Université d'Angers

M2 Intelligence Décisionnelle

Recherche de motifs fréquents par algorithmes évolutionnaires

Auteur :
Ugo Rayer

Encadrants :
Benoit Da Mota
Béatrice Duval
David Lesaint

3 avril 2017

Remerciements

Avant toute chose, je tiens à remercier l'ensemble des personnes qui ont participé, de près ou de loin, à la réalisation de ce stage et à l'écriture de ce rapport.

Des remerciements spéciaux sont adressés d'une part au Laboratoire d'Etude et de Recherche en Informatique d'Angers pour son accueil et d'autre part au projet GRIOTE de la région Pays de la Loire qui a financé cette étude. Ensuite, je tiens à remercier chaleureusement Madame Duval et Messieurs Da Mota et Lesaint pour la qualité de leur encadrement et les différentes remarques qu'ils m'ont prodiguées pendant ces quelques mois.

Enfin, un grand merci à Josépha pour ses précieux conseils d'écriture malgré l'incompréhension générale des sujets abordés.

Table des matières

1 Introduction **4**

2 Le problème du calcul motifs fréquents **6**
2.1 Définition du problème 6
2.1.1 Formalisation 6
2.1.2 Exemple 7
2.1.3 Complexité et Propriétés 7
2.2 Problème des itemsets clos maximaux 8

3 Etat de l'art **10**
3.1 Méthodes exactes 10
3.1.1 A Priori 10
3.1.2 Eclat 12
3.1.3 FP-Growth 13
3.1.4 Algorithmes pour itemsets clos 17

Bibliographie **19**

Chapitre 1

Introduction

Le calcul des motifs fréquents est une notion essentielle dans de nombreux domaines liés à la découverte et l'extraction de connaissances. Initiallement introduit par Agrawal et al. dans [1], ces motifs étaient alors utilisés pour l'établissement de règles d'associations visant à caractériser les habitudes d'achats de clients d'un supermarché. Par exemple, une règle de la forme "Pain & Beurre $\Rightarrow$ Jambon (75%)" signifie que 3 clients sur 4 achetant du pain et du beurre achètent également du jambon. Le calcul de telles règles se fait en deux étapes, dont la principale (i.e. disposant de la plus grosse complexité calculatoire) correspond au calcul des motifs fréquents. Depuis son introduction, le problème du calcul des motifs fréquents a été très largement étudié et appliqué à de nombreux domaines comme en bio-informatique, en cybersécurité et bien évidemment en marketing.

L'avènement de l'ère du Big Data a fait rentrer le problème de calcul des motifs fréquents dans une nouvelle dimension. En effet, le volume de données produites dans tous les domaines a cru de manière vertigineuse ces dernières années, rendant l'extraction de connaissances d'autant plus nécessaire et délicate. De fait, la problématique du passage à l'échelle des méthodes exactes est devenue primordiale. Bien que divers efforts en ce sens aient été faits au travers de nombreuses publications, ils se concentrent généralement sur l'optimisation et la parallélisation des méthodes existantes.

Les algorithmes évolutionnaires font partie des techniques de résolution de problèmes combinatoires appelées méta-heuristiques. Les méta-heuristiques regroupent un ensemble de méthodes approchées à la résolution de problème combinatoire. Elles sont naturellement envisagée lorsque la complexité du problème étudié ne permet pas l'usage de méthodes exactes (aussi bien en temps qu'en espace). Le principe des algorithmes évolutionnaires est de manipuler un ensemble d'individus représentant chacun une solution au problème étudié. A chaque itération, certains individus sont modifiés via des opérateurs de croisement et de mutation. Chaque individu est évalué au regard d'une fonction à optimiser dépendant du problème étudié.

Ainsi, nous formaliserons le problème de calcul de motifs fréquents dans la section suivante avant d'effectuer un état de l'art des méthodes existantes en

section 3. Le chapitre 4 sera dédié à la présentation de notre méthode dont nous présenterons les résultats en section 5. Le dernier chapitre sera consacré à la conclusion de cette étude et à une ouverture vers de futurs travaux.

Chapitre 2

Le problème du calcul motifs fréquents

2.1 Définition du problème

La définition la plus courante du problème de calcul des motifs fréquents se fait par la théorie des ensembles. Nous verrons cependant dans le chapitre suivant qu'il peut également être décrit par la théorie des graphes. Nous présenterons dans un premier temps un cadre formel nécessaire à la définition du problème que nous illustrerons ensuite. Enfin, nous introduirons quelques propriétés dérivés de la définition du problème.

2.1.1 Formalisation

Soit $\mathcal{I}$ un ensemble de *symboles* appelées **items**. Quelque soit $I \subseteq \mathcal{I}$, I est un *motif* appelé **itemset**.

Soit $\mathcal{T} = \{ t_1, \ldots, t_n \}$ un ensemble de **transactions**. Chaque élément t_i est un couple $< tid, I >$ où tid est l'identifiant de la transaction et $I \subseteq \mathcal{I}$. $\mathcal{T}$ est communément appelé **base de transactions**.

Pour tout itemset $I \subseteq \mathcal{I}$ la **couverture** de I par $\mathcal{T}$ est définie par :

$$\mathbf{cover}_{\mathcal{T}}(I) = \{ t \in \mathcal{T} |\ I \subseteq t \}$$

La cardinalité de la couverture d'un itemset I par $\mathcal{T}$

$$\mathbf{sup}_{\mathcal{T}}(I) = |\ \mathbf{cover}_{\mathcal{T}}(I)\ |$$

est appelée **support** de I. Etant donné un support minimal $minsup$ l'ensemble des **itemsets** (i.e. motifs) **fréquents** est défini par :

$$\mathbf{F}_{\mathcal{T},minsup} = \{ I \subseteq \mathcal{I} |\ sup_{\mathcal{T}}(I) \geq minsup \}$$

Le problème du **calcul des itemsets fréquents** (**FIM** - *Frequent Itemsets Mining*) est, étant donné une base de transactions $\mathcal{T}$ et un support minimal *minsup* de calculer l'ensemble F des itemsets fréquents. Comme F. Boden le remarque dans [2], bien qu'historiquement définie comme une valeur relative et donc asujettie à un support minimal défini dans l'intervalle [0,1], le support est de nos jours mesuré de manière absolue. Si nécessaire, nous y ferons référence sous la notion de fréquence :

$$\mathbf{Freq}_{\mathcal{T}}(I) = \frac{|\ \mathbf{cover}_{\mathcal{T}}(I)|}{|\mathcal{T}|}$$

avec *minfreq* simplement définie par $\frac{minsup}{|\mathcal{T}|}$.

2.1.2 Exemple

Situons nous dans le contexte de la définition historique de problème et considérons la base de transactions suivante (que nous conserverons tout au long de cet article). Le tableau 1 décrit chaque transaction par : un identifiant, une liste d'objets achetés et la liste des items fréquents vis à vis d'un support minimal de 3.

ID transaction	Objets achetés	Items Fréquents Ordonnés
100	f, c, a, d, g, i m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

TABLE 2.1 – Base de transactions exemple

Une fois calculé, l'ensemble des itemsets fréquents vis à vis de ce jeu de données $\mathcal{T}$ est l'ensemble $\mathbf{F}_{\mathcal{T},3}$ = { (f :4), (c :4), (a :3), (b :3), (m :3), (p :3), (fc :3), (fa :3), (fm :3), (cm :3), (cp :3), (ca :3), (am :3), (fca :3), (fcm :3), (fam :3), (cam :3), (fcam :3) }.

2.1.3 Complexité et Propriétés

La complexité du problème vient d'une part du nombre d'itemsets à considérer en fonction du nombre d'objets et d'autre part de nombre de transactions dans la base. En effet, soit n objets fréquents dans la base, il y a alors 2^n itemsets possibles. D'autre part, le calcul du support d'un itemset se fait au travers de l'ensemble des transactions. L'efficacité d'une méthode à résoudre ce problème se fera donc par sa capacité à explorer intelligement l'espace des 2^n itemsets et par son efficacité à calculer le support d'un itemset vis à vis de $|\mathcal{T}|$.
Différents théorèmes et propriétés issus de l'étude de ce problème sont utilisés dans les méthodes proposées pour le résoudre. Nous présentons les propriétés liées à la monotonie du support d'un ensemble et renvoyons le lecteur vers [2] et [3] pour plus de détails.

Monotonie du support. Soit une base de transactions $\mathcal{T} sur \mathcal{I}$ et soient X, $Y \subseteq \mathcal{I}$ deux itemsets. Alors,

$$X \subseteq Y \Rightarrow \mathbf{sup}_{\mathcal{T}}(Y) \leq \mathbf{sup}_{\mathcal{T}}(X)$$

Cette propriété nous permet de dire que si un k-itemset X (i.e. un itemset comprenant k objets) est fréquent, alors l'ensemble Y des k-1-itemsets $\subset X$ est fréquent. Par exemple, (fca :3) est fréquent, de même que (fc :3), (fa :3) et (ca :3). Nous pouvons de manière duale dire que si un k- itemset X est non-fréquent, alors aucun des k+1-itemset Y tel que $X \subset Y$ n'est fréquent. Ces deux observations sont à la base des différents sens de parcours de l'espace de recherche des 2^n itemsets possibles dans la plupart des algorithmes.

2.2 Problème des itemsets clos maximaux

Afin de réduire l'espace de recherche, il a été proposé de contraindre la recherche à l'ensemble des itemsets clos. Un k-itemset$X$ fréquent est clos si $\mathbf{sup}_{\mathcal{T}}(X) > \mathbf{sup}_{\mathcal{T}}(Y)\ \forall\ Y$ tel que $X \subset Y$. Un itemset clos est maximal si aucun de ses surensembles n'est fréquent. Ainsi dans notre exemple, l'ensemble des itemsets clos est $\mathbf{C}_{\mathcal{T},3}$ = { (f :4), (c :4), (b :3), (cp :3), (fcam :3) } et l'ensemble des itemsets clos maximaux est $\mathbf{MC}_{\mathcal{T},3} = \mathbf{C}_{\mathcal{T},3}-$ { (f :4), (c :4) }. La figure suivante représente le treillis complet des différents itemsets possibles sur les objets fréquents. Y figurent d'une part les itemsets fréquents en vert, d'autre part les itemsets clos en jaune et enfin les itemsets clos maximaux en rouge. Pour plus de lisibilité, les itemsets non-fréquents ne figurent pas dans le treillis.

Zaki et al. et Pas. et al. prouvent dans [4] et [5] que l'intégralité des itemsets fréquents peut être dérivée de l'ensemble des itemsets clos maximaux. Ils proposent alors d'adapter les méthodes existantes pour le calcul des itemsets clos maximaux.

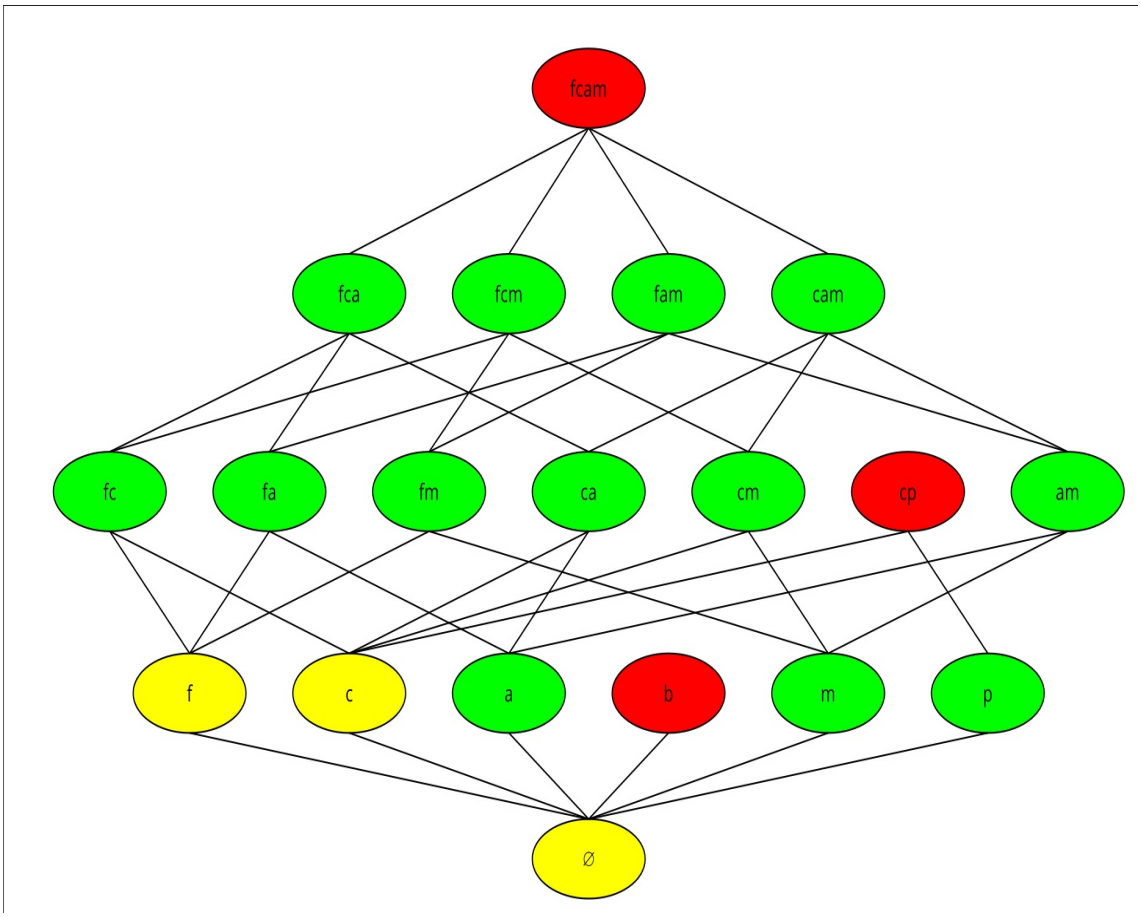


FIGURE 2.1 – Treillis des itemsets fréquents, clos et maximaux de notre exemple

Chapitre 3

Etat de l'art

Dans ce chapitre, nous nous attacherons à présenter les principales méthodes de la littérature pour le calcul des itemsets fréquents. La première section sera dédiée à la présentation des méthodes exactes. Nous y aborderons les trois principaux algorithmes que sont A Priori [6], Eclat [7] et FP-Growth [8]. D'autres méthodes dédiées au calcul des itemsets clos maximaux seront également abordées. Enfin nous introduirons quelques méthodes présentant un intérêt pour la problématique du passage à l'échelle. La section 2 sera consacrée aux méthodes métaheuristiques de la littérature. [TO COMPLETE]

3.1 Méthodes exactes

3.1.1 A Priori

Un an après avoir introduit le problème des règles d'associations (et par conséquent le problème de calcul des itemsets fréquents) dans [1], les auteurs ont proposé l'algorithme A Priori [6] restant la référence littéraire du domaine. Son principe général est la génération de k+1-itemsets candidats à partir des k-itemsets fréquents précédemment calculés. L'algorithme 1 présente la phase de calcul des itemsets fréquents par A Priori. Nous ne présenterons pas l'algorithme calculant les règles d'associations dans cette étude.

Algorithm 1: APriori Itemset Mining

```
   Input  : 𝒯, minsup
   Output: F_{𝒯,minsup}
 1 begin
 2   C_1 = { { i } | ∈ ℐ }
 3   k = 1
 4   while C_k ≠ {} do
 5     // Calcul du support de chaque itemset candidat
 6     forall t = ( tid, I ) ∈ 𝒯 do
 7       forall itemset candidat X ∈ C_k do
 8         if X ⊆ I then
 9           X.support ++
10         end
11       end
12     end
13     // Extraction de l'ensemble des itemsets fréquents
14     F_k = {X | X.support ≥ minsup}
15     // Génération des candidats de taille k+1
16     forall X, Y ∈ F_k | X[i] = Y[i] ∀i ∈ [1, k − 1]& X[k] < Y[k] do
17       I = X ∪ { Y[k] }
18       if ∀J ⊂ I, |J| = k et J ∈ F_k then
19         C_{k+1} = C_{k+1} ∪ I
20       end
21     end
22     k ++
23   end
24 end
```

Comme nous pouvons le constater, l'algorithme considère dans un premier temps l'ensemble des items comme candidats avant d'effectuer un premier scan de la base $\mathcal{T}$. Les 1-itemsets retenus sont utilisés pour générer des candidats de taille 2. L'algorithme poursuit cette routine jusqu'à ce qu'aucun candidat ne soit généré.

Différentes implémentations efficaces ont été proposées pour améliorer la rapidité et l'espace mémoire nécessaire à l'exécution de l'algorithme Apriori. En général, les optimisations sont basées sur l'utilisation de structures spécifiques pour la représentation de la base de transactions et des itemsets candidats. Par exemple, [9] utilise un arbre préfixe pour stocker les itemsets candidats. D'autres structures incluant des vecteurs, des arbres de hachage et des structures hybrides existent également.

Enfin, nous pouvons également mentionner l'existence d'implémentations d'Apriori utilisant le paradygme de la programmation parallèle. Par exemple [10] présente une méthode parallèle partitionnant la base de transactions en sous-bases chacune assignée à un processeur différents. Une fois le calcul des k-itemsets localement fréquents, les résultats sont transmis à un noeud maître effectuant une centralisation des résultats. Notons également

l'emploi de vecteurs pour stocker les 1 et 2-itemsets puis un arbre préfixe ensuite.

3.1.2 Eclat

A la différence d'Apriori, qui effectue un parcours du treillis des candidats en largeur, Eclat est le premier algorithme à utiliser une représentation verticale de la base de transactions pour effectue une recherche en profondeur. Ainsi, Zaki et al. ont proposé cet algorithme dans [7] en même temps que divers autres algorithmes dont certains basés sur une relation de pseudo-equivalence de cliques maximales.

Le tableau suivant illustre la base de transactions utilisées dans l'exemple précédent représentée verticalement. Ainsi, nous ne traitons plus une liste d'items par transaction, mais une liste d'identifiants de transaction par item. Une telle liste est appelée *tid-list*. De plus, Eclat utilise la notion de classe d'équivalence pour distribuer le calcul des itemsets candidats sur différents processeurs. Par exemple, les itemsets ABC et ABD appartiennent à la même classe d'équivalence $\mathcal{E}_{AB}$.

Item	*tid-lists*
f	100 200 300 500
c	100 200 400 500
a	100 200 500
b	200 300 400
m	100 200 500
p	100 400 500

TABLE 3.1 – Base de transactions verticale

L'algorithme 2 présente la structure générale de l'algorithme Eclat. Le principe général est une recherche en profondeur d'abord calculant pour chaque item fréquent l'ensemble des itemsets le contenant. Un *k+1*-itemset candidat est généré à partir de deux k-itemsets partageant un même préfixe de taille *k-1* et son support est calculé par l'intersection des *tid-lists* des deux k-itemsets. Dans le but de réduire le nombre de candidats générés, il est important de noter

l'ordonnancement des items par support croissant.

Algorithm 2: Eclat

Input : $\mathcal{T}$, *minsup*, $I \subseteq \mathcal{I}$ (préfixe, initiallement $\emptyset$)
Output: $\mathbf{F}[I](\mathcal{T}, minsup)$= ensemble des k-itemsets fréquents partageant le même préfixe I

```
1  begin
2      F[I] = ∅
3      forall i ∈ T do
4          F[I] = F[I] ∪ {I ∪ {i}}
5          // Création des candidats de taille |I| + 1
6          D^i = ∅
7          forall j ∈ T|j > i do
8              C = tid-list[{i}] ∩ tid-list[{j}]
9              if |C| > minsup then
10                 D^i = D^i ∪ {(j, C)}
11             end
12         end
13         // Récursion en profondeur sur i
14         Calculer F[I ∪ {i}](D^i, minsup )
15         F[I] = F[I] ∪ F[I ∪ {i}]
16     end
17 end
```

Sur notre exemple, l'algorithme Eclat extrairait dans un premier temps les itemsets contenant l'item p, puis ceux contenant m mais non p etc. La propriété de monotonie du support n'est pas totalement employée puisqu'un itemset n'est pas candidat à condition que l'intégralité des ses sous-ensembles soient fréquents mais uniquement 2. Diverses améliorations ont été proposées dont une par les auteurs originels d'Eclat introduisant la notion de *diffset* utilisée pour le calcul du support d'un itemset candidat. Références et détails sont disponibles dans [3].

3.1.3 FP-Growth

Le dernier des 3 algorithmes les plus répandus et étudiés de la littérature est l'algorithme FP-Growth. Proposé par Han et al. dans [8], il est également basé sur un parcours en profondeur item après item. Toutefois, il utilise une structure de données particulière, FP-Tree, permettant de condenser la base de transactions et d'en extraire diverses informations supplémentaires (distribution des items, bases conditionnelles de motifs...). Dans un premier temps nous introduirons la structure de FP-Tree en construisant celui associé à notre base de transactions exemple.

Structure FP-Tree

Un FP-Tree est un arbre de préfixes construit à partir d'une base de transactions $\mathcal{T}$ et un support minimum *minsup*. La construction d'un tel arbre se fait via deux scans de la base $\mathcal{T}$. Le premier permet de calculer $\mathcal{L}_{FI} = \{ (i, \mathrm{C}) | i \in \mathcal{I}$ et $\mathrm{C} = sup_{\mathcal{T}}(\{i\}) \}$.
Cette liste est ensuite ordonnée par support décroissant. Sur notre exemple, le

Item	f	c	a	b	m	p
Support	4	4	3	3	3	3
Item Pointer	null	null	null	null	null	null

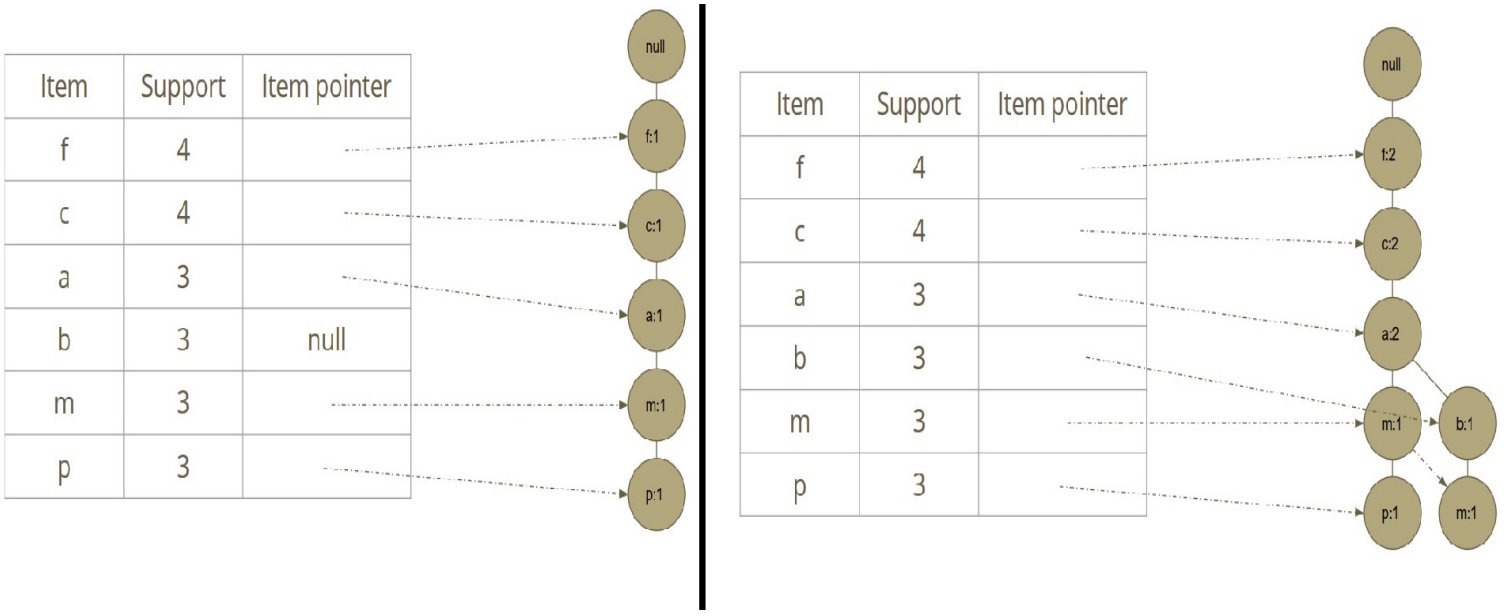
FIGURE 3.1 – FP-Tree T après initialisation

premier scan de $\mathcal{T}$ nous donne la liste suivante :

$$\mathcal{L}_{FI} = \{ (f, 4), (c, 4), (a, 3), (b,3), (m, 3), (p, 3) \}$$

La seconde étape va nous permettre via cette liste de construire l'arbre de préfixe T de la manière suivante. Premièrement, un noeud null correspondant à la racine de l'arbre, de même qu'une header-table comprenant pour chaque item son label, le support de l'1-itemset qu'il induit et un pointeur vers le premier noeud de l'arbre portant le même label (initialisé à null). Un récapitulatif de cette première étape est visuellement présenté dans la figure suivante.

Ensuite, chaque transaction t de la base $\mathcal{T}$ est de nouveau scannée. Soit $[p|P]$ la liste des items $i \in t$ où p est le premier item et P le reste de la liste. Un noeud de l'arbre possède les informations suivantes : un label, un compteur, un pointeur vers son père et une liste de pointeurs vers ses noeuds fils. La fonction d'insertion de $[p|P]$ dans T est alors appelée et fonctionne comme suit. Si T fonctionne un fils N tel que le label de N soit le même que le label de p, alors on incrémente simplement le compteur de N de 1. Sinon, nous créons N avec un compteur initialisé à 1 et ayant T pour parent. On relie ensuite le dernier pointeur de la header-table correspondant au label de p à ce nouveau noeud N. Pour terminer, si P (le reste de la liste des items fréquents dans la transaction t) est non vide, on appelle récursivement cette fonction avec P et N en paramètres.
Le processus est explicité dans la figure suivante présentant l'évolution du FP-Tree après lecture des deux premières transactions et après l'intégralité de la base.



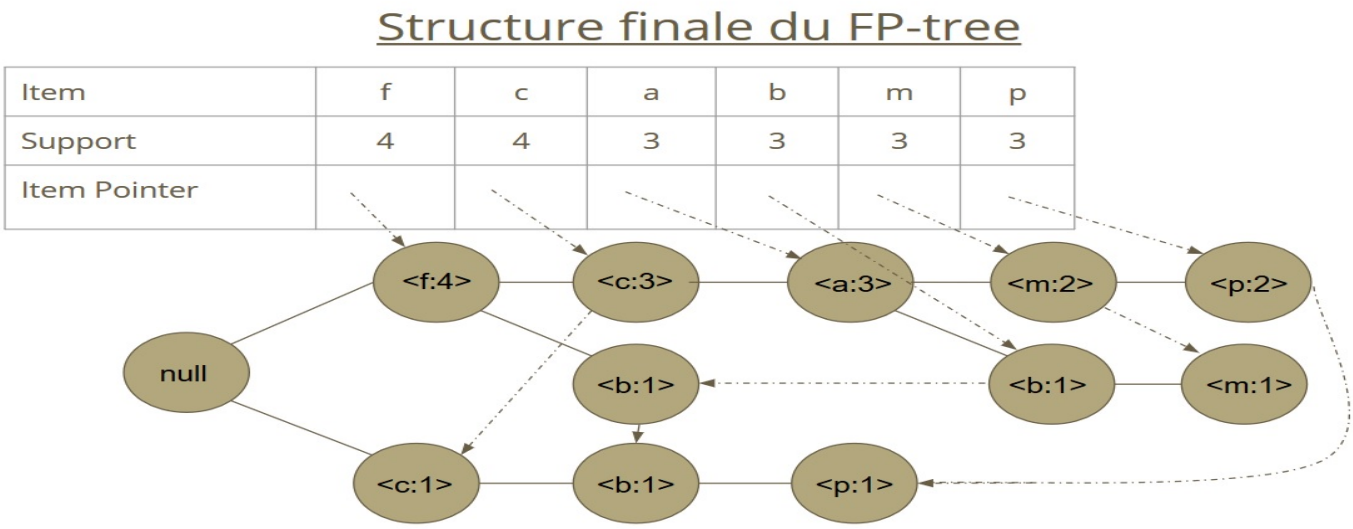


FIGURE 3.2 – FP-Tree T après t1 et t2 et FP-Tree final

Extraction des itemsets fréquents

Une fois le FP-Tree obtenu à partir de $\mathcal{T}$, l'extraction des itemsets fréquents se fait de manière récursive en profondeur d'abord sur chacun des 1-itemsets contenus dans la header-table. Le principe d'augmentation de motif décrit est basé sur le calcul de base conditionelle de motifs depuis le FP-Tree. Le processus récursif se fait via le calcul de nouvelles bases conditionnelles sur les motifs venant d'être augmenté via la création de FP-Tree conditionnel sur ces bases.

La base conditionnelle d'un itemset $I_1...I_k$ correspond à l'ensemble des chemins du FP-Tree où l'itemset apparaît. L'algorithme traite les 1-itemsets de la header table du moins fréquent au plus fréquent et calcule pour chacun l'ensemble des itemsets fréquents le contenant. Le calcul de cette base conditionnelle se fait grâce à la propriété suivante :

Pour chaque item fréquent a_i, l'ensemble des motifs contenant uniquement des items fréquents et a_i peut être obtenu en suivant les noeuds-liens de a_i en commençant par le noeud de la header-table.

Sur notre exemple, l'extraction des itemsets fréquents commence avec l'item p. En suivant les liens de la header-table pour l'item p, nous pouvons extraire deux chemins le contenant (f :4, c :3, a :3, m :2, p :2) et (c :1, b :1, p :1). N'apparaissant respectivement que deux et une fois avec p, nous conservons donc les préfixes (fcam :2) et (cb :1) qui forment la base conditionnelle de p. La construction d'un FP-Tree sur cette base conditionnelle, appelé FP-Tree conditionnel de p et dénoté FP-Tree$|_p$, mène à la création d'un seul noeud (c :3) et par conséquent seul l'itemset (cp :3) est dérivé. Ne pouvant construire de base conditionnelle sur FP-Tree$|_p$, la recherche associée à l'item p est terminée et les deux itemsets fréquents le contenant sont (cp :3) et (p :3).
L'extraction des itemsets où figurent l'item m est décrit dans la figure suivante. La base conditionnelle de$m$ mène à la construction d'un FP-Tree conditionnelle devant être exploité, enclenchant ainsi la récursion en profondeur. Il est toutefois important de noter que lors de la construction d'un tel arbre, les items doivent être ordonnés par fréquence décroissante du nombre de noeuds dans lequel ils figurent.
La génération des itemsets fréquents via la base conditionnelle d'un sous-motifs se fait grâce à la propriété suivante :

Différentes optimisations sont proposées pour accélérer le processus d'extraction. La principale et plus intéressante est liée à la topologie du FP-Tree. En effet, si celui-ci contient un seul chemin, alors l'ensemble des itemsets fréquents peut être obtenu en énumérant l'ensemble des combinaisons d'items le long de ce chemin avec pour support celui de l'item y appartenant ayant le plus faible. Sur notre exemple, nous aurions par exemple pu énumérer l'ensemble des combinaisons du FP-Tree$|_m$ pour en extraire l'intégralité des itemsets le contenant. Cette propriété a été étendue pour exploiter les arbres contenant un préfixe unique en considérant ce préfixe comme un chemin unique (énumération des combinaisons) et la partie multiple comme un autre FP-Tree en ajoutant une racine au niveau de l'embranchement. Ainsi, l'extraction complète se

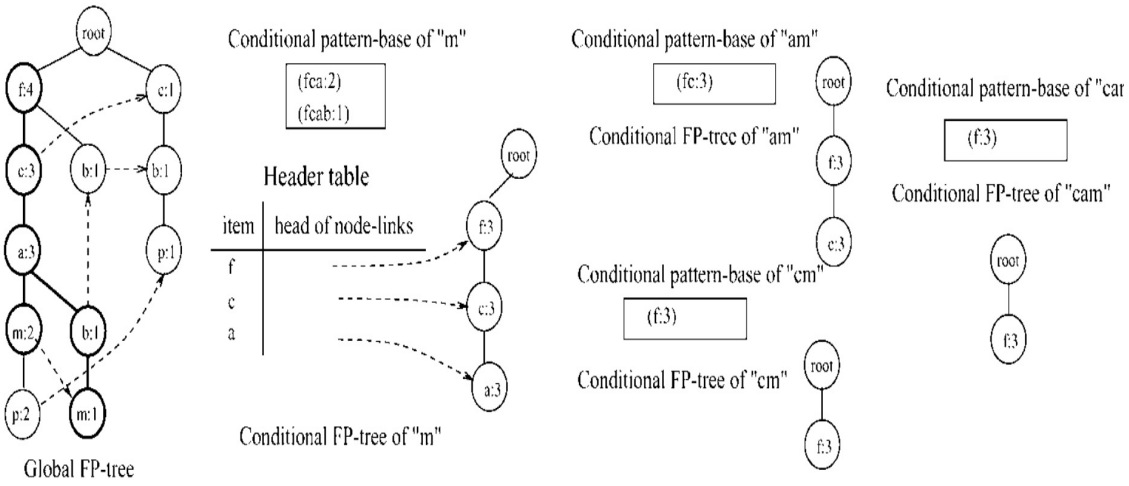


FIGURE 3.3 – Extraction des itemsets contenant m

fait par l'union des itemsets produit par l'énumération avec ceux produits par le processus récursif et enfin avec les itemsets obtenus par combinaisons d'éléments appartenant à l'un et l'autre des ensembles.

Bien qu'étant les trois références en matière de méthodes exactes pour l'extraction des itemsets fréquents, ces algorithmes ont malheureusement de nombreuses difficultés à être comparés. En effet, différentes optimisations et implémentations ayant été proposées, de nombreux articles de la littérature présentent des conclusions différentes sur l'efficacité des méthodes. Nous pouvons toutefois considérer que chaque algorithme possède des performances variables principalement selon les caractéristiques du jeu de données étudié et de la qualité de l'implémentation qui en est faite. Nous allons maintenant présenter rapidement quelques algorithmes pour la recherche des itemsets clos avant d'introduire les méta-heuristiques utilisées pour résoudre ce problème.

3.1.4 Algorithmes pour itemsets clos

De la même manière que trois algorithmes pour les itemsets fréquents ont été présentés, trois pour l'extraction des itemsets clos le seront également. Toutefois, nous serons plus succint et renverrons le lecteur au article correspondant quand nécessaire. Ainsi, nous présenterons rapidement les algorithmes A-Close ([5]), Charm ([4])et Closet, utilisant également un FP-Tree pour fonctionner ([11]).

A-Close

Proposé par Pasquier et al. en 1999 dans [5], l'algorithme A-Close est l'adaptation d'A Priori pour l'extraction des itemsets clos. Il est composé de deux phases. La première effectue une recherche bottom-up dans le but d'identifier

l'ensemble des *générateurs*, i.e. les plus petits itemsets qui déterminent un itemset clos via l'opérateur de clôture défini sur la correspondance de Galois entre itemsets et transactions. Le calcul des *générateurs* se fait via une légère modification d'A Priori. En effet, à chaque fois qu'un ensemble de candidats est généré, A-Close calcule leur support et élimine l'ensemble des non-fréquents. Pour chaque itemset candidat restant, leur support est comparé à celui de ses sous-ensembles calculé au niveau précédent. Si le support d'un candidat égale celui d'un de ses sous-ensembles, cet itemset ne peut pas être un *générateur* et est donc supprimé. Cette phase continue jusqu'à ce qu'aucun *générateur* ne soit produit.

La seconde phase de l'algorithme A-Close consiste, une fois l'ensemble des *générateurs* obtenus, à calculer la clôture de chacun d'entre eux et à éliminer les redondances. Le calcul de la clôture d'un itemset se fait par l'intersection de l'ensemble des transactions dans lesquelles cet itemset apparaît. Ce calcul peut se faire via un seul scan du jeu de données sous réserve que l'intégralité des *générateurs* tient en mémoire centrale.

Charm

De la même manière qu'A-Close, Charm définit une correspondance de Galois entre les ordres partiels induits par les itemsets et les transactions. Toutefois, Charm explore simultanément l'espace des itemsets et celui des tidsets en effectuant des opérations d'union sur les premiers et d'intersection sur les seconds. L'algorithme construit l'arbre des itemsets clos fréquents via les quatre propriétés suivantes tout en élaguant dès que possible les itemsets non fréquents et non clos.

Propriété 1
Si $\mathbf{cover}_{\mathcal{T}}(I_1) = \mathbf{cover}_{\mathcal{T}}(I_2)$ alors $\mathbf{cover}_{\mathcal{T}}(I_1 \cup I_2) = \mathbf{cover}_{\mathcal{T}}(I_1) \cap \mathbf{cover}_{\mathcal{T}}(I_2) = \mathbf{cover}_{\mathcal{T}}(I_1) = \mathbf{cover}_{\mathcal{T}}(I_2)$. Dans ce cas, nous pouvons remplacer toutes les occurences de I_1 par $I_1 \cup I_2$ et retirer I_2 des considérations futures.

Propriété 2
Si $\mathbf{cover}_{\mathcal{T}}(I_1) \subset \mathbf{cover}_{\mathcal{T}}(I_2)$ alors $\mathbf{cover}_{\mathcal{T}}(I_1 \cup I_2) = \mathbf{cover}_{\mathcal{T}}(I_1) \cap \mathbf{cover}_{\mathcal{T}}(I_2) = \mathbf{cover}_{\mathcal{T}}(I_1) \neq \mathbf{cover}_{\mathcal{T}}(I_2)$. Dans ce cas, nous pouvons remplacer toutes les occurentes de I_1 par $I_1 \cup I_2$ mais devons conserver I_2 dans les considérations futures puisque $\mathbf{cover}_{\mathcal{T}}(I_1) \neq \mathbf{cover}_{\mathcal{T}}(I_2)$.

Propriété 3
Si $\mathbf{cover}_{\mathcal{T}}(I_1) \supset \mathbf{cover}_{\mathcal{T}}(I_2)$ alors $\mathbf{cover}_{\mathcal{T}}(I_1 \cup I_2) = \mathbf{cover}_{\mathcal{T}}(I_1) \cap \mathbf{cover}_{\mathcal{T}}(I_2) = \mathbf{cover}_{\mathcal{T}}(I_2) \neq \mathbf{cover}_{\mathcal{T}}(I_1)$. De la même manière que dans la propriété précédente, nous pouvons remplacer toutes les occurences de I_2 par $I_1 \cup I_2$ et conserver I_1 dans les futures considérations.

Propriété 4
Si $\mathbf{cover}_{\mathcal{T}}(I_1) \neq \mathbf{cover}_{\mathcal{T}}(I_2)$ alors $\mathbf{cover}_{\mathcal{T}}(I_1 \cup I_2) = \mathbf{cover}_{\mathcal{T}}(I_1) \cap \mathbf{cover}_{\mathcal{T}}(I_2) \neq \mathbf{cover}_{\mathcal{T}}(I_2) \neq \mathbf{cover}_{\mathcal{T}}(I_1)$. Dans ce cas, on ne peut rien éliminer car I_1 et I_2 ont des fermetures différentes. On peut en revanche ajouter le noeud $I_1 \cup I_2$ et son tidset associé $\mathbf{cover}_{\mathcal{T}}(I_1) \cap \mathbf{cover}_{\mathcal{T}}(I_2)$.

Closet

L'algorithme Closet ([11]) est l'adaptation par ses auteurs de l'algorithme FP-Growth pour l'extraction des itemsets clos fréquents. En effet, il utilise différentes optimisations pour extraire rapidement des itemsets clos et ainsi réduire d'une part le nombre d'appels récursifs de FP-Growth et d'autre part de réduire l'espace mémoire total nécessaire.

Bibliographie

[1] R. Agrawal, T. Imielinski, and A.N. Swani. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–213. ACM Press, 1993.

[2] F. Bodon. A survey on frequent itemset mining. 2006.

[3] B. Goethals. Survey on frequent pattern mining.

[4] M.J. Zaki and C. Hsiao. Charm : An efficient algorithm for closed association rule mining. Technical report, Rensselaer Polytechnic Institute, 1999.

[5] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *roc. 7th Int. Conf. Database Theory (ICDT'99)*, pages 398–416, Jerusalem, Israel, January 1999.

[6] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB '94 : Proceedings of the 20th International Conference on Very Large Databases*, pages 487–499. Morgan Kaufmann Publishers Inc., 1994.

[7] Mohammed Javeed Zaki. Scalable algorithms for association mining. *IEEE Trans. Knowl. Data Eng.*, 12(3) :372–390, 2000.

[8] J. Han, P. Jian, Y. Yiwen, and M. Runying. *Mining Frequent Patterns without Candidate Generation : A Frequent-Pattern Tree Approach*, volume 8, pages 53–87. 2004.

[9] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, volume 26(2) of *SIGMOD Record*, pages 255–264. ACM Press, 1997.

[10] *Fourth International Conference on Software Engineering, Research, Management and Applications (SERA 2006), 9-11 August 2006, Seattle, Washington, USA*. IEEE Computer Society, 2006.

[11] J. Pei, J. Han, and R. Mao. Closet : An efficient algorithm for mining frequent closed itemsets. In *Proc. 2000 ACM-SIGMOD Int. Workshop Data Mining and Knowledge Discovery (DMKD2000)*, pages 11–20, Dallas, TX, 2000.