Motifs Séquentiels Flous : Un Peu, Beaucoup, Passionnément

Céline Fiot, Anne Laurent, Maguelonne Teisseire

Motifs séquentiels flous : un peu, beaucoup, passionnément

Céline Fiot, Anne Laurent, Maguelonne Teisseire

LIRMM - UM II, 161 rue Ada, 34392 Montpellier Cedex 5
fiot, laurent, teisseire@lirmm.fr

Résumé. La plupart des bases de données issues du monde réel sont constituées de données numériques et historisées (données de capteurs, données scientifiques, données démographiques). Dans ce cadre, les algorithmes d'extraction de motifs séquentiels, s'ils sont adaptés au caractère temporel des données, ne permettent pas le traitement de données numériques. Les données sont alors pré-traitées pour les transformer en données binaires, ce qui entraîne une perte d'information. Des algorithmes ont donc été proposés pour traiter les données numériques sous forme d'intervalles et d'intervalles flous notamment. En ce qui concerne la recherche de motifs séquentiels fondée sur des intervalles flous, les deux méthodes de la littérature ne sont pas satisfaisantes car incomplètes soit dans le traitement des séquences soit dans le calcul du support. Dans cet article, nous proposons donc trois méthodes d'extraction de motifs séquentiels flous (SpeedyFuzzy, MiniFuzzy et TotallyFuzzy) et en détaillons les algorithmes sous-jacents en soulignant les différents niveaux de fuzzification. Ces algorithmes sont implémentés et évalués à travers différentes expérimentations menées sur des jeux de tests synthétiques.

1 Introduction

La plupart des bases de données issues du monde réel sont constituées de données numériques et historisées (données de capteurs, données démographiques, ...). Dans le cadre de la fouille de grandes bases de données, peu de travaux ont été réalisés pour traiter cette problématique et la majorité des propositions sont restreintes aux règles d'association (Fu et al., 1998; Kuok et al., 1998; Srikant and Agrawal, 1996). Une première proposition (Srikant and Agrawal, 1996) traite les données quantitatives pour la recherche de règles d'association grâce à un découpage des attributs en intervalles discrets. Toutefois, ce découpage peut dissimuler des associations fréquentes en raison des bornes trop restrictives des différents intervalles. Plus récemment, l'utilisation de la théorie des sous-ensembles flous a permis des coupures moins brutales entre les intervalles, ce qui conduit à l'obtention de règles plus pertinentes.

Contrairement aux approches basées sur les règles d'association, les algorithmes d'extraction de motifs séquentiels permettent de prendre en compte le caractère temporel des données (suivi, évolution de phénomènes, concepts émergeants, détection ...) et sont donc adaptés aux données historisées. Néanmoins, ils ne permettent pas le traitement des données numériques. En effet, de telles données doivent être pré-traitées afin

d'être transformées en données binaires (absence/présence), ce qui entraîne nécessairement une perte d'information.
Notre proposition se situe dans ce contexte : comment prendre en compte les données numériques à l'aide des motifs séquentiels ? Dans cet article, nous présentons une approche complète et efficace d'extraction de motifs séquentiels flous qui permet de considérer les données numériques. Cette approche est fondée sur la définition d'intervalles et plus précisément sous forme d'intervalles flous. Les motifs obtenus sont du type "60% des gens qui achètent *beaucoup* de bonbons et *peu* de jeux vidéo achètent ensuite *beaucoup* de dentifrice". Nous définissons trois approches SpeedyFuzzy, MiniFuzzy et TotallyFuzzy qui diffèrent dans leur définition du support. Ceci permet à l'utilisateur final de choisir entre rapidité d'obtention des résultats et précision des motifs fréquents obtenus. L'implémentation des différentes solutions est basée sur un parcours des données efficace et original, extension de l'algorithme PSP proposé par (Masseglia et al., 1998). Les tests réalisés sur des jeux de données synthétiques soulignent la faisabilité d'une approche floue, d'autres expérimentations, actuellement en cours, tendent à montrer sa robustesse.

Cet article est structuré de la façon suivante : la section 2 présente les intervalles flous, les motifs séquentiels ainsi que les lacunes des propositions antérieures de recherche de motifs séquentiels flous. La section 3 présente les trois algorithmes (SpeedyFuzzy, MiniFuzzy et TotallyFuzzy) en détaillant les différentes définitions du support et leur implication dans le parcours des données. TotallyFuzzy, l'algorithme le plus complexe, est ensuite détaillé et illustré à l'aide d'un exemple. La section 4 présente la mise en œuvre des algorithmes, les expérimentations menées et l'intérêt des résultats obtenus. Enfin la section 5 conclut et présente les différentes perspectives associées à ce travail.

2 Motifs séquentiels, intervalles flous et motifs séquentiels flous

Nous décrivons brièvement les concepts de base de la théorie des sous-ensembles flous et des motifs séquentiels pour ensuite examiner les deux propositions existantes autour des motifs séquentiels flous en soulignant leurs faiblesses.

2.1 Théorie des sous-ensembles flous

La théorie des sous-ensembles flous, introduite par (Zadeh, 1965), autorise l'appartenance partielle à une classe, et donc la gradualité de passage d'une situation à une autre. Cette théorie constitue une généralisation de la théorie classique des ensembles, des situations intermédiaires entre le tout et le rien étant admises. Dans ce cadre, un objet peut donc appartenir à un ensemble et en même temps à son complément.

On considère par exemple l'univers X des tailles possibles d'un individu. Un **sous-ensemble flou** A (par exemple *Petit* ou *Grand*) est défini par une fonction d'appartenance μ_A qui décrit le degré avec lequel chaque élément $x \in X$ appartient à A, ce degré étant compris entre 0 et 1. Ainsi, un individu de 1m63 pourra à la fois être grand

et petit avec, par exemple, un degré de 0.7 pour le sous-ensemble flou *Grand* et de 0.3 pour le sous-ensemble flou *Petit*.

Les **opérateurs** en logique floue sont une généralisation des opérateurs classiques. On considère notamment la négation, l'intersection et l'union. L'opérateur $\top$ ou t-norme (norme triangulaire) est l'opérateur binaire d'intersection : $\mu_{A\cap B}(x) = \top(\mu_A(x), \mu_B(x))$. L'opérateur $\perp$ ou t-conorme (conorme triangulaire) est l'opérateur binaire d'union : $\mu_{A\cup B}(x) = \perp(\mu_A(x), \mu_B(x))$. Nous noterons $\overline{\top}$ (resp. $\underline{\perp}$) l'opérateur $\top$ (resp. $\perp$) généralisé au cas n-aire.

La **cardinalité d'un sous-ensemble flou** est le nombre d'éléments qui appartiennent à cet ensemble, ce qui peut être réalisé selon quatre comptages : (1) En comptabilisant tous les éléments pour lesquels le degré d'appartenance est non nul (2) En considérant seulement les éléments dont le degré d'appartenance dépasse un certain seuil. Ce comptage est appelé *comptage seuillé* (3) En sommant les degrés d'appartenance de chaque élément. Ce comptage est appelé *sigma-comptage* (4) En sommant les degrés d'appartenance supérieurs à un seuil donné, appelé *sigma-comptage seuillé*.

2.2 Les motifs séquentiels

Considérons DB une base regroupant l'ensemble des achats réalisés par des clients. Chaque n-uplet t correspond à une transaction et consiste en un triplet *(id-client, id-date, itemset)* : l'identifiant du client, la date de l'achat ainsi que l'ensemble des produits (items) achetés .
Soit $I = \{i_1, i_2, ..., i_m\}$ l'ensemble des *items*. Un *itemset* est un ensemble d'items non vide noté $(i_1 i_2 \dots i_k)$. Il s'agit d'une représentation non ordonnée. Une *séquence* s est une liste ordonnée, non vide, d'itemsets notée $< s_1 s_2 ... s_p >$. Une *n-séquence* est une séquence composée de n items.
Une séquence $< s_1\ s_2 ... s_p >$ est une sous-séquence d'une autre séquence $< s'_1\ s'_2\ ...s'_m >$ s'il existe des entiers $l_1 < l_2 < ...l_j ... < l_n$ tels que $s_1 \subseteq s'_{l_1}$, $s_2 \subseteq s'_{l_2}, ...,\ s_p \subseteq s'_{l_n}$. Tous les achats d'un même client sont regroupés et triés par date. Ils constituent la séquence de données du client. Un client *supporte* une séquence s si s est incluse dans la séquence de données de ce client (s est une sous-séquence de la séquence de données). Le *support* d'une séquence s est le pourcentage des clients qui supportent s. Soit $minSupp$ le support minimum fixé par l'utilisateur, une séquence qui vérifie le support minimum (*i.e.* dont le support est supérieur à $minSupp$) est une *séquence fréquente*.

2.3 Les motifs séquentiels flous

Intuitivement, pour obtenir des motifs séquentiels flous, il s'agit de partitionner les quantités de chaque item ou produit acheté en plusieurs sous-ensembles flous puis d'utiliser ces sous-ensembles flous pour la recherche de séquences fréquentes.

La première proposition d'une approche de recherche de motifs séquentiels flous a été réalisée par (Hong et al., 2001). Leur proposition est basée sur un découpage en intervalles flous. Cependant, pour minimiser le nombre d'items flous manipulés, ils ne conservent, pour chaque item, que le sous-ensemble flou de cardinal le plus élevé pour toute la base (par Σ-comptage).

(Chen et al., 2001; Hu et al., 2003) ont adopté quant à eux une approche très théorique du problème sans algorithme ou implémentation. Leur proposition présente un formalisme et des notations ambigus pour le calcul du support d'un itemset flou et donc d'une séquence floue. Il est notamment difficile d'identifier les différences dans le calcul du support des séquences <(a)(b)> et <(a b)>. Or ce point est fondamental dans un contexte de recherche de motifs séquentiels puisque les dates associées aux items interviennent lors de l'extraction des fréquents. Les auteurs ne proposant ni illustration ni démarche algorithmique dans leurs publications, il est difficile de se satisfaire de ces travaux.

3 Proposition : du Peu flou au Grand flou

Pour pallier les lacunes des approches existantes de recherche de motifs séquentiels flous, nous avons tout d'abord souhaité une définition claire et non ambigüe des concepts associés (item, itemset, g-k-séquence et support flous) (Fiot et al., 2004). Le cœur de cet article est d'en proposer plusieurs mises en œuvre possibles en fonction, en particulier, de la méthode de comptage adoptée (comme introduit dans le paragraphe 2.1). Pour cela nous proposons trois algorithmes, associés chacun à une définition précise du support, permettant ainsi trois niveaux de "fuzzification" lors de la recherche de motifs séquentiels flous.

3.1 Préambule : item, itemset, g-k-séquence

Les notions d'item et d'itemset sont redéfinies par rapport aux motifs séquentiels classiques. Un **item flou** est l'association d'un item et d'un sous-ensemble flou correspondant noté $[x, a]$ avec x un item et a un sous-ensemble flou. Par exemple, $[bonbons, beaucoup]$ est un item flou où $beaucoup$ est un sous-ensemble flou défini par une fonction d'appartenance. Un **itemset flou** est alors défini comme un ensemble d'items flous. On note (X, A) un itemset flou, X étant un ensemble d'items et A un ensemble de sous-ensembles flous associés. Par exemple, (X,A)=($[bonbons, beaucoup][soda, peu]$) est un itemset flou. Enfin, une g-k-séquence $S =< s_1 \cdots s_g >$ est définie comme une séquence composée de g itemsets flous $s = (X, A)$ regroupant au total k items flous. Par exemple, la séquence $< ([bonbons, beaucoup][jeuxvideos, peu])([soda, beaucoup]) >$ regroupe 3 items flous au sein de 2 itemsets. Il s'agit d'une 2-3-séquence floue.

Nous utiliserons dans la suite de cet article les notations suivantes : $\mathcal{C}$ représente l'ensemble des clients et $\mathcal{T}_c$ est l'ensemble des transactions d'un client c. $\mathcal{I}$ représente l'ensemble des attributs et $t[i]$ la valeur de l'attribut i pour la transaction t. Chaque attribut i est partitionné en sous-ensembles flous.
Pour illustrer les différentes définitions, nous utilisons la base d'achats décrite figure 1 (une case vide indique que le produit n'a pas été acheté). Nous considérons le degré minimal ω=0.49 et le support minimal $minSupp = 0.55$. Pour les opérateurs $\overline{\top}$ et $\underline{\perp}$, nous adoptons respectivement min et max. L'opérateur d'agrégation $\odot$ correspond à la moyenne.
Dans un premier temps, il s'agit de convertir la base des quantités en base de degrés d'appartenance. Chaque attribut est donc partitionné en sous-ensembles flous selon

la figure 2 qui représente les fonctions d'appartenance pour chaque sous-ensemble. La construction des partitions est réalisée de manière automatique en séparant les univers des quantités en intervalles en regroupant la même proportion de clients et en "fuzzifiant" ensuite ces intervalles pour garantir une meilleure généralisation.
A partir des fonctions d'appartenance ci-dessus, on obtient les degrés d'appartenance de chaque transaction pour chacun des sous-ensembles flous. La figure 3 décrit ces valeurs pour l'ensemble des transactions du client 1.

		Items				
Clients	**Date**	bonbons	dentifrice	soda	ballon	jeu video
C1	d1	2				
	d2	1	3	1		
	d3	4		1		
	d4			1	5	
	d5			2		2
C2	d1	2			1	
	d2			2		
	d3		4	1		
	d4	3				
C3	d1					3
	d2	3	1			
	d3				4	5
	d4			2		
	d5		2			
C4	d1					
	d2					
	d3	2				4
	d4			3		
	d5		2			
	d6			2		

FIG. 1 – *Transactions*

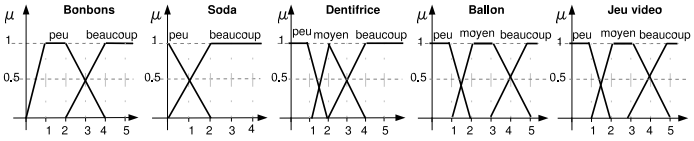

FIG. 2 – *Fonctions d'appartenance pour les différents sous-ensembles d'attribut*

		Items												
		bonbons		dentifrice			soda		ballon			jeu video		
	Date	peu	bcp	p	my	bcp	peu	bcp	p	my	bcp	p	my	bcp
C1	d1	0.75	0.25											
	d2	1			0.5	0.5	0.5	0.5						
	d3	0.25	0.75				0.5	0.5						
	d4						0.5	0.5			1			
	d5							1					1	

FIG. 3 – *Degrés d'appartenance pour le Client 1*

3.2 Supports Flous

Le **support d'un itemset flou** se calcule comme le pourcentage de clients supportant cet itemset flou par rapport au nombre total de clients dans la base :

$$FSupp_{(X,A)} = \frac{\sum_{c \in \mathcal{C}} \Big[S(c,(X,A)) \Big]}{|\mathcal{C}|}$$

où le degré support $S(c,(X,A))$ indique si le client c supporte l'itemset flou (X,A).
Nous avons pu constater, paragraphe 2.1, que la cardinalité d'un sous-ensemble flou dépend de la technique de comptage utilisée. Nous transposons trois de ces techniques dans le contexte des motifs séquentiels flous et proposons trois définitions du support :
• SpeedyFuzzy se base sur le comptage "supporte / ne supporte pas" (première méthode de comptage). Pour le support d'un itemset flou, cela consiste à comptabiliser chaque client ayant réalisé au moins une fois la séquence d'achats. Quel que soit le degré d'appartenance de l'achat du client pour l'item flou, si celui-ci est non nul, chaque client aura le même poids :
$S_{SpeedyFuzzy}(c,(X,A)) = 1$ si $\forall [x,a] \in (X,A)$, $\mu_a\big(t[x]\big) > 0$ et 0 sinon.
• MiniFuzzy repose sur un comptage seuillé (deuxième méthode de comptage). Dans cette méthode, le nombre de clients supportant un itemset flou n'est incrémenté que si chaque item de la séquence candidate vérifie le seuil pour le degré d'appartenance dans la séquence d'achats du client :
$S_{SpeedyFuzzy}(c,(X,A)) = 1$ si $\forall [x,a] \in (X,A)$, $\mu_a\big(t[x]\big) > \omega$ et 0 sinon.
• TotallyFuzzy réalise un Σ-comptage seuillé (quatrième méthode de comptage). Il s'agit de prendre en compte l'importance de chaque itemset flou parmi les transactions d'un client dans le calcul du support. Pour cela, on définit α qui représente la fonction d'appartenance seuillée :

$$\alpha_a\big(t[x]\big) = \begin{cases} \mu_a\big(t[x]\big) & \text{si } \mu_a\big(t[x]\big) > \omega \\ 0 & \text{sinon} \end{cases}$$

Nous obtenons alors : $S_{TotallyFuzzy}(c,(X,A)) = \underline{\perp}_{j=1}^{\theta_c} \overline{\top}_{[x,a]\in(X,A)} [\alpha_a\big(t_j[x]\big)]$, où $\overline{\top}$ et $\underline{\perp}$ sont les opérateurs de t-norme et t-conorme généralisés.

Le Σ-comptage (troisième méthode) est en fait un Σ-comptage seuillé, avec un seuil ω à zéro, il n'est donc pas nécessaire d'en faire un cas particulier.

3.3 Les algorithmes SpeedyFuzzy et MiniFuzzy

La démarche adoptée pour SpeedyFuzzy et MiniFuzzy est similaire. Pour chaque client, il s'agit de parcourir l'ensemble de ses transactions pour trouver la séquence candidate. Pour chaque itemset de la séquence, il est nécessaire de vérifier si le degré d'appartenance est : non nul pour SpeedyFuzzy ou supérieur au seuil ω pour MiniFuzzy. Dès que la séquence candidate est validée (l'ensemble ordonné des itemsets est supporté par le client), le parcours dans les transactions du client est stoppé et le support de la séquence est incrémenté.
L'algorithme SpeedyFuzzy (resp. MiniFuzzy) se base sur la fonction de calcul du support *CalcSpeedySupp* (resp. *CalcMiniSupp*) et la fonction *FindSpeedy-Seq* (resp. *FindMiniSeq*) qui recherche une séquence candidate parmi toutes les transactions d'un client.

3.4 TotallyFuzzy

L'algorithme TotallyFuzzy quant à lui est plus complexe car il repose sur un Σ-comptage seuillé. Ainsi pour chaque client et chaque séquence, il faut considérer le meilleur degré d'appartenance des itemsets parmi les transactions de ce client. Ce degré

est calculé comme l'agrégation des supports des itemsets de la séquence. Il faut également respecter l'ordre des itemsets flous retenus. Ceci impose un parcours exhaustif des transactions. Néanmoins, à partir de la démarche proposée dans (Fiot et al., 2004), nous proposons une mise en œuvre efficace d'un tel parcours par l'intermédiaire de la notion de chemin. Un chemin correspond à une instanciation possible des itemsets de la séquence candidate dans les transactions du client. Plusieurs chemins peuvent être initiés pour un client et il s'agit de conserver celui de plus fort degré, s'il est complet, pour le calcul du support.

3.4.1 Illustration

Toujours à partir des transactions du client 1, figure 3, nous allons illustrer la démarche adoptée par TotallyFuzzy pour calculer le support de la séquence candidate g-S $= <([bonbons, peu])([soda, bcp])>$, avec un seuil ω à 0.2. Un chemin est un triplet contenant la séquence déjà trouvée, l'item suivant recherché ainsi que les degrés d'appartenance des différents itemsets. Pour initier le processus, il y a création d'un chemin $ch1$ qui contient $(\varnothing, ([bonbons, peu]), 0)$ qui correspond à la séquence déjà trouvée (seq), l'item recherché ($rechCour$) et le degré d'appartenance ($degCour$). Pour la transaction d1, $rechCour = d1[1]$, donc le chemin $ch1$ est mis à jour par $(< ([bonbons, peu]) >, ([soda, bcp]), 0.75)$.
Ensuite, pour la transaction d2, un nouveau chemin est créé $ch2 \leftarrow (< ([bonbons, peu]) >, ([soda, bcp]), 1)$ car d2 contient l'item $[bonbons, peu]$ qui est le premier item de la séquence candidate. $ch1$ est mis à jour car d2 contient $[soda, bcp]$. $ch1$ est clos car il contient tous les éléments de la séquence g-S. La transaction d3 est ensuite examinée. Elle contient $ch2$.rechCour, le chemin $ch2$ est donc modifié en $(< ([bonbons, peu])$ $([soda, bcp]) >, \varnothing, 0.75)$, ce chemin est clos. Néanmoins, le parcours est optimisé et ne conserve, pour deux chemins au même stade de parcours, que celui de meilleur degré d'appartenance. Ainsi, le chemin $ch1$ est éliminé de la liste des chemins pour le client 1 car ayant un degré d'appartenance inférieur à $ch2$. Le parcours se poursuit ensuite comme indiqué figure 4.

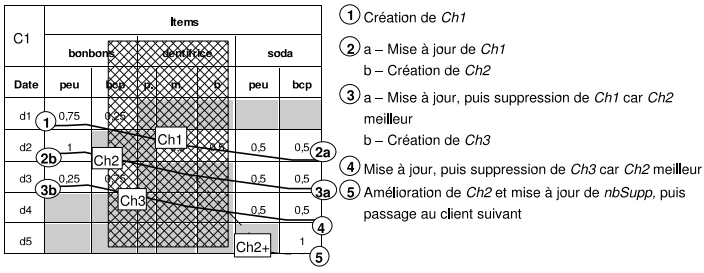

Fig. 4 – *Exemple de parcours pour le client 1*

3.4.2 Les fonctions utilisées

L'algorithme TotallyFuzzy utilise la fonction *FindTotallySeq* qui réalise un parcours ordonné des transactions d'un client. Lorsque le premier itemset de la séquence est trouvé, un chemin est créé avec le support de l'itemset. Les transactions suivantes

FindTotallySeq - **Input :** g-S, g-k-séquence candidate, T, ensemble de transactions à parcourir
Ouput : m, le degré support de la meilleure représentation de g-S instanciée parmi les transactions de T

$Chemins$: **liste de parcours → (seq, rechCour, degCour)**

[seq est la sous-séquence de g-S déjà trouvée, rechCour est l'itemset suivant dans g-S]

[degCour est la liste des degrés d'appartenance des différents itemsets de seq]

$Chemins \leftarrow$Chemin($\varnothing$, gS.first, 0)

Pour chaque transaction $t \in$ T **faire**
  Pour chaque parcours $ch \in Chemins$, non mis à jour à la transaction t **faire**
    Si (ch non clos) **Alors**
      Si (ch.rechCour $\in t$) **Alors**
        [t contient l'itemset si le degré de chaque item de l'itemset est supérieur au seuil ω]
        ch.degCour $\leftarrow ch$.degCour | $\overline{\top}_{[x,a]\in ch.\text{rechCour}} \alpha_a(t[x])$;
        Update(ch) ;
      Fin Si
    Fin Si
    Pour j **de** 2 **à** ch.rechCour -1 **faire**
      [on recherche ici une amélioration possible du chemin courant]
      Si ((gS.get(j) $\in t$) & ($\overline{\top}_{[x,a]\in gS.get(j)} \alpha_a(t[x]) > ch$.degCour[j])) **Alors**
        newRechCour $\leftarrow$ gS.get(j) ;
        Pour i **de** 1 **à** j-1 **faire**
          newSeq $\leftarrow$ newSeq | gS.get(i) ;
          newDegCour $\leftarrow$ newDegCour | ch.degCour[i] ;
        Fin Pour
        newDegCour $\leftarrow$ newDegCour | $\overline{\top}_{[x,a]\in gS.get(j)} \alpha_a(t[x])$;
        $Chemins \leftarrow Chemins \cup$ update((newSeq, newRechCour, newDegCour)) ;
      Fin Si
    Fin Pour
  Fin Pour
  Si ((gS.first $\in t$) & (non(PremierPassage))) **Alors**
    [un nouveau chemin est créé si on rencontre le premier itemset de la séquence]
    $ch \leftarrow$Chemin($\varnothing$, gS.first, $\overline{\top}_{[x,a]\in gS.first} \alpha_a(t[x])$) ;
    $Chemins \leftarrow Chemins \cup ch$;
    Update(ch) ;
  Fin Si
  Chemins.Optimize() ; *[élimination des moins bons chemins]*
Fin Pour
Pour chaque parcours $ch \in Chemins$ **faire**
  Si (ch non clos) **Alors**
    Cut(ch) ; *[élimination des chemins qui ne couvrent pas toute la séquence]*
  Fin Si
Fin Pour
$ch \leftarrow Chemins.first$; *[Chemins ne contient plus que le meilleur parcours complet]*
$m \leftarrow \odot(ch.\text{degCour})$; *[on agrège pour trouver puis renvoyer le degré support]*
Retourner m ;

FIG. 5 – *Fonction FindTotallySeq*

sont examinées pour chercher soit la suite de la séquence, soit une nouvelle fois le début de la séquence, soit une amélioration des chemins trouvés. Tous les *chemins* possibles sont ainsi complétés au fur et à mesure de l'examen des transactions du client, le degré support du meilleur chemin pour toute la séquence est ensuite retourné.
La fonction *Update* permet de mettre à jour l'état d'avancement de chaque chemin. La fonction *Optimize*, non présentée ici, permet d'éliminer au fur et à mesure les chemins si un chemin identique mais de meilleur degré d'appartenance a été trouvé pour le client. La fonction *CalcTotallySupp* calcule le support d'une séquence candidate en agrégeant, pour chaque client, la valeur du chemin optimal pour la séquence.

CalcTotallySupport
Input : gS, g-k-séquence candidate ;
Ouput : $FSupp$ support flou de la séquence gS ;
$FSupp, nbSupp, m \leftarrow 0$;
Pour chaque client $c \in \mathcal{C}$ **faire**
  $m \leftarrow$ FindTotallySeq(g-S, $\mathcal{T}_c$) ;
  [on ajoute au support courant de la séquence le degré support du client]
  $nbSupp \mathrel{+}= m$;
Fin Pour
$FSupp \leftarrow nbSupp/|\mathcal{C}|$;
Retourner $FSupp$;

Update
Input : ch, parcours à mettre à jour
ch.seq $\leftarrow$ ch.seq $\cup ch$.rechCour ;
Si (ch.rechCour.next $\neq \varnothing$) **Alors**
  ch.rechCour $\leftarrow$ ch.rechCour.next ;
Sinon
  [une fois toute la séquence trouvée, on clot le chemin]
  Clore(ch) ;
Fin Si

FIG. 6 – *Fonctions CalcTotallySupport et Update*

4 Expérimentations

Dans cette section, nous présentons une comparaison des performances des trois algorithmes SPEEDYFUZZY, MINIFUZZY et TOTALLYFUZZY et de l'algorithme PSP. Ces expérimentations ont été menées sur un PC - système d'exploitation Linux 2.6.7, CPU 500MHz et 256MB de RAM. Les trois algorithmes ont été implémentés en Java sur le principe de PSP (Masseglia et al., 1998) dont, en particulier, la structure de Prefix Tree utilisée pour stocker les séquences candidates et fréquentes.

Nous avons utilisé des jeux de données synthétiques. Les bases de données ont été créées en plusieurs étapes à partir d'un outil basé sur IBMQuest (IBM, 2003). Cet outil permet de générer des bases de données au format client-date-item auxquelles on vient ajouter aléatoirement des quantités. Dans un second temps, un partitionnement automatique est réalisé en utilisant un sous-module de Weka (Witten and Frank, 2000). IBMQuest génère des données obéissant à une loi de Poisson, les bases obtenues sont très denses et rendent les tests ardus car elles intégrent de très fortes corrélations ainsi que des séquences à forte répétition d'items (par exemple <(2)(8)(2 2)(2 8 8)(8)>) sur lesquelles la méthode générer-élaguer n'est pas forcément la plus appropriée. Les différentes bases de degrés d'appartenance retenues pour les tests comportent les achats correspondant à une moyenne de 50, 100 puis 1000 clients ayant en moyenne 5 à 10

transactions de 5 à 15 items environ pour un ensemble de 15 à 20 items au total. Chaque item est partitionné en trois sous-ensembles flous. Les différents résultats représentent la moyenne des tests effectués sur les différentes bases.

La figure 7 présente le nombre de motifs séquentiels extraits en fonction du support minimal. La figure 10 présente le temps d'extraction des motifs séquentiels en fonction du support minimal.

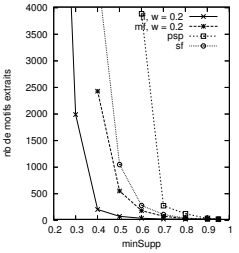

FIG. 7 – *Nombre de motifs/minSupp, selon l'approche*

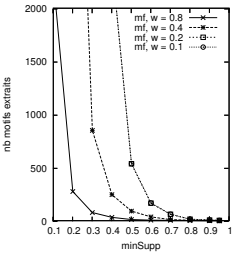

FIG. 8 – *Pour* MINIFUZZY *selon* ω

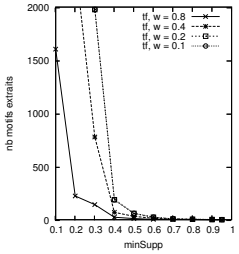

FIG. 9 – *Pour* TOTALLYFUZZY *selon* ω

Sur les figures 8 et 9, on peut observer l'influence du paramêtre ω sur le nombre de motifs séquentiels extraits par MINIFUZZY et TOTALLYFUZZY.

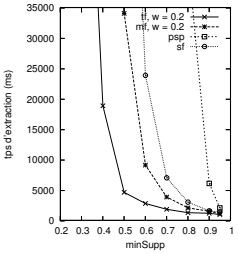

FIG. 10 – *Temps/minSupp, selon l'approche*

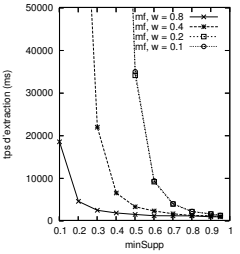

FIG. 11 – *Pour* MINIFUZZY *selon* ω

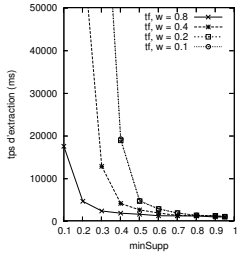

FIG. 12 – *Pour* TOTALLYFUZZY *selon* ω

Toutes ces expérimentations indiquent un comportement similaire à l'approche PSP. Il faut pondérer les résultats des temps d'exécution par rapport au nombre de candidats générés qui décroît pour les différents algorithmes de PSP à SPEEDYFUZZY , MINIFUZZY , et TOTALLYFUZZY. Ces tests mettent ainsi en évidence la faisabilité d'une approche floue dès lors qu'elle s'appuie sur des algorithmes et des structures de données adéquats. Les figures 8, 9, 11, 12 soulignent l'interêt du comptage seuillé dans une extraction de connaissances avec faible support comme en particulier la détection de signaux faibles.

Nous réalisons actuellement des tests sur des bases synthétiques à répartition homogène des items dans les séquences et comportant un nombre de clients plus élevé (10000 à 100000), afin de montrer la robustesse de notre approche.
Par ailleurs, nous cherchons à définir un certain nombre de critères pour évaluer les motifs séquentiels flous extraits par rapport aux motifs séquentiels classiques.

5 Conclusion et perspectives

Dans cet article, nous présentons trois algorithmes permettant une implémentation efficace des motifs séquentiels flous. Les motifs séquentiels flous ont en effet été récemment étudiés dans la littérature dans (Hong et al., 2001; Chen et al., 2001) et plus récemment dans (Fiot et al., 2004). Ils sont essentiels pour la manipulation de données numériques stockées de manière historisée, comme c'est le cas par exemple pour les données démographiques, les données de capteurs, *etc.* L'extraction de motifs séquentiels sur de telles bases est en effet très intéressante pour la détection d'enchaînements d'événements et la recherche de tendances, mais les algorithmes classiques existants ne permettent pas la gestion de données numériques. Dans (Fiot et al., 2004), nous avons défini de manière claire et complète les différents concepts associés aux motifs séquentiels flous, ces concepts étant présentés de manière incomplète dans les autres travaux portant sur cette problématique. Les algorithmes proposés ici, SpeedyFuzzy, MiniFuzzy et TotallyFuzzy ont été implémentés et ont fait l'objet de tests qui montrent l'intérêt de notre approche et la faisabilité de la mise en œuvre de méthodes intégrant la logique floue. L'utilisateur peut alors choisir entre les trois niveaux de "fuzzification" que nous introduisons au travers de nos trois algorithmes. Ce choix permet l'extraction de fréquents qui sont plus ou moins précis et donc obtenus plus ou moins rapidement. Ces travaux permettent d'envisager l'application de nos méthodes sur différents types de données, et nous envisageons en particulier d'étudier le problème du traitement des données manquantes dans le cadre des bases de données numériques et historisées. Ce problème est en effet récurrent dans le cas des données réelles et n'a pas été traité à ce jour dans le cadre des motifs séquentiels.

Références

Chen, R.-S., Tzeng, G.-H., Chen, C.-C., and Hu, Y.-C. (2001). Discovery of fuzzy sequential patterns for fuzzy partitions in quantitative attributes. In *ACS/IEEE International Conference on Computer Systems and Applications (AICCSA)*, pages 144–150.

Fiot, C., Dray, G., Laurent, A., and Teisseire, M. (2004). A la recherche des motifs séquentiels flous. In *12èmes rencontres francophones sur la Logique Floue et ses Applications.*

Fu, A., Wong, M., Sze, S., Wong, W., , and Yu, W. (1998). Finding fuzzy sets for the mining of fuzzy association rules for numerical attributes. In *the First International Symposium on Intelligent Data Engineering and Learning (IDEAL)*, pages 263–268.

Hong, T., Lin, K., and Wang, S. (2001). Mining fuzzy sequential patterns from multiple-items transactions. In *Proceedings of the Joint 9th IFSA World Congress and 20th NAFIPS International Conference*, pages 1317–1321.

Hu, Y.-C., Chen, R.-S., Tzeng, G.-H., and Shieh, J.-H. (2003). A fuzzy data mining algorithm for finding sequential patterns. *International Journal of Uncertainty Fuzziness Knowledge-Based Systems*, 11(2) :173–193.

IBM (2003). Almaden research center.

Kuok, C. M., Fu, A. W.-C., and Wong, M. H. (1998). Mining fuzzy association rules in databases. *SIGMOD Record*, 27(1) :41–46.

Masseglia, F., Cathala, F., and Poncelet, P. (1998). The PSP approach for mining sequential patterns. In *Principles of Data Mining and Knowledge Discovery*, pages 176–184.

Srikant, R. and Agrawal, R. (1996). Mining quantitative association rules in large relational tables. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 1–12, Montreal, Quebec, Canada.

Witten, I. H. and Frank, E. (2000). Data mining : Practical machine learning tools with java implementations.

Zadeh, L. (1965). Fuzzy sets. *Information and Control*, 3(8) :338–353.

Summary

Most real world databases are constituted from historical and numerical data such as sensors data, scientific data or even demographic data. In this context, algorithms extracting sequential patterns, well adapted to the temporal aspect of the data, do not allow processing numerical information. Therefore, the data are pre-processed to be transformed into binary data, which leads to a loss of information. Thus, algorithms have been proposed to process numerical data by means of intervals and particularly fuzzy intervals. With regard to the search of sequential patterns based on fuzzy intervals, both former presented methods are unsatisfactory because they are incomplete either in the processing of sequences or in the support calculation. Therefore, we propose in the present paper three methods for extracting fuzzy sequential patterns (SpeedyFuzzy, MiniFuzzy and TotallyFuzzy) and we detail the subjacent algorithms by highlighting the different fuzzification levels. Finally, these algorithms are implemented and assessed through different experimentations carried out by means of different datasets.