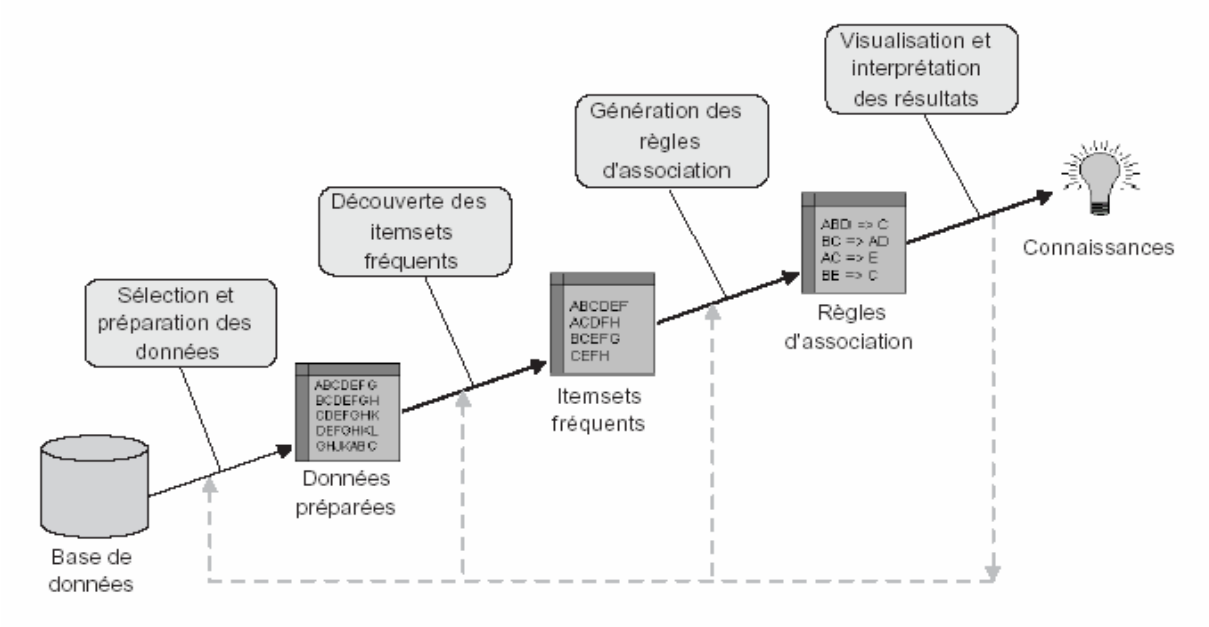

Figure 1 Les étapes d'extraction de règles d'association (ABDELALI Mouad, 2003)

Décrivons ces quatre étapes :

3.2.1 Sélection et préparation des données :

Cette étape permet de préparer les données afin de leur appliquer les algorithmes d'extraction des règles d'association. Elle est constituée de deux phases :

- La sélection des données de la base qui permettront d'extraire les informations intéressant l'utilisateur. Ainsi la taille des données traitées est réduite ce qui assure une meilleure efficacité de l'extraction.
- La transformation de ces données en un contexte d'extraction (il s'agit d'un triplet constitué d'un ensemble d'objets, d'un ensemble d'itemsets et d'une relation binaire entre les deux).

La transformation des données sélectionnées en données binaires améliore l'efficacité de l'extraction et la pertinence des règles d'association extraites.

3.2.2 Découverte des itemsets fréquents :

C'est l'étape la plus coûteuse en terme de temps d'exécution car, le nombre d'itemsets fréquents dépend exponentiellement du nombre d'items manipulés (pour n items, on a 2^n itemsets potentiellement fréquents).

Dans le Tableau 2 Un exemple de base de données contenant 4 transactions, en fixant le support minimal à 50% c'est-à-dire : pour qu'un item soit considéré fréquent, il faut qu'il soit acheté par au moins deux clients. On aura :
Le pain a été acheté par les trois clients et le beurre par les clients *client1* et *client3.*
Donc les itemsets fréquents sont : {pain, beurre}.

3.2.3 Génération des règles d'association :

A partir de l'ensemble des itemsets fréquents pour un seuil minimal de support *minsup*, la génération des règles d'association est un problème qui dépend exponentiellement de la taille de l'ensemble des itemsets fréquents.
Du Tableau 2 Un exemple de base de données contenant 4 transactions1, on a la règle suivante : Pain → Beurre

3.2.4 Visualisation et interprétation des règles d'associations :

Elle met entre les mains de l'utilisateur un ensemble de déductions fiables qui peuvent l'aider à prendre une décision.

3.3 Recherche de règles séquentielles :

Une règle séquentielle est une règle d'association à laquelle on rajoute le facteur temps. La recherche de règles séquentielles est un processus complexe et passe par différentes étapes notamment, la recherche de motifs séquentiels, c'est pourquoi dans cette section nous commencerons par la description d'une séquence, d'une séquence fréquente et d'un motif séquentiel.

3.3.1 Séquence :

Une séquence est une liste ordonnée d'itemsets non vides. Contrairement à la théorie ensembliste des règles d'association dont les éléments ne sont pas ordonnés, une séquence utilise le principe de précédence c'est-à-dire chaque élément de la liste est précédé des éléments qui l'ont précédé dans les transactions d'un client donné.

3.3.2 Fréquence d'une séquence :

Une séquence est considérée fréquente, si le support de cette séquence respect le support minimum, en d'autres termes, si le support de cette séquence est supérieur ou égal au support minimum. Celui-ci est introduit par le client afin de mesurer la pertinence d'une séquence.

Exemple : Le MinSupp est fixé à 66% (c'est-`a-dire deux clients minimum, sur les trois, doivent supporter la séquence). Du Tableau 2 Un exemple de base de données contenant 4 transactions, en considérant l'item « pain », il est supporté par les clients *client1* et *client3*.

Remarque : Pour chaque client, le support d'une séquence ne peut être incrémenté que d'une unité, même si la séquence apparaît plusieurs fois.

3.3.3 Séquences fréquentes maximales ou motifs séquentiels :

Une fois toutes les séquences fréquentes trouvées, on procède à la recherche de celles qui ont une fréquence maximale, c'est-à-dire celles qui ne sont incluses dans aucune autre séquence, on les appellera des motifs séquentiels.
Les motifs séquentiels peuvent être vus comme une extension de la notion de règles d'association, intégrant diverses contraintes temporelles. Aussi, la recherche de tels motifs consiste à extraire des ensembles d'items, couramment associés sur une période de temps bien spécifiée. En fait, cette recherche met en évidence des associations inter-transactions, contrairement à celle des règles d'association qui extrait des combinaisons intra-transaction. Dans ce contexte, et contrairement aux règles d'association, l'identification des individus ou objets est indispensable, afin de pouvoir suivre leur comportement au cours du temps.
Par exemple, des motifs séquentiels peuvent montrer que :

- "60% des gens qui achètent une télévision achètent un magnétoscope plus tard".
- "75%des gens qui achètent du pain achètent du beurre ".

Ce problème, posé à l'origine dans un contexte de marketing, intéresse à présent des domaines aussi variés que les télécommunications (détection de fraudes), la finance, ou encore la médecine (identification des symptômes précédant les maladies), ou la biologie (reconstitution de séquences d'ADN manquantes).

3.3.4 Extraction des motifs séquentiels

L'extraction des motifs séquentiels est un problème difficile, si plusieurs techniques algorithmiques ont permis l'extraction efficace des ensembles fréquents dans les données transactionnelles, ces résultats ne se transposent pas facilement à l'extraction des motifs séquentiels.
L'ordre des éléments doit être conservé dans une séquence, ce qui demande des efforts supplémentaires par rapport aux ensembles fréquents.
Face à cette problématique, plusieurs auteurs ont proposés des structures de données efficaces mais, la complexité intrinsèque reste non polynomiale.

La problématique de l'extraction des motifs séquentiels dans une base de données, est une sorte d'extension de celle des règles d'association. Une notion est ajoutée par rapport aux règles d'association qui est la prise en compte de la temporalité dans les enregistrements. Cela permet une plus grande précision dans les résultats, mais implique aussi une plus grande difficulté d'implémentation.
La prise en compte de la temporalité permet d'organiser les éléments traités avec une notion d'ordre (chronologique).

La recherche de motifs séquentiels se base sur un format de données bien précis, qui relate des événements liés à différents acteurs. L'exemple du supermarché est le plus utilisé pour illustrer ce concept. Cela est, en partie, dû au fait que ce sont des besoins de type marketing qui ont apporté cette problématique.
Ce format de données est illustré par le Tableau 3 Format de données utilisé ci-dessous, où l'on peut distinguer les achats de trois clients. Le but d'un algorithme d'extraction de motifs séquentiels sera alors de trouver des comportements fréquents, dans les achats des clients. Le résultat attendu est donc une ou plusieurs séquences d'achats, représentant un comportement récurrent.
Nous pouvons alors définir le comportement fréquent comme étant un comportement respecté par au moins n clients (n étant considéré comme un minimum de clients qui respectent ce comportement afin que celui-ci soit estimé fréquent).

Client	01/04/2008	01/05/2008
Client1	Pain, beurre	huile
Client2	Pc	Clé USB
Client3	Tv	Lecteur DVD

Tableau 3 Format de données utilisé

3.3.5 Propriétés des séquences fréquentes

Il existe trois propriétés principales qui sont des éléments déterminants pour l'extraction. Il faut noter que toutes ces propriétés sont déjà appliquées sur les règles d'association.

3.3.5.1 L'anti-monotonie :

L'anti-monotonie signifie que si une séquence quelconque S viole une contrainte C, alors toute sur-séquence S' de S la viole aussi. L'exemple le plus connu de contrainte anti-monotone est la contrainte de fréquence minimale, si une séquence n'est pas fréquente, aucune de ces sur-séquences ne le sera.

Ce type de contrainte permet un élagage dans l'espace de recherche et la diminution du nombre de séquences à considérer.

Les contraintes succinctes permettent d'énumérer les candidats sans nécessairement explorer tout l'espace de recherche en générant toutes les séquences possibles.

3.3.5.2 La monotonie :

La monotonie est une propriété qui stipule que si une sous-séquence S' d'une séquence quelconque S satisfait une contrainte C, alors S le satisfait aussi. Ce type de propriété permet de génération de candidats efficace.

3.3.5.3 Support des sous-séquences :

Le support de S est supérieur ou égal au support de S' *« (supp(S)>=supp(S') »* car, toutes les transactions dans la base de données qui supportent S' supportent aussi nécessairement S.

3.3.6 Support d'une séquence :

Le support d'une séquence quelconque S est le pourcentage de clients qui supportent cette séquence S. c'est une mesure dite d'utilité.

Supp({ae} → {bc}) = supp({ae} U {bc})

3.3.7 Confiance d'une règle :

La confiance d'une règle est une mesure dite de précision, c'est la probabilité qu'on achète un certain nombre d'articles A sachant qu'on a déjà acheté B, soit la probabilité conditionnelle : p(A/B).

Conf({ae → {/bc}) = supp({abce})/supp({ae})

On voit immédiatement que la confiance se traduit par un rapport de support. La notion de confiance serra bien détaillée dans la partie mise en œuvre.

Il existe d'autres critères d'évaluation utilisant différentes formules comme, L'Intérêt, la Conviction, et le Cosinus.

Algorithmes existants

4 Algorithmes existants

Avant d'aborder le mode de fonctionnement de l'algorithme VPSP avec lequel le travail a été effectué, la présentation et la compréhension de certains algorithmes de type « Générer- Elaguer » (AGRAWAL R., March 1995) s'imposent.

4.1 L'algorithme AprioriAll :

A l'origine l'algorithme bien connu Apriori ne considérait que des transactions indépendantes. Des algorithmes traitant les séquences de transactions, c'est-à-dire les listes de transactions par client, ont été développés pour arriver à ce qu'on appelle AprioriAll.

L'algorithme AprioriAll est une adaptation de l'algorithme Apriori pour les séquences où la génération de candidats et les calculs de supports sont modifiés par rapport à la méthode de base.

4.2 L'algorithme Apriori :

L'algorithme Apriori, introduit par Agrawal et al. (1994) est un algorithme clé pour les règles d'association car il est à la base de la majorité des algorithmes servant à découvrir des règles d'associations plus complexes telles que les associations séquentielles et multidimensionnelles. Il tire son nom de son heuristique qui utilise l'information connue a priori sur la fréquence des items. Cette heuristique stipule que si A, un sous-ensemble d'items de l'ensemble I, ne possède pas le support minimal, il ne peut être engagé dans une règle d'association avec tout autre item ij de l'ensemble I, ij $\notin$ A. Ainsi, si A est peu fréquent la règle A $\Rightarrow$ ij l'est également et il est donc inutile d'examiner toute règle d'association où A est impliqué. Le problème consistant à identifier des règles d'associations se divise en deux étapes : une étape de jointure et une autre d'élagage.

4.2.1 L'étape de jointure :

Pour trouver les itemsets fréquents dans la base de données transactionnelle, l'algorithme Apriori effectue plusieurs balayages de la base de données(en anglais pass). Le premier balayage sert à identifier les candidats Ck, un ensemble d'itemsets, et à compter le nombre de fois qu'apparaît chaque item, c'est-à-dire leur support respectif. Tous les items dont le support est plus grand qu'une valeur prédéterminée appelée min_sup, sont conservés afin de former Lk, l'ensemble des k-itemsets fréquents. Cet ensemble sert d'amorce pour générer l'ensemble de candidats Ck+1.

L'ensemble Ck+1, qui regroupe les (k+1)-itemsets, est généré en liant Lk avec lui-même. Pour que deux k-itemsets puissent être liés, ils doivent posséder k-1 items en commun.

Par conséquent la liaison de deux 1-itemsets ne requiert aucun élément en commun, alors que le liaison de deux 3-itemsets requiert 2 éléments en commun. Les deux 1-itemsets {1} et {2} peuvent être liés ensemble pour générer le 2-itemset {1,2}. Le 3-itemset {1,2,3} peut être lié avec {2,3,4} pour générer {1,2,3,4}, mais ne peut pas être lié avec {3,4,5} puisque seul l'items {3} est commun aux itemsets {1,2,3} et {3,4,5}.

Il est fort possible qu'un itemset généré ne respecte pas le seuil de support minimal. Si c'est le cas, cet itemset est éliminé lors de l'étape d'élagage.

4.2.2 L'étape d'élagage :

Une fois l'ensemble des candidats Ck+1 généré, le support de tous les (k+1)-itemsets est calculé. Tous les (k+1)-itemsets ∈ Ck+1 dont le support ne dépasse pas le min_sup sont retirés de la liste des candidats. Comme la liste des candidats Ck+1 est réalisée à partir de la liste antérieure des candidats Ck, tout candidat retiré à l'étape k n'est plus considéré dans l'étape k+1 comme le stipule l'heuristique d'Apriori.

4.2.3 Exemple de l'algorithme Apriori :

Voici un exemple détaillé des étapes suivies par l'algorithme Apriori. La base de données utilisée pour cet exemple est celle qui se retrouve dans le Tableau 4. Cette base de données contient 10 transactions et pour cet exemple, le support minimal est fixé à 30 % soit un décompte minimal requis de 3 transactions. Les résultats détaillés de chacune des étapes sont illustrés dans le Tableau 5.

No_transaction	Items_achetés
10	1, 2, 5
20	2 , 4
30	2 , 3
40	1 , 2 , 4
50	1 , 2 , 3
60	2 , 3 , 5
70	1 , 3
80	1 , 2 , 3 , 5
90	1 , 2 , 3
100	2 , 3

Tableau 4 : exemple d'une base de données contenant 10 transactions.

1. Lors de la première itération, l'algorithme compte le support de chaque 1-itemset de la base de données. Ces itemsets forment l'ensemble des candidats C1 qui sert à générer l'ensemble L1, c'est à dire l'ensemble des 1-itemsets fréquents.

2. Premier élagage : L'algorithme compare ensuite la fréquence de chaque 1-itemset avec la fréquence minimale prédéterminée ici à 3 ou un support de 30 %. Tous les itemsets ayant une fréquence inférieure à 3 sont retirés et ceux dont la fréquence est supérieure ou égale à 3 sont conservés afin de générer l'ensemble L1. Les 1-itemsets fréquents qui forment L sont dans ce cas-ci {1}, {2}, {3} et {5}.

3. Étape de jointure : Les 1-itemsets de l'ensemble L1 sont utilisés pour générer les candidats C2. La génération des candidats est réalisée en liant l'ensemble L1 avec lui-même. Comme il s'agit de 1-itemsets, le nombre de combinaison possible est de n(n-1)/2, n étant le nombre d'itemsets. Dans cet exemple, le nombre d'itemsets étant de 4, six candidats sont formés. Les candidats sont {1,2}, {1,3}, {1,5}, {2,3}, {2,5} et {3,5}.

4. Calcul du support : Lorsque les 2-itemsets candidats ont été générés, l'algorithme effectue un autre balayage de la base de données afin de calculer la fréquence respective des candidats. Cette fréquence est inscrite dans une table comme illustré dans le Tableau 5 : étapes de l'algorithme Apriori avec un support minimal de 30%.2.

C1 **Itemset** **Fréquence** {1} 6 {2} 9 {3} 7 {4} 2 {5} 3		**Retirer les candidats** **---------->>** **Dont la fréquence est inférieure à 3**	**L1** **ItemSet** **Support** {1} 6 {2} 9 {3} 7 {5} 3	**Générer les candidats** **---------->>** **C2 à partir des L1**
C2 **ItemSet** **{1,2}** **{1,3}** **{1,5}** **{2,3}** **{2,5}** **{3,5}**	**Effectuer un balayage de** **---------->>** **la BD pour compter la fréquence des candidats**	C2 Itemset Fréquence {1,2} 5 {1,3} 4 {1,5} 2 {2,3} 6 {2,5} 3 {3,5} 2	**Retirer les** **---------->>** **candidats dont la fréquence est inférieure à 3**	L2 Itemset Fréquence {1,2} 5 {1,3} 4 {2,3} 6 {2,5} 3
C3 **ItemSet** **{1,2,3}** **{1,2,5}** **{2,3,5}**		**Effectuer un balayage De la BD pour compter** **---------->>** **La fréquence des candidats**	C3 ItemSet Fréquence {1,2,3} 3	**Retirer les** **---------->>** **candidats dont la fréquence est inférieure à 3**
L3 **ItemSet** **{1,2,3}**		**Générer les candidats** **---------->>** **C4 à partir de L4**	C4 ItemSet {}	**Aucun candidat n'est généré et donc l'algorithme se termine ici.**

Tableau 5 : étapes de l'algorithme Apriori avec un support minimal de 30%.

5. Deuxième élagage. L'algorithme va par la suite parcourir l'ensemble C2 afin d'éliminer tous les 2-itemsets ayant une fréquence inférieure à 3. Dans ce cas-ci, les 2-itemsets {1,5} et {3,5} sont retirés. Les autres 2-itemsets sont conservés et forment l'ensemble L2, c'est-à-dire l'ensemble des 2-itemsets fréquents.

6. Génération des candidats. La génération des 3-itemsets est réalisée en fusionnant L2 avec lui-même. Comme cette fusion implique des 2-itemsets, les itemsets doivent avoir 1 itemset (2-1) en commun. De plus, tous les sous-ensembles de (k-1)-itemsets formés doivent être fréquents Si un k-itemset généré est composé de (k-1)-itemsets non fréquents, celui-ci est automatiquement éliminé, ce qui évite de calculer son support ou sa fréquence. Dans l'exemple du Tableau 5 : étapes de l'algorithme Apriori avec un support minimal de 30%.2, {1,2} et {2,5} peuvent être fusionnés pour former l'itemset {1,2,5} Les sous-ensembles de {1,2,5} sont {1,2}, {1,5} et {2,5}.

Comme {1,5} ne figure pas parmi l'ensemble L2 des 2-itemsets fréquents, l'itemset {1,2,5} ne peut pas avoir une fréquence supérieure à 3. Il est donc retiré des candidats et son support n'est pas comptabilisé. L'itemset {2,3,5} est également élagué car {3,5} ne figure pas parmi l'ensemble L2 des 2-itemsets fréquents.
À la fin de cette étape, seul l'itemset {1,2,3} est généré.

7. Calcul du support. Un troisième balayage de la base de données sert à calculer la fréquence de l'itemset {1,2,3}, et cette dernière est de 3.

8. Étape d'élagage. L'algorithme compare la fréquence de l'itemset {1,2,3} avec la fréquence minimale. Comme {1,2,3} possède la fréquence minimale, celui-ci est conservé et devient le seul itemset de L3, l'ensemble des 3-itemsets fréquents.

9. Génération des candidats. Étant donné que l'ensemble d'amorce L3 ne contient qu'un seul itemset, soit {1,2,3}, aucun 4-itemset candidat ne peut être généré. L'algorithme se termine ici.

Les associations identifiées par l'algorithme sont celles formées par les itemsets de l'ensembleL2 et de l'ensemble L3 soit {1,2}, {1,3}, {2,3}, {3,5}et {1,2,3}. Ces itemsets fréquents engendrent les règles d'associations. Ainsi, l'itemset {1,2} engendre les règles d'associations « 1 => 2 » et « 2 =>1 » alors que l'itemset {1,2,3} engendre les associations « 1 => 2 => 3 », « 1 => 3 => 2 », « 2 => 1 => 3 », « 2 => 3 => 1 », « 3 => 1=> 2 » et « 3 => 2 => 1 ».

4.2.4 Discussion et limites de l'algorithme Apriori :

En examinant l'algorithme Apriori, on peut en tirer les conclusions suivantes :

- Premièrement, l'étape décisive de la performance de l'algorithme est l'étape de jointure, car elle requiert plusieurs lectures de la base de données.
- Deuxièmement, l'algorithme ne tient pas compte de la quantité achetée (dans le cas d'une analyse de panier d'achat).
- Troisièmement, l'algorithme Apriori ne peut traiter directement les variables continues, comme l'âge ou le revenu.

Pour traiter de telles variables, l'analyste doit préalablement les regrouper en catégories, avant de les soumettre à l'algorithme.
Finalement, les dimensions temporelles ou séquentielles ne sont pas prises en charge par l'algorithme. Ainsi, l'algorithme Apriori ne peut déceler une association entre les transactions faites par un même client qui achèterait du pain en matinée et du beurre en après-midi, lors de deux transactions différentes, sauf si l'unité de base est le client au lieu de la transaction.

4.3 L'algorithme SPADE :

Spade, présenté dans (ZAKI M., 2001), se classe dans la catégorie des algorithmes qui cherchent à réduire l'espace des solutions en regroupant les motifs séquentiels par catégorie. Pour Spade, les motifs fréquents présentent des préfixes communs qui permettent de décomposer le problème en sous-problèmes qui seront traités en mémoire.
Le calcul de F2 (les fréquents de taille 2) par Spade, passe par une inversion de la base qui la transforme d'un format vertical vers un format horizontal. Il gère les candidats et les séquences fréquentes à l'aide de classes d'équivalence comme suit : deux k-séquences appartiennent à la même classe si elles présentent un suffixe commun de taille (k-1). Plus formellement, soit Pk-1(α) la

séquence de taille k-1 qui préfixe la séquence α.Comme α est fréquente, Pk-1(α) ⍰ Fk-1 les fréquents de taille k-1.
Une classe d'équivalence est définie de la manière suivante : [ρ ⍰ Fk-1] = {α ⍰ Fk| Pk-1(α)=ρ}. Chacune de ces classes d'équivalence contient alors deux types d'éléments : [ρ.l1]=< ρ (x) > ou bien [ρ.l2]=< ρ x > selon que l'item x appartient ou pas à la même transaction que le dernier item de ρ.
Les candidats sont ensuite générés selon trois critères : Auto jointure ([ρ.l1] x [ρ.l1]),
Auto jointure ([ρ.l2] x [ρ.l2]) et jointure ([ρ.l1] x [ρ.l2]).
Le reste de l'algorithme, à savoir le comptage du support pour les candidats générés, repose sur la réécriture préalable de la base de données. En effet, la transformation consiste à associer à chaque k-séquence l''ensemble des couples (client, itemset) qui lui correspondent dans la base.

4.3.1 Limite de SPADE

1. La nécessité d'une très grande mémoire pour transformer et puis après stocker toute la base de données.

2. Les temps de réponse au moment de compter le support des candidats générés à chaque étape est très intéressant.

3.4 L'algorithme GSP :

Les deux premiers algorithmes proposés par Agrawal et Srikant (1995) pour découvrir des règles d'associations séquentielles ont été les algorithmes AprioriAll et AprioriSome. Par contre, aucun intervalle ni fenêtre d'événement ne peuvent être spécifiés à l'aide de ces algorithmes.
Afin de corriger ces lacunes, Srikant et Agrawal (1996) ont amélioré ces deux algorithmes en y ajoutant la possibilité de définir une fenêtre d'événement ainsi qu'un intervalle ce qui a donné naissance à l'algorithme GSP (Generalized Sequential Pattern).
La fenêtre temporelle (nommée en anglais sliding-window) donne une plus grande flexibilité dans la définition d'une transaction en offrant la possibilité de considérer comme simultanées des transactions qui se déroulent à l'intérieur d'un laps de temps précis. L'intervalle, tel que défini par Srikant et Agrawal (1996) est donné par deux paramètres : le min-gap et le max-gap.
Le min-gap spécifie la borne inférieure de temps écoulé requis pour que les transactions d'une séquence soit considérées comme valides, alors que le max-gap spécifie la borne supérieure de temps écoulé. Si le temps écoulé entre deux itemsets d'une séquence est supérieur au max-gap, celle-ci n'est pas considérée comme une séquence valide et par conséquent son support n'est pas augmenté. Le min-gap doit toujours être supérieur à la fenêtre temporelle car celle-ci définit l'unité fondamentale de temps d'une transaction.
Le fonctionnement du GSP est similaire à celui de l'algorithme Apriori. Dans les prochaines sections, les étapes de l'algorithme GSP sont présentées et sont suivies d'un exemple détaillé de chacune de ces étapes.

3.4.1 Étapes de l'algorithme GSP :

L'algorithme débute par un tri de la base de données en utilisant comme clé primaire l'identifiant unique du client et comme clé secondaire la date de transaction.
Cette étape permet d'identifier toutes les séquences-clients. Chaque séquence-client devient une ligne de la base de données séquentielle, qui sera utilisée par la suite par l'algorithme GSP.

Il est à noter qu'à ce stade les paramètres donnés par l'intervalle et la fenêtre d'événement ne sont pas pris en considération.
Par conséquent, deux transactions se déroulant dans la même fenêtre d'événement demeurent distinctes et ne sont pas donc pas fusionnées.
L'algorithme effectue de multiples balayages de la base de données. Le premier balayage détermine le support de chaque item, c'est-à-dire le nombre de séquences-clients contenant au moins une fois l'item en question.
À la fin de chaque balayage, l'algorithme sait quels sont les 1-itemsets fréquents, c'est-à-dire ceux qui dépassent le seuil de support minimal.
Tous les balayages subséquents débutent avec un ensemble d'amorce : l'ensemble des séquences fréquentes identifiées au cours du balayage précédent. L'ensemble d'amorce sert à générer les nouvelles séquences potentiellement fréquentes, nommées séquences candidates.
Chaque séquence candidate contient un item supplémentaire que les séquences ayant servies d'amorce, de telle sorte que toutes les séquences candidates d'un balayage particulier possèdent le même nombre d'items. Le support de ces séquences candidates est par la suite déterminé par un autre balayage de la base de données. À la fin du balayage, les séquences candidates fréquentes forment l'ensemble des séquences fréquentes et servent à leur tour d'amorce pour générer les séquences candidates ayant un item supplémentaire. L'algorithme s'arrête lorsqu'aucune séquence fréquente n'est identifiée ou lorsqu'aucun candidat ne peut être généré à partir de l'ensemble d'amorce.
L'algorithme GSP se distingue de l'algorithme Apriori sur deux aspects : la génération des candidats et le calcul du support.

- **La génération des candidats :**

La génération des candidats se déroule en deux phases : la phase de jointure, qui produit l'ensemble de candidats après jointure, et la première phase d'élagage, qui produit l'ensemble de candidats après élagage. La phase d'élagage sert à éliminer toutes les séquences candidates qui, selon l'heuristique Apriori, ne peuvent être fréquentes et dont le calcul du support est donc inutile.

- **Le calcul du support :**

L'algorithme GSP augmente de 1 le support de la séquence s pour chaque séquence-client contenant au moins une fois la séquence s en question. Dans le cas où aucune fenêtre d'événement ni max-gap ni min-gap ont été définis, l'algorithme ne fait que calculer le nombre de séquences-clients qui contiennent les séquences candidates dont les supports doivent être déterminés. Par contre si un ou plusieurs de ces paramètres sont définis, l'algorithme doit s'assurer que les séquences respectent les contraintes imposées par les paramètres avant d'en augmenter le support.

3.4.2 Limites de l'algorithme GSP

Le GSP possède toutefois des faiblesses :

1. Une grande quantité de candidats peut être générée dans les grandes bases de données. À titre indicatif, une base de données contenant 1000 1-séquences formera 1 499 500 candidats (1000 x 1000 + 1000 x (1000-1)/2). Etant donné que les candidats générés sont formés à partir de la concaténation du fichier d'amorce, plusieurs de ces candidats ne se retrouveront pas dans la base de données, ce qui représente une perte de temps.

2. Une grande quantité de balayages de la base de données est requise. Étant donné que la longueur de chaque séquence candidate grandit d'un item à chaque balayage, l'identification d'une 15-séquence requiert 15 balayages de la base de données.
3. Les méthodes basées sur l'algorithme Apriori, comme c'est le cas pour le GSP, ont de la difficulté à découvrir de longues séquences. Ceci vient du fait que les longues séquences sont formées à partir d'un nombre important de séquences plus courtes et le nombre de candidats générés varie de manière exponentielle avec la longueur de ces derniers.

3.5 L'algorithme PSP :

Lors de la recherche des feuilles susceptibles de contenir des candidats inclus dans la séquence analysée, la structure utilisée ne tient pas compte des changements de date entre les items de la séquence qui servent à la navigation. Par exemple,avec la séquence < (A C) (B D) >, l'algorithme va atteindre la feuille du sommet C (fils de A), alors que cette feuille peut contenir deux types de candidats :

- ceux qui commencent par < (A) (C)... d'un côté
- et ceux qui commencent par < (A C) ... de l'autre.

Le but est alors de mettre en place une structure d'arbre de préfixes, pour gérer les candidats.
L'algorithme PSP (Prefix Tree for Sequential Pattern), destiné à exploiter cette structure, est basé sur la méthode générer-élaguer. Le principe de base de cette structure consiste à factoriser les séquences candidates en fonction de leur préfixe. Cette factorisation, inspirée pousse un peu plus loin l'exploitation des préfixes communs que présentent les candidats.
En effet les auteurs proposent de prendre en compte les changements d'itemsets dans cette factorisation. L'arbre de préfixes ainsi proposé ne stocke plus les candidats dans les feuilles, mais permet de retrouver les candidats de la façon suivante : tout chemin de la racine à une feuille représente un candidat et tout candidat est représenté par un chemin de la racine à une feuille.
De plus, pour prendre en compte le changement d'itemset, l'arbre est doté de deux types de branches. Le premier type, entre deux items, signifie que les items sont dans le même itemset alors que le second signifie qu'il y a un changement d'itemset entre ces deux items.

3.5.1 Limites de PSP

Le principal problème de PSP est le nombre de passes dans la base de données. Pour une séquence de longueur k, k passes sont effectuées.

Ceci provoque une perte de temps lors de l'étape de comptage du nombre de clients qui supportent une séquence donnée.

.

VPSP En Action

Dans les parties précédentes nous avons abordé les différents préliminaires nécessaire à la bonne compréhension de l'algorithme que nous allons utiliser dans le cadre de notre TER.

Un retour sur les algorithmes existants était nécessaire car ceux-ci constituant la base dont émane les principes de l'algorithme VPSP.

Cette partie décrit le fonctionnement et le déroulement de l'algorithme VPSP par étape.

En premier lieu nous expliquerons le principe de transformation de la base de données en un format vertical, ensuite nous détaillerons la structure de données utilisée et l'algorithme en question.

Pour finir nous discuterons des faiblesses et avantages de l'algorithme ainsi que du travail à effectuer dans le cadre de notre TER.

4 Fonctionnement de VPSP :

4.1 Transformation de la base de données façon SPADE :

A des fins d'optimisation du temps d'exécution, l'algorithme VPSP charge la base de données en mémoire pour n'effectuer par la suite qu'une seule passe sur celle-ci, ce qui implique une concession au niveau de la consommation de la mémoire.

La transformation proposée par SPADE simplifie le comptage du nombre de clients supportant une séquence, car cette opération nous permet de disposer d'un ensemble de données où l'accès aux informations (Client, Date de transaction) pour un item est facilité.

Exemple :

Dans cette section, nous utiliserons la base de données suivante comme exemple pour illustrer tous les concepts qui seront décrits par la suite.

Client	Date	Item
C1	17/04/08	A, B, C
C1	18/04/08	D
C2	17/04/08	B, C
C3	18/04/08	D
C4	19/04/08	C
C4	20/04/08	D

Tableau 6 : Représentation horizontale de la base de données

<A>	(C1, 17/04/08)
<B>	(C1, 17/04/08), (C2, 17/04/08)
<C>	(C1, 17/04/08), (C2, 17/04/08), (C4, 19/04/08)
<D>	(C1, 18/04/08), (C4, 20/04/08), (C3, 18/04/08)

Tableau 7 Représentation verticale de la base de données

La base de données horizontale représente la liste des items achetés par un client à une date donnée. En revanche, la base de données verticale nous permet d'extraire, pour un item donné, la liste des couples (Client, Date transaction) qui témoignent de son apparition dans la base de données.

Exemple :

Considérons le client **C2** et la date de transaction **17/04/08**.
L'information qui est obtenue au niveau de la base verticale est la suivante :
« Le client **C2** a acheté à la date **17/04/08** les items **B** et **C** »
En revanche, la base de données verticale nous offre l'information suivante :

- L'item **B** a été acheté par le client **C2** à la date **17/04/08**.
- L'item **C** a été acheté par le client **C2** à la date **17/04/08**.

L'un des avantages premiers de cette transformation est la simplification de la recherche des séquences et motifs fréquents.
Voyons maintenant comment et avec quelle structure de données VPSP exploite cette représentation de la base de données.

4.2 VPSP:

La structure de données utilisée par l'algorithme VPSP (Vertical Prefix-Tree for Sequential Pattern), destiné à parcourir l'arbre des séquences candidates, combine l'utilisation de la structure d'arbre préfixé au chargement de la base de données en mémoire, ce qui apporte un gain de performances considérable.

Dans ce qui suit, nous allons décrire l'algorithme VPSP ainsi que toutes les optimisations qu'il apporte au niveau de la navigation au sein de l'arbre des candidats.

4.2.1 Structure de données utilisée :

L'une des premières optimisations offertes par la structure de l'algorithme VPSP, réside dans le fait que la base de données n'est parcourue qu'une seule fois tout au long de l'algorithme permettant l'extraction des séquences et motifs fréquents.
Lors de cet unique parcours le premier niveau de l'arbre préfixé est construit, chaque nœud correspondant à un item de la base de données, garde une trace de la transaction correspondante à chaque apparition dans la base de données.

Exemple :

<A>	**(C1, 17/04/08)**
<B>	(C1, 17/04/08), (C2, 17/04/08)
<C>	(C1, 17/04/08), (C2, 17/04/08), (C4, 19/04/08)
<D>	(C1, 18/04/08), (C4, 20/04/08), (C3, 18/04/08)

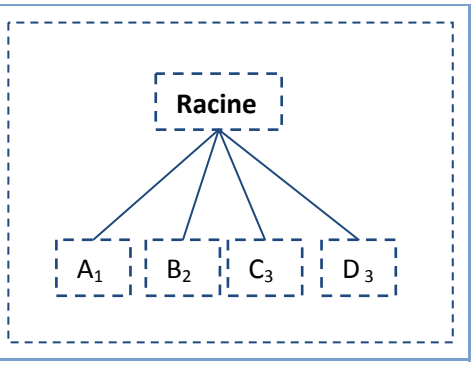

Figure 2 Projection de la représentation verticale de la base de données dans l'arbre préfixé de profondeur 1.

La Figure 2 schématise le fonctionnement de cette structure qui permet de n'effectuer qu'un seul passage dans la base de données.
En effet, les séquences de longueur 1 (items) conservent une représentation verticale de la base de données.
Nous commençons d'abords par présenter la façon dont sont stockés les items fréquents (K =1) puis nous passerons au cas où K> 1.

K= 1 :

Lors de cette première étape et comme illustré dans la Figure 4, chaque branche issue de la racine de l'arbre préfixé relie celle-ci à une feuille qui représente un Item.

Chacune de ces feuilles contient : l'item, son support (Nombre de clients qui ont acheté cet item) ainsi que sa séquence d'apparition (correspond à l'ensemble des transactions où l'item apparaît, ces transactions sont de la forme (Client, date de transaction)).

Ce qui facilite et accélère le calcul du support, il suffit juste de dénombrer les clients qui participent à l'incrémentation du support grâce à la séquence d'apparition.

1. Phase d'élagage :

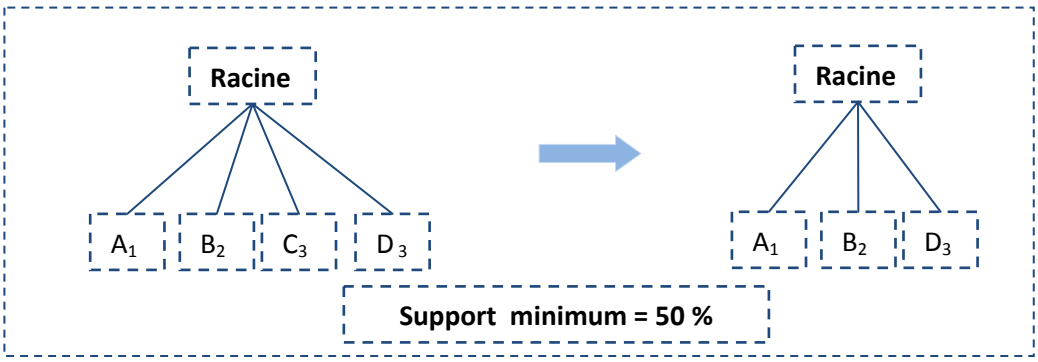

Figure 3 Phase d'élagage des items

L'arbre représenté à gauche de la Figure 5 illustre l'état de la structure après évaluation du support de chaque Item.

Considérons un support minimal égal à 50%, i.e. pour qu'une séquence soit retenue, elle doit apparaître au moins dans trois séquences de données.

L'arbre de droite de la même figure représente la structure contenant uniquement les Items fréquents. Considérons la feuille contenant l'item A dans l'arbre de gauche, son support a été évalué à 1, ce qui signifie que seulement 25% des clients supportent la séquence constituée de l'item A.

Cet item ayant un support inférieur au support minimal spécifié par l'utilisateur, il est éliminé des séquences candidates lors de la phase d'élagage.

K> 1 :

Chaque nœud de l'arbre représente un item pour une ou plusieurs séquences.
Chaque chemin de la racine de l'arbre vers une feuille représente une séquence.
Afin de pouvoir différencier les itemsets à l'intérieur d'une séquence (Ex. (B D) et (B) (D)), les fils d'un nœud sont séparés ont deux catégories comme dans PSP : « Same Transaction » « Other Transaction ».

Exemple :

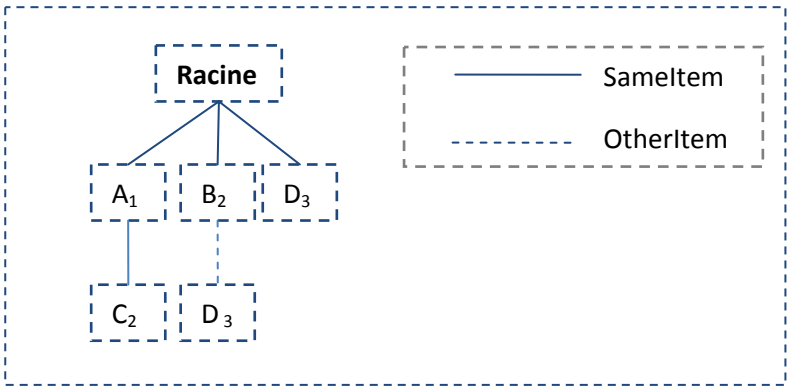

Figure 4 Séquences fréquentes de taille 1 et 2

L'arbre de la Figure 4 représente les séquences fréquentes de taille 1 et 2.
Une branche en **trait plein** entre deux items signifie que ces items font partie de la **même transaction** (< **(B C)** >), et une branche en **trait pointillé** marque le début d'un **nouvel itemset** dans la séquence (<**(C) (D)** >).
Les séquences fréquentes de longueur 2 représentées sont donc les suivantes :
<**(B C)**>, <**(C) (D)**> .
Le mécanisme de génération des candidats sera détaillé dans ce qui suit.

Propriété 1 : Sur un chemin de l'arbre la valeur du support augmente d'un nœud à l'autre.
Exemple : le support de la séquence < **(A) (C) (B)**> est inférieur ou égal au support de la séquence <(**A) (B)**>.

4.2.2 Algorithme VPSP :

Dans cette partie nous allons vous présenter l'algorithme VPSP (Aurélien Serra, 2006) en décrivant ces différentes étapes.

Algorithme VPSP :

Entrées: *une base de données, un support minimum minSupp*
Sorties: *l'ensemble des séquences dont le support est supérieur à minSupp*
depth ⟵0
//Génération des items fréquents
getFrequentItems() ;
depth ⟵depth + 1 ;
supportCount() ;
pruning() ;
//Génération des fréquents de longueur 2
twoCandidateGeneration() ;
depth ⟵depth + 1 ;
supportCount() ;
pruning() ;
Tantque *{k −candidats} != Ø* ***faire***
//Génération des fréquents de longueur k
candidateGeneration() ;
depth ⟵depth + 1 ;
supportCount() ;
pruning() ;
Fin Tantque

4.2.2.1 Génération des candidats :

Les candidats générés par l'algorithme VPSP correspondent à ceux déterminés par les algorithmes cités précédemment quelle que soit la longueur des candidats.

Nous allons maintenant détailler la génération des candidats par VPSP en fonction de leur longueur K.

K ∈ [1,2] :

Les candidats générés de longueur 1, correspondent aux items existants dans la base de données (Figure 2).

Les candidats de longueur 2 sont générés de la manière suivante :

Pour tout couple (X,Y) dans l'ensemble des items fréquents, si X= Y alors générer les séquences candidates suivantes : <(X) (Y)> ((X X) n'est pas possible car les itemsets sont définis comme étant des ensembles d'items).Sinon, si X≠Y alors on génère les 2_candidats suivants :
<(X Y)> et <(X) (Y)>.

Notons qu'à cette étape également, chaque séquence candidate de longueur 2 garde en mémoire ça liste d'apparition, et cela est généralisé pour les **K** séquences candidates.

Lors de la création des listes d'apparition pour les candidats d'ordre supérieur ou égal à 2, deux cas de figure se présentent à nous.
En effet, supposons que K≥ 2 soit la longueur de la séquence candidate pour laquelle on souhaiterai créer une liste d'apparition. Cette séquence candidate est construite à partir d'une séquence fréquente S de longueur K-1 et d'un item fréquent I et cela soit :

- En insérant l'item I dans le dernier itemset de la séquence S (SAME),
- En étendant la séquence S par un nouvel itemset formé de I (OTHER).

L'algorithme VPSP met donc en œuvre deux méthodes distinctes pour réaliser cette opération et créer les listes d'apparitions et dans ce qui suit nous vous présentons ces deux méthodes :

1. Ajout d'un item I dans le dernier Itemset de la séquence S :

   Dans ce cas la liste d'apparition du K candidat n'est autre que l'intersection de la liste d'apparition de la séquence fréquente S et celle de l'item fréquent I.

Génération de toutes les règles séquentielles : Méthode Sous Séquence Génération

Entrées: *S (k-1)-séquence, I item fréquent*
Sorties: *un ensemble de couples (Client, Date de transaction) pour lesquels le k-candidat apparaît dans la base*
lApparition ← Ø
Pour *chaque couple (Client, Date de transaction) (ci, dj) de S* ***faire***
Pour *chaque couple (Client, Date de transaction) (ck, dl) de I* ***faire***
Si *ci = ck et dj = dl* ***alors*** *lApparition ← lApparition ∪ (ci, dj)*
Fin si
Fin pour
Fin pour

2. Extension de S par un nouvel itemset formé de I :

   Cela signifie que la liste d'apparitions de la séquence candidate générée est constituée de tous les couples (Client, Date de transaction) de I où, pour un même client, la date de transaction est strictement postérieure à celle de la première apparition de S.

Construction de la liste d'apparitions d'un k-candidat "Other" :

Entrées: *S (k-1)-séquence, I item fréquent*
Sorties: *Un ensemble de couples (Client, Date de transaction) pour lesquels le k-candidat apparaît dans la base*
lApparition ← Ø
Pour *chaque client c qui incrémente le support de S* ***faire***
dS ← première date o`u c incrémente le support de S
Pour *chaque transaction t où c a incrémenté le support de I* ***faire***
dI ← t.date
Si *dI > dS* ***alors*** *lApparition ← lApparition ∪ (c, dI)*
Fin si
Fin pour
Fin pour

Exemple :

Considérons la séquence candidate < **(B C)** >. Cette séquence est générée en insérant l'item C dans le dernier itemset de la séquence <**(B)** >. La Figure 2 nous donne la liste d'apparitions de <(B)> : ({**(C1, d1)**, **(C2, d1)**}) et de C ({**(C1, d1)**, **(C2, d1)**, **(C4, d3)**}).

La liste d'apparitions de la séquence candidate est constituée de tous les couples (**Client**, **Date de transaction**) communs à <(B)> et <(C)>, soit ({**(C1, d1)**,**(C2, d1)**}).

Maintenant, considérons la séquence candidate <(C) (D)>. Cette séquence est générée en étendant la séquence <(C)> avec un nouvel itemset formé uniquement de l'item D.

La Figure 4 nous donne la liste d'apparitions de C ({**(C1, d1), (C2, d1)**, **(C4, d3)**}) et de D ({**(C1, d2), (C3, d2)**, **(C4, d4)**}).

La liste d'apparitions de la séquence candidate déterminée grâce à l'algorithme est constituée des **t** couples (Client, Date de transaction) suivants : ({**(C1, d2), (C4, d4)**}).

K> 2 :

Nous allons maintenant décrire comment se fait la génération des K candidats pour K> 2 avec VPSP.

L'algorithme dans VPSP qui se charge d'effectuer cette opération est l'algorithme « **CANDIDATEGENERATION** ».Pour chaque feuille **L** à étendre, l'algorithme recherche à la racine l'item **x** représenté par **L**. Ensuite, l'algorithme étend la feuille **L** en construisant pour cette feuille une copie des fils de **x**.

A cette étape de la génération, l'algorithme applique un filtrage, identique à celui effectué par PSP à cette étape, destiné à ne pas générer de séquences dont nous savons à l'avance qu'elles ne sont pas fréquentes. Pour cela, l'algorithme considère **F**, l'ensemble des fils de x et **B** l'ensemble des frères de **L**. Pour chaque **f** dans **F**, si **f** n'appartient pas à **B** alors il est inutile d'étendre **L** avec **f**.

En effet nous savons que si **f** n'est pas frère de **L**, alors si nous considérons **p** le père de **L**, alors (**p**, **f**) n'est pas fréquent et par conséquent (**p**, **L**, **f**) non plus.

***CANDIDATEGENERATION* :**

```
Entrées: l'arbre des candidats T de profondeur k (k ≥2) représentant les fréquents de
longueur inférieure ou égale à k
Sorties: T étendu à la profondeur (k+1), contenant les candidats de longueur (k + 1) à tester
NT = {N ∈ T/leaf(N) et N.depth = k /*ensemble des feuilles de profondeur k*/
Pour chaque N ∈ NT faire
   Pour chaque Nr ∈ root.children faire
      Si item(N) = item(Nr) et Nr.child ∈ N.brother alors
         N.children = Nr.childrenTN.brother
      Fin si
   Fin pour
   Pour chaque n ∈ N.children faire
     fillApparitionsList(n)
  Fin pour
Fin pour
```

L'algorithme VPSP procède de la même manière pour générer ses K candidats. Cependant, VPSP affecte pour chaque candidat trouvé une liste d'apparition ce qui simplifie le calcul du support de ce dernier par la suite.

Exemple :

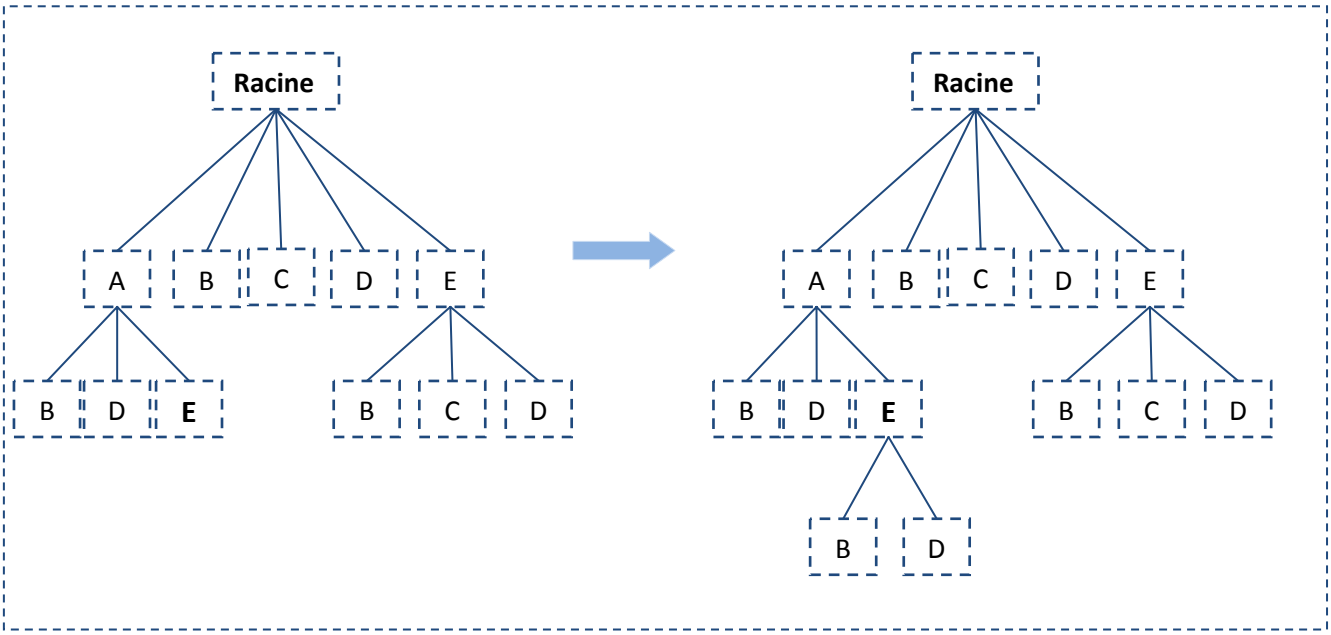

Figure 5 Optimisation de la génération des candidats

On va appliquer sur l'arbre de gauche de la Figure 5 l'algorithme de VPSP qui se charge de la génération des candidats.

Cette figure expose l'état de l'arbre préfixé avant et après la génération des candidats de longueur 3. La feuille représentant l'item E, en gras dans l'arbre des candidats de gauche (Figure 5), est étendue dans l'arbre de droite uniquement avec les items B et D.
Cependant, on remarque que l'item C n'est pas généré au niveau de la feuille E car C n'étant frère de E, la séquence <(A) (C)> n'est pas fréquente. Donc d'après la propriété de monotonie la séquence <(A) (E) (C)> n'est pas fréquente, il est donc inutile de générer ce candidat.

4.2.2.2 Suppression des candidats non fréquents :

L'algorithme VPSP utilise la méthode **Générer-Elaguer** utilisée par la majorité des algorithmes d'extraction de motifs séquentiels depuis AprioriAll, afin de générer son arbre préfixé.
Après chaque étape de génération, le support de chaque séquence est compté, puis une phase d'élagage permet de ne garder que les séquences fréquentes.

Dans ce qui suit nous allons détailler chacune de ces opérations.

1. **Calcul du support :**

   Le comptage du support est une opération qui est simplifiée et accélérée avec la structure utilisée par l'algorithme VPSP.

   En effet, cela consiste pour chaque candidat, à déterminer le nombre de clients qui ont participé à incrémenté le support. Ce calcul est facilité grâce à la liste d'apparitions dont dispose chaque candidat.

   Voici l'algorithme qui effectue cette opération pour une séquence :

SUPPORTCOUNT:

Entrées: *une séquence candidate S et sa liste d'apparitions listeApparitions dans la base de données*
Sorties: *S disposant de son support dans la base de données*
support ← 0
dernierClient ← ?
Pour *chaque couple Client-Date de transaction (ci, dj) ∈ listeApparitions* ***faire***
  Si *ci > dernierClient* ***alors***
    support ← support + 1
    dernierClient ← ci
  Fin si
Fin pour
S.support ← support

Remarque :
Cet algorithme suppose, pour fonctionner, que la liste d'apparitions est ordonnée par client croissant et par date croissante pour un même client.

Exemple :

Appliquons cet algorithme pour déterminer le support de la séquence candidate suivante :
<(B) (C)> dont la liste d'apparition est ((C1, d7), (C2, d4),(C3, d3), (C3, d5)).

Le support obtenu est égal à trois, ce qui correspond au nombre de clients qui supportent cette séquence. On remarque que le support diffère du cardinal de la liste d'apparitions en effet, nous retrouvons pour le client C3 deux apparitions de cette séquence aux dates d3 et d5, mais un client ne peut incrémenter le support d'une séquence que d'une unité au maximum.

Cette phase de comptage du support précède, à chaque étape de l'algorithme, une phase d'élagage qui permet de supprimer les séquences candidates non fréquentes.

2. Elagage des séquences candidates non fréquentes :

La phase d'élagage (**pruning**) permet de supprimer de l'arbre préfixé toutes les séquences générées dont le quotient **support**/(**Nombre de Clients**) est strictement inférieur au support minimum fixé par l'utilisateur.

PRUNING :

Entrées: *l'arbre des candidats T, de profondeur k contenant les candidats de longueur k à tester*
Sorties: *l'arbre des candidats T, de profondeur k représentant les h-fréquents (h ∈ [1..k])*
*NT = {N ∈ T/leaf(N) et N.depth = k /*ensemble des feuilles de profondeurk*/*
Pour *chaque N ∈ NT* ***faire***
    Si *N.support/CustomerCount < minSupp* ***alors***
        delete(N)
    Fin si
Fin pour

Exemple :

La Figure 3 Phase d'élagage des itemsillustre le fonctionnement de l'algorithme d'élagage. En effet, l'arbre de gauche représente les candidats de longueur 1 générés, à savoir les items suivants : <(A)>, <(B)>, <(C)>, <(D)>.
Après comptage du support et suppression des items non fréquents nous obtenons l'arbre de droite de la Figure 4 qui contient les candidats fréquents suivants : <(B)>, <(C)>, <(D)>.

Cette vérification est réalisée à la profondeur k ($k \geq 1$) pour l'ensemble des k-candidats.

4.2.3 Optimisations apportées par VPSP :

1. Déséquilibre de l'arbre :

VPSP reprend certaines optimisations réalisées par PSP. Ainsi, il conserve l'optimisation du déséquilibre de l'arbre puisqu'il évite de stocker des séquences redondantes.
Pour ce faire, un ordre est imposé sur les items des itemsets : les items au sein d'un même itemset sont classés par ordre lexicographique.

Exemple :

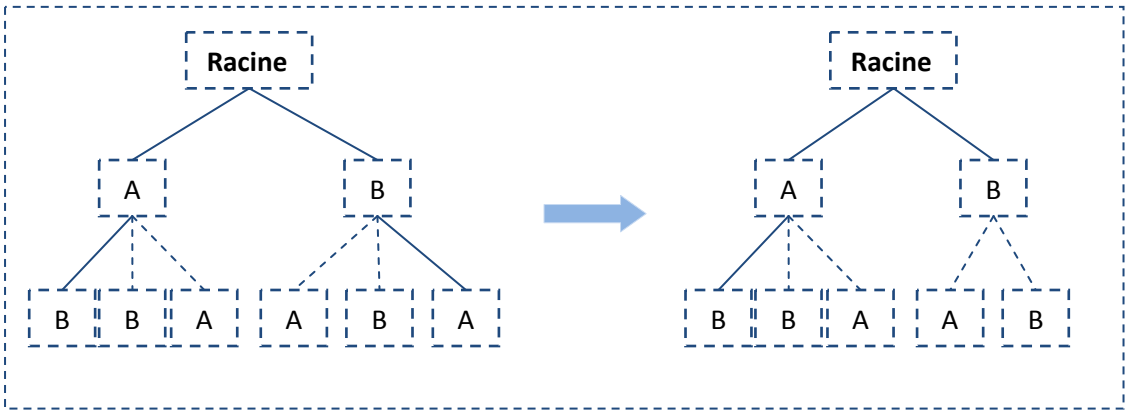

Figure 6 Déséquilibre de l'arbre

Dans l'arbre représenté à gauche de la Figure 6, aucun ordre n'est établi sur les itemsets. Nous remarquons que la séquence < (A B) > apparaît deux fois. En effet les séquences <(A B)> et <(B A)> désignent une même séquence dans notre problème. Dans l'arbre de droite de la figure 1, un ordre sur les itemsets est cette fois imposé. Chaque séquence ne peut figurer au plus qu'une seule fois dans l'arbre.

2. Limitation du nombre de candidats :

L'algorithme VPSP reprend aussi les optimisations de PSP qui permettent de limiter le nombre de séquences candidates et fait même plus pour accroître les performances.

Ainsi, l'algorithme PSP étend un nœud de profondeur k en générant pour fils les nœuds qui appartiennent à l'intersection des frères du nœud à ´étendre et des fils du fils de la racine portant le même item que le nœud à étendre. VPSP effectue un filtrage plus important puisque les candidats générés proviennent de l'intersection entre les frères du nœud à étendre et les fils de l'oncle du nœud à étendre qui porte le même item. Ce qui réduit le nombre de candidats générés.

3. Conservation des données essentielles :

Pour générer les séquences candidates d'ordre K seules les séquences de taille K-1 et 1 sont utiles. Afin d'optimiser de l'espace mémoire, VPSP supprime les listes d'apparitions des séquences fréquentes de taille K-1, lorsque toutes les séquences candidates de longueur K ont été générées.

Extraction des Règles Séquentielles et Calcul de la Confiance

5 Mise en œuvre :

Nous avons vu tous les préliminaires et concepts théoriques nécessaires à la bonne compréhension de VPSP, et ceci pour bien appréhender notre travail et situer où s'effectuera **l'extraction des règles séquentielles et le calcul de la confiance** dans VPSP.

Afin de rendre possible le calcul de la confiance, deux approches s'offraient à nous :

- Calcul en parallèle : consiste à faire le calcul de la confiance en même que la construction de l'arbre des séquences candidates ;
- Calcul en post-traitement : consiste à calculer la confiance une fois la génération de l'arbre des séquences candidates effectuée.

Nous expliquerons le principe de fonctionnement de ces deux approches, et citerons leurs avantages et inconvénients respectifs, et détaillerons le fonctionnement des méthodes implémentées, et des modifications apportées à VPSP pour rendre le calcule de la confiance possible.

5.1 Rappel

5.1.1 Confiance d'une règle séquentielle

La confiance d'une règle séquentielle de type **(X) → (Y)**, est définie comme étant la probabilité d'obtenir (Y) tout en sachant qu'on avait obtenu auparavant (X).

Exemple :

Soit la base de données suivante représentant les achats de deux clients :

Client	Date	Item
C1	17/04/08	A
C1	18/04/08	B
C2	17/04/08	A

La confiance de la règle séquentielle (A) → (B) est définie comment étant le nombre de clients qui ont acheté (B) sachant qui ont acheté (A) auparavant.

$$\text{Confiance ((A) → (B))} = \frac{Nombre\ de\ client\ qui\ ont\ achté\ (\boldsymbol{A})\ puis\ (\boldsymbol{B})}{Nombre\ de\ client\ qui\ ont\ acheté\ (\boldsymbol{A}) seulement} = \frac{\mathbf{1}\ (Client\ C1)}{\mathbf{2}\ (Client\ C1\ et\ C2)}$$

5.1.2 Extraction des Règles séquentielles d'une séquence :

A partir d'une séquence plusieurs règles séquentielles peuvent être extraites, dont les longueurs varient entre 2 et la taille de la séquence globale.

Exemple :

Soit la séquence suivante : **(A B) (C) (D)**

De cette séquence, les règles suivantes peuvent être extraites :

Règle 1 : (A B) (C) → (D)*
Règle 2 : (A B) → (C) (D) *

Règle 3 : (A B) →(D) **
Règle 3 : (A B) → (C) **
Règle 4 : (A) (C) →(D) **
Règle 5 : (A) →(C) (D) **
Règle 6 : (B) (C) →(D) **
Règle 7 : (B) →(C) (D) **
Règle 8 : (A) →(D) **
Règle 8 : (B) →(D) **
Règle 9 : (C) → (D) **
Règle 8 : (A) →(C) **
Règle 8 : (B) →(C) **

Toutes les règles suivi par ** sont générées par VPSP au fur et à mesure que l'arbre des séquences est construit, notre travail consistera à extraire toutes les règles de type *, c'est-à-dire toutes les règles dont la longueur est égale à la longueur de la séquence, et qui inclus tout les itemsets de la séquence.

5.2 Les différentes approches du calcul

5.2.1 Calcul en parallèle

Principe :

Dans cette approche, le calcul de la confiance s'effectue au fur et à mesure que l'arbre des séquences candidates est construit, on opère le calcul de confiance à chaque génération d'un nouveau niveau dans l'arbre, et cela, après avoir calculé le support de la nouvelle séquence construite, le calcul de la confiance est effectué si et seulement si la séquence n'a pas été élaguée.

Le déroulement de l'algorithme peut être schématisé par la figure ci-dessous :

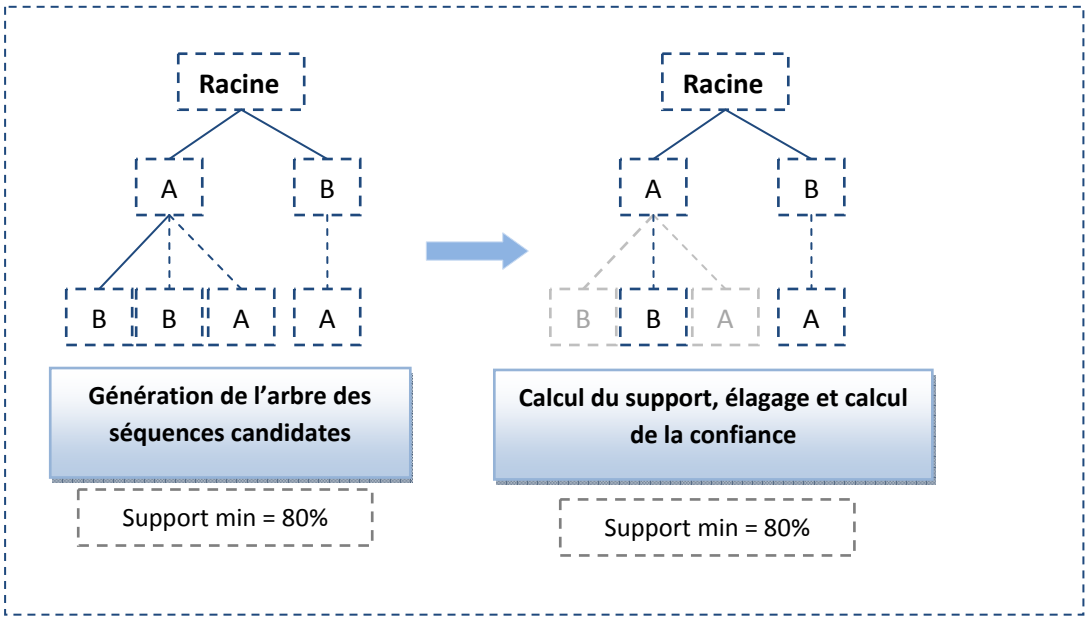

Figure 7 Schéma d'exécution de l'algorithme en calcul parallèle

Avantages et inconvénients

L'utilisation de cette approche offre un certain nombre d'avantages et d'inconvénients, résumés dans le tableau suivant :

Avantages	Inconvénients
Si l'algorithme s'exécute une seule fois (nécessite de connaitre **la bonne valeur du support à utiliser**), cette approche offrira les meilleures performances, car l'arbre préfixé n'est parcouru qu'une seule foi.	Si on ne connaît pas la valeur du support (le plus souvent), le calcul de la confiance va ralentir l'exécution de l'algorithme.
Vu que les confiances sont données au fur et à mesure que l'algorithme s'exécute, on peut modifier la valeur du support afin d'affinée le résultat, sans attendre l'exécution total du l'algorithme.	
	Redéfinition de plusieurs méthodes dans VPSP.

Tableau 8 Avantages et inconvénients de calcul en parallèle

Notre avis

Étant donné que les performances sont un point très important dans VPSP, et sa rapidité d'exécution représente l'une de ces grandes forces, cette approche n'a pas été retenue, car elle détériore sa vitesse d'exécution globale.

5.2.2 Calcul en Post-traitement

Principe :

Dans cette approche, le calcul de la confiance s'effectue une fois que tout l'arbre préfixé est généré, et le calcul du support avec élagage effectué.

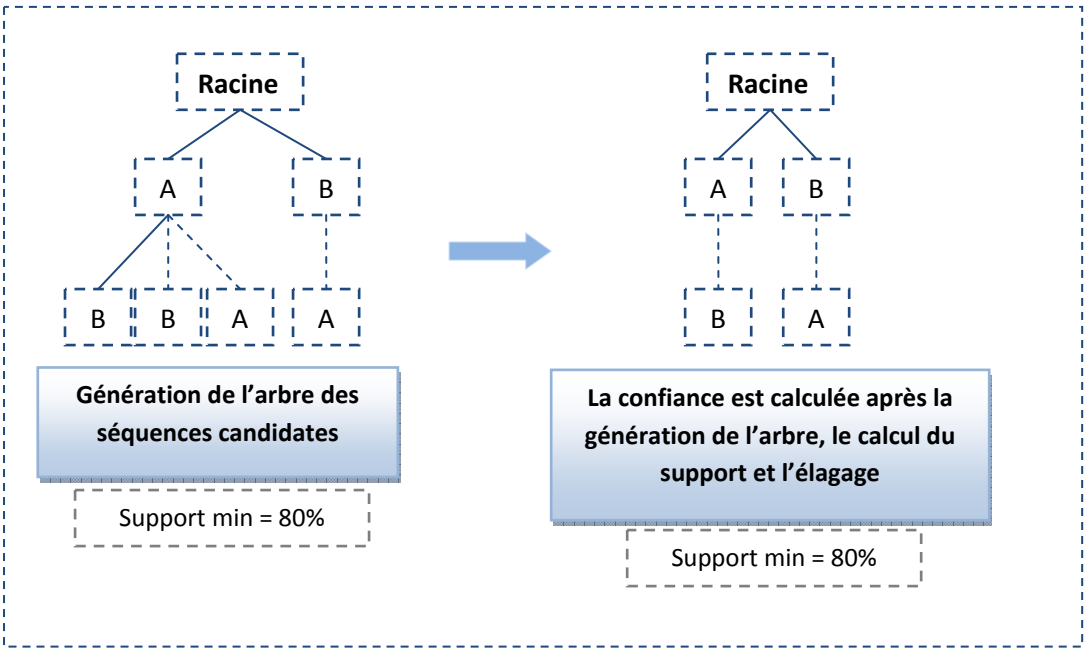

Figure 8 Schéma d'exécution de l'algorithme en post-traitement

Avantages et inconvénients

L'utilisation de cette approche offre un certain nombre d'avantages et d'inconvénients, résumés dans le tableau suivant :

Avantages	Inconvénients
Étant donné que le calcul de la confiance se fait indépendamment du calcul du support, le changement de la valeur du Support Minimum n'induit pas le recalcule de la confiance	Si la confiance est utilisée pour affinée la valeur du support, cela induira une nouvelle exécution de tout l'algorithme
Modification minimes dans VPSP	
Facile à implémenter	

Tableau 9 Avantages et inconvénients de calcul en parallèle

Notre avis

Cette approche a été retenue car comparée à la première, elle offre des meilleures performances, de plus son implémentation n'affecte pas les performances du calcul du support dans VPSP.

5.3 Application :

Afin de rendre le calcul de la confiance possible dans VPSP, nous avons implémenté de nouvelles méthodes dans ce dernier dans ce but. Les deux méthodes les plus importantes rajoutées sont : « **sousSequenceGeneration** » et « **confianceCount** », le fonctionnement détaillé ainsi que des exemples de ses méthodes seront traités dans le paragraphe suivant.

5.3.1 Outils et environnement de développement:

Étant donné que VPSP est

implémenté en Java, l'implémentation de nos méthodes c'est faite aussi sous java, et comme environnement de développement nous avons travaillez sous l'IDE Eclipse 3.3

5.4 Méthodes rajoutées à VPSP

5.4.1 Génération des règles séquentielles de même longueur pour une séquence:

Principe :

La méthode « **sousSequenceGeneration** » génère toutes les règles séquentielles de même longueur pour une séquence donnée.

Génération de toutes les règles séquentielles : Méthode Sous Séquence Génération

Pile Nœuds Niveau1 ← Sommets niveau 1
Tant que Niveau1 a des fils faire
| ***Nœud*** *←* *Sommet Pile Nœud*
| ***NœudSequenceParente*** *←* ***Nœud***
| ***Séquence*** *←Séquence Nœud*
| ***SéquenceParente*** *←Séquence Nœud*
| *Si* ***Nœud*** *est au* ***niveau 1*** *de l'arbre «* ***Pas de règle*** *»*
| *Sinon*
| *Tant que père(**NœudSequenceParente**) != null*
| *Tant que **NœudSequenceParente** à des pères en «* ***Same »***
| ***NœudSequenceParente ← Père(NœudSequenceParente)***
| *Fin Tant que*
| *Si **NœudSequenceParente** est en «* ***Other »***
| ***SéquenceParente ←Séquence(NœudSequenceParente)***
| ***SéquenceReste ← SéquenceReste(NœudSequenceParente,Noeud)***
| *Sauvegarder **SéquenceParente***
| *Sauvegarder **SéquenceReste***
| *Fin Si*
| *Fin Tant que*
| *Fin Si*
| *Empiler dans Pile Nœuds les fils du nœud courant*
| *Fin Tant que*

Exemple :

Soit la séquence suivante : **(A B) (C) (D E) (F G),** l'algorithme « **sous séquence génération** » va gêner toutes les règles séquentielles de longueur 4 et inclus les Items set (A B), (C), (D E) et (F G),

En résultat, on obtient les règles suivantes :

Règle 1 : (A B) (C) (D E) → (F G)
Règle 2 : (A B) (C) → (D E) (F G)
Règle 3 : (A B) → (C) (D E) (F G)

5.4.2 Calcul de la confiance :

Principe :

La fonction du calcul de la confiance est invoquée au niveau de chaque Itemset d'une séquence. Le calcul est effectué pour chaque règle séquentielle générée depuis la séquence courante. Le calcul consiste à faire le **ratio** entre le **support de la séquence globale** avec le **support de la sous séquence générée**.

Calcul de la Confiance : Méthode Confiance Count

Pile Nœuds Niveau1 ← Sommets niveau 1
Tant que Niveau1 a des fils faire
| ***Nœud*** *← Sommet Pile Nœud*
| ***Séquence*** *← Séquence Nœud*
| *Si* ***Nœud*** *est au* ***niveau 1*** *de l'arbre «* ***Pas de confiance*** *»*
| *Sinon*
| *Tant que* ***sousSequenceGeneration***
| *Séquence_sous_régles ← Générer sous règle séquentielle*
| *Calculer la confiance de la règle séquentielle courante :*
| *Confiance ← support (séquentielle courante) / support (Séquence_sous_régles)*
| *Fin Tant que*
| *Fin Si*
| *Empiler dans Pile Nœuds les fils du nœud courant*
| *Fin Tant que*

Exemple :

Soit la séquence suivante : **(A B) (C) (D E) (F G),** l'algorithme « **Confiance Count** » va utiliser le résultat de l'algorithme « **sous séquence génération** » pour effecteur le calcul de la confiance de toutes les sous séquences générer

En résultat, on obtient:

A. Génération de la **Règle 1** avec l'algorithme **sous séquence génération**

(A B) (C) (D E) → (F G)

Puis calcul de la confiance de la **Règle 1**

Confiance (A B) (C) (D E) → (F G) = $\frac{\boldsymbol{Support}\big((\boldsymbol{A\ B})(\boldsymbol{C})(\boldsymbol{D\ E})(\boldsymbol{F\ G})\big)}{\boldsymbol{Support}\big((\boldsymbol{A\ B})(\boldsymbol{C})(\boldsymbol{D\ E})\big)}$

B. Génération de la Règle 2 avec l'algorithme sous séquence génération

(A B) (C) → (D E) (F G)

Puis calcul de la confiance de la Règle 2

$$\text{Confiance (A B) (C) } \rightarrow \text{(D E) (F G)} = \frac{\text{Support}\big((\text{A B})(\text{C})(\text{D E})(\text{F G})\big)}{\text{Support}\big((\text{A B})(\text{C})\big)}$$

C. Génération de la Règle 3 avec l'algorithme sous séquence génération

(A B)→ (C) (D E) (F G)

Puis calcul de la confiance de la Règle 3

$$\text{Confiance (A B) } \rightarrow \text{ (C) (D E) (F G)} = \frac{\text{Support}\big((\text{A B})(\text{C})(\text{D E})(\text{F G})\big)}{\text{Support}\big((\text{A B})\big)}$$

Le calcul de la confiance dans VPSP :

Appliquer dans VPSP, notre algorithme donne le résultat suivant:

Séquence en cours de traitement : [**(1 2) (4) (5 6) (8 9) (9)**]

Le support de la séquence [**(1 2) (4) (5 6) (8 9)**] est : **100.0 %**
Le support de la séquence [**(1 2) (4) (5 6) (8 9) (9)**] est : **100.0 %**
La confiance de la règle [**(1 2) (4) (5 6) (8 9)] → [(9)**] est : **100.0 %**

Le support de la séquence [**(1 2) (4) (5 6)**] est : **100.0 %**
Le support de la séquence [**(1 2) (4) (5 6) (8 9) (9)**] est : **100.0 %**
La confiance de la règle [**(1 2) (4) (5 6)] → [(8 9) (9)**] est : **100.0 %**

Le support de la séquence [**(1 2) (4)**] est : **100.0 %**
Le support de la séquence [**(1 2) (4) (5 6) (8 9) (9)**] est : **100.0 %**
La confiance de la règle [**(1 2) (4)] → [(5 6) (8 9) (9)**] est : **100.0 %**

Le support de la séquence [**(1 2)**] est : **100.0 %**
Le support de la séquence [**(1 2) (4) (5 6) (8 9) (9)**] est : **100.0 %**
La confiance de la règle [**(1 2)**] → [**(4) (5 6) (8 9) (9)**] est : **100.0 %**

Le support de la séquence [**(1 2)**] est : **100.0 %**
Le support de la séquence [**(1 2) (4) (5 6) (8 9) (9)**] est : **100.0 %**
La confiance de la règle [**(1)**] → [**(4) (5 6) (8 9) (9)**] est : **100.0 %**

5.4.3 Optimisation :

Afin d'accélérer le calcul des confiances, et pour ne pas avoir à reparcourir l'ensemble des règles séquentielles générées, nous avons rajouté une méthode qui calcul la confiance en même temps que la génération des sous règles séquentielles est effectuée.

De ce faite on obtient une nette amélioration des performances, comparé à la méthode ou la calcul de la confiance ce fait après génération des règles.

5.5 Schéma UML

Schéma conceptuel de la solution proposée

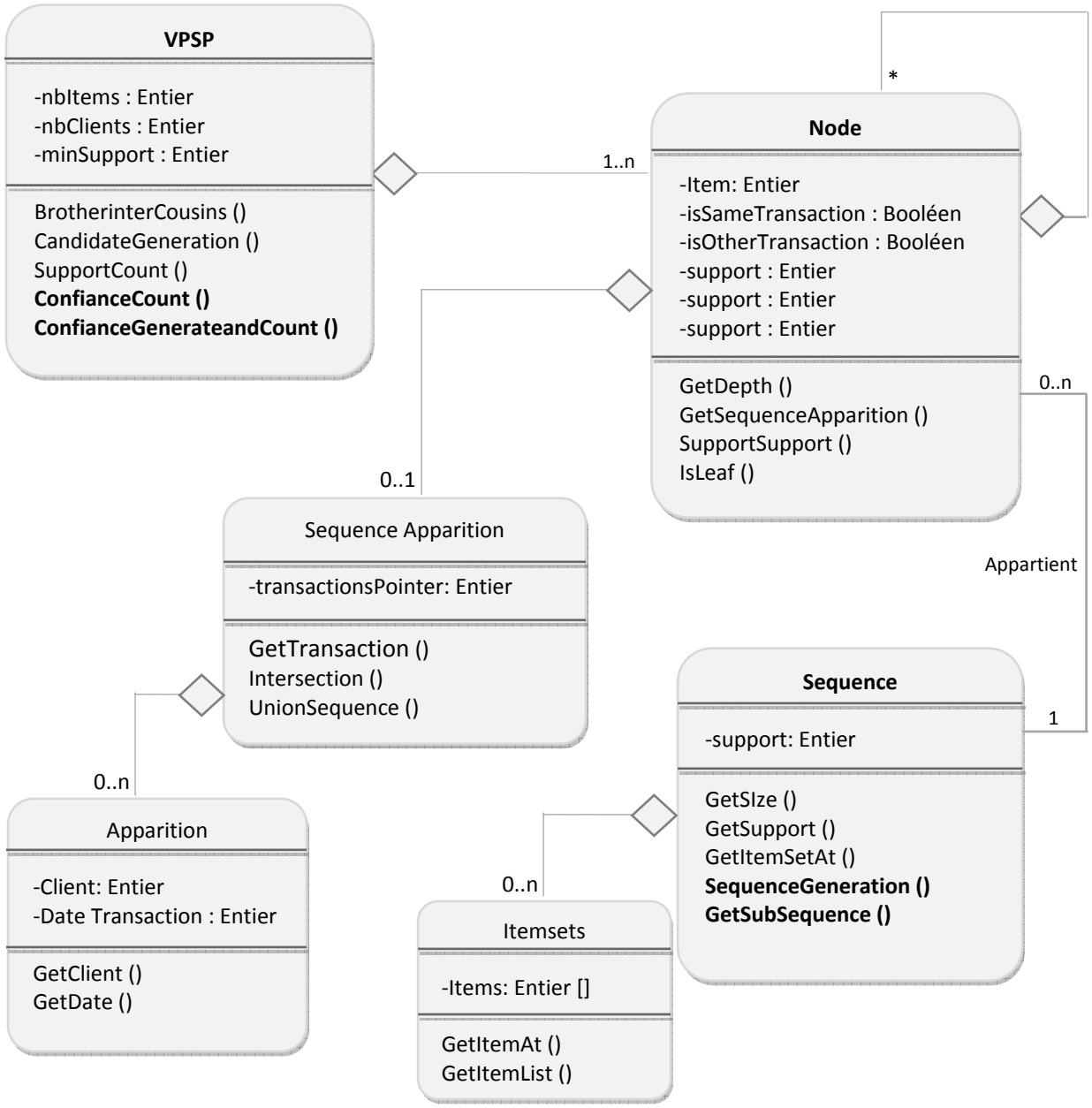

Figure 9 Schéma UML

5.6 Résultats obtenus :

Notre implémentation répond aux objectifs attendus par l'équipe TATOO.
En effet, on a réussit à bien cerner et comprendre le fonctionnement de l'algorithme VPSP, pour pouvoir par la suite étendre VPSP et inclure dans ce dernier l'extraction des règles séquentielles et le calcul de la confiance de toutes les règles générées.

6 Conclusion

6.1 Bilan

Pour conclure, ce TER nous a été grandement bénéfique et représente une expérience très enrichissante, il nous a permis de découvrir des domaines complètement nouveaux telle la fouille de données et le monde de la recherche.

Il nous a aussi permis de travailler en groupe et ainsi d'améliorer nos capacités de travail en équipe.

Lors de ce TER, nous avons essayé de répondre au mieux aux objectifs qui nous ont été assignés au tout début à savoir :

- Compréhension du fonctionnement de l'algorithme VPSP ;
- Réalisation de l'extraction des règles séquentielles ;
- Calcul de la confiance des règles séquentielles générées.

Nous espérons que les méthodes que nous avons implémentées dans VPSP, seront exploitées par l'équipe TATOO et leurs permettront de répondre au mieux à leurs préoccupations.

Cependant, nous avons rencontrés à certaines difficultés notamment :

- Un manque de documentation dû à la nouveauté du domaine de recherche ;
- Des difficultés d'assimilation de certains concepts théoriques.

D'autre part, cette expérience fut très enrichissante, dans le sens où elle nous a amenés à comprendre les diverses problématiques liées aux motifs séquentiels et à l'extraction des règles séquentielles.

6.2 Perspectives

Les comportements trouvés dans la vie réelle sont rarement binaires « blanc ou noir », En effet la plupart des données réelles intéressantes dans le contexte de données séquentielles sont numériques (quantitatives).

La modélisation de ces comportements intermédiaires est alors assez facilement représentable en utilisant la théorie des sous-ensembles flous.
Il serrait donc intéressant d'introduire la notion de *flou* à ce problème d'extraction de règles séquentielles, nous pourrons alors avoir des informations de ce type :

(Biscottes, peu) -> (Nutella, beaucoup) : L'achat comporte le produit Nutella en grande quantité s'il comporte le produit biscottes en petites quantités.

7 Table des figures :

8 Tableaux

9 Bibliographie

ABDELALI Mouad, O. H. (2003). *Création de régles séquentielles.*

AGRAWAL R., S. R. (March 1995). *"Mining Sequential Patterns", Procee-dings of the 11th International Conference on Data Engineering (IC-DE'95).* Tapei, Taiwan.

Aurélien Serra, D. K. (2006). *Intégration des motifs séquentiels dans Weka.* Montpellier.

FIOT C. (2004). *Traitement des données manquantes en fouille de données.* Montpellier.

ZAKI M. (2001). *"SPADE : an efficient algorithm for Mining Frequent Sequences",Machine Learning, Vol. 42,2001, p31-60.* Kluwer Academic Publishers.