Un nouvel algorithme d'extraction des motifs graduels appelé Sgrite

Tayou Djamégni Clémentin, Tabueu Fotso Laurent Cabrel, Kenmogne Edith Belise

Un nouvel algorithme d'extraction des motifs graduels appelé Sgrite

Extraction des motifs graduels

Tayou Djamégni Clémentin [*] [**] — Tabueu Fotso Laurent[**] — Kenmogne Edith Belise[**]

[*] Département de Génie Informatique, UIT-FV
[**] Département de Mathématiques-Informatique, FS
Université de Dschang, Cameroun
dtayou@gmail.com
laurent.tabueu@gmail.com
ebkenmongne@gmail.com

RÉSUMÉ. L'extraction des motifs graduels est un problème important en informatique et largement étudié par la communauté scientifique de fouille de données. Les motifs graduels traduisent des co-variations récurrentes entre les attributs d'une base de données. Plusieurs applications issues de nombreux domaines, tels que l'économie, la santé, l'éducation, le commerce, la bio-informatique, l'astronomie ou le web mining, reposent sur l'extraction des motifs graduels. Les algorithmes de découverte des motifs dans des grandes bases de données sont gourmands en temps CPU et en espace mémoire. Ceci pose clairement le problème d'amélioration des performances de ces algorithmes. Ce papier présente une technique d'amélioration des performances des algorithmes d'extraction des motifs graduels. L'exploitation de cette technique débouche sur un nouvel algorithme plus performant appelé Sgrite. Les expérimentations effectuées confirment l'intérêt de la technique proposée.

ABSTRACT. The extraction of frequent gradual pattern is an important problem in computer science and largely studied by the scientist's community of research in data mining. A frequent gradual pattern translates a recurrent covariation between the attributes of a database. Many applications issues from many domains, such as economy, health, education, market, bio-informatics, astronomy or web mining, are based on the extraction of frequent gradual patterns. Algorithms to extract frequent gradual patterns in the large databases are greedy in CPU time and memory space. This raises the problem of improving the performances of these algorithms. This paper presents a technique for improving the performance of frequent gradual pattern extraction algorithms. The exploitation of this technique leads to a new, more efficient algorithm called Sgrite. The experiments carried out confirm the interest of the proposed technique.

MOTS-CLÉS : Motif graduel, extraction de connaissances, optimisation,support graduel, matrice d'adjacence

KEYWORDS : Gradual pattern, knowledge extraction, optimization, gradual support, adjacency matrix

1. Introduction

De nos jours, les moyens informatiques permettent de produire et de stocker d'énormes masses de données numériques, dans nombreux domaines tels que l'économie, la santé, l'éducation. Ces données renferment un certain nombre de connaissances cachées qui décrivent des dépendances ou des corrélations, implicites et utiles. Ce qui pose le problème d'extraction automatique de connaissances nouvelles, utiles et valides, à partir de grandes quantités de données. Ceci est l'un des principaux objectifs de la fouille de données[12, 5, 3, 10, 9]. Dans ce contexte, une connaissance représente un schéma récurrent (corrélation, dépendance, co-variation entre attributs)[1, 2, 8, 10]. Ce papier est consacré à l'étude des algorithmes d'extraction des connaissances qui traduisent des covariations entre attributs, encore appelées motifs graduels. Étant donnée la grande taille des jeux de données(duplicité de chaque item en positif et négatif), les tailles et le grand nombre de matrices d'adjacence des motifs graduels générés simultanément en mémoire, il se pose clairement un problème de performance, ces observations justifient l'optimisation de l'extraction des motifs graduels. Les raisons du choix de grite[5] comme base sont d'une part qu'il est l'un des meilleurs algorithmes séquentiel, exhaustif le plus utilisé dans de nombreux travaux futurs[11, 12] dans la littérature, d'autre part son expression par les chemins maximaux du support graduel, bonne interprétation du support graduel dans la plupart des jeux de données. La particularité de ce travail est son orientation sur l'optimisation en architecture non distribuée de grite contrairement aux autres approches d'optimisation directement orientées sur les architectures multi-coeur[11]. Pour nos expérimentations, nous allons utiliser deux jeux de données synthétiques (F20Att100Li et F20Att500Li) et un jeu réel météorologique. La suite de ce papier est organisée comme suit. La section 2 est consacrée à la revue de la littérature. La section 3 présente une technique d'amélioration des performances des algorithmes d'extraction des motifs graduels. L'exploitation de cette technique débouche sur un nouvel algorithme plus performant et appelé Sgrite. La section 4 est consacrée aux expérimentations. La section 5 conclut le papier.

2. Revue de littérature

2.1. Notations et définitions

Item graduel[4, 5, 12] : Un item graduel, noté A^*, où A est un attribut et $* \in \{\leq, \geq, <, >\}$ exprime une variation sur les valeurs de l'attribut A. Si l'opérateur de comparaison $*$ est égal à $\geq$ (resp. $\leq$), A^* traduit une variation croissante (resp. décroissante) des valeurs de A.
Certains travaux[7, 4] ne considèrent que les opérateurs de comparaison stricts ($<, >$) et d'autres [6, 10] considèrent uniquement les opérateurs de comparaison large ($\leq, \geq$).
Itemset graduel[5, 10] : Un itemset graduel, noté $\{ (A_i , *i), \text{i = 1...k } \}$ ou $\{ A_i^{*i} , \text{i = 1...k } \}$, est une concaténation de plusieurs items graduels interprétée sémantiquement comme une conjonction d'items graduels. Il exprime une co-variation des valeurs de ses attributs. La taille d'un itemset graduel est le nombre de ses attributs.

Si l'attribut A représente l'âge et l'attribut S représente le salaire, alors l'itemset graduel $A^{\geq} S^{\geq}$ traduit la covariation "*plus l'âge augmente, plus le salaire augmente*".

Motif graduel : Un motif graduel est un item ou un itemset graduel.
Ordre induit par un motif[12, 5] : L'ordre $\preceq_M$ induit par un motif M = $\{A_i^{*i}, i = 1...k\}$ est défini comme suit : o $\preceq_M$ o' ssi $\forall j \in [1,k] A_j(o) *_j A_j(o')$, où o et o' sont deux objets et $A_j(o)$ désigne la valeur de l'attribut A_j pour l'objet o.
Chemin [5, 12] : Un sous-ensemble d'objets D = $\{o_1$,...,$o_m\}$ est un chemin pour un motif M = $\{(A_i , *_i)$, i = 1...k $\}$ s'il existe une permutation Π de D, notée $\{o_{\Pi(1)}, o_{\Pi(2)}, .., o_{\Pi(m)}\}$, tel que $\forall j \in [1,k], \forall l \in [1, m-1], A_j(o_{\Pi(l)}) *_j A_j(o_{\Pi(l+1)})$.
Chemin maximal [5, 12] : Un chemin est dit maximal par rapport à un motif M s'il est un plus long chemin complet par rapport à M. On note $\mathcal{L}^*$(M) l'ensemble des chemins maximaux par rapport au motif M.
Motif graduel complémentaire[5, 12] : Le complémentaire du motif graduel M = $\{$ (A_i^{*i}), i = 1...k $\}$ est le motif graduel c(M) = $\{$ $(A_i^{c(*i)})$, i = 1...k $\}$, où $c(*i)$ est le complémentaire de l'opérateur de comparaison $*i$.
Dans la littérature[12], $c(\leq) = \geq, c(\geq) = \leq, c(<) = >, c(>) = <$.

Tableau 1 – **base de données numérique de salaires**

id	Age(A)	Salaire(S)	Voiture(V)
o1	20	1200	1
o2	28	1850	1
o3	24	1200	0
o4	35	2200	1
o5	30	2000	1
o6	40	3400	1
o7	52	3400	2
o8	41	5000	2

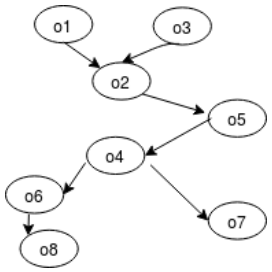


Figure 1 – graphe de précédence induit par le motif graduel $A^{>}S^{>}$

Le tableau 1 décrit une base de données simplifiée et constituée de trois attributs : Âge (A), Salaire (S) et Voiture(V) qui désigne le nombre d'allocations de voitures effectuées par l'intéressé. Dans ce contexte, l'item graduel $S^{\geq}$ signifie *"plus le salaire augmente"*, l'itemset graduel $A^{\geq}S^{\geq}$ signifie *"plus l'âge croit, plus le salaire croit"*. Les deux chemins *{o1,o2,o4} et {o6,o7}* associés au motif $A^{\geq}S^{\geq}$ sont complets. Le chemin *{o1,o2,o4,o6,o8}* associé au même motif est maximal.
Règle graduelle : Une règle graduelle est une expression de la forme $M \rightarrow M'$, où M et M' sont des motifs graduels, qui traduisent une implication de variation. M est la prémisse et M' la conclusion de la règle.

Extraction des motifs graduels : Le problème d'extraction des motifs graduels dans une base de données concerne la recherche de tous les motifs graduels fréquents. Un motif graduel est dit fréquent si son poids, encore appelé Support Graduel (SG), est supérieur au seuil de support minimal fixé par l'utilisateur.
Inclusion de motifs graduels : Le motif graduel A est inclus dans le motif B, noté $A \subseteq B$, si chaque item graduel de A apparait dans B.
Les deux propriétés suivantes sont vraies pour chacune des trois définitions de la notion de support graduel présentées à la section 2.2. Elles permettent d'élaguer significativement l'espace de recherche.

Lemme 1. *Propriété d'égalité du support graduel[12, 6] : Le support d'un motif graduel est égal au support graduel de son complémentaire.*

Lemme 2. *Propriété d'anti-monotonie du support graduel[12, 6, 10] : Si le motif graduel A est inclus dans le motif graduel B alors SG(A) $\geq$ SG(B).*

Treillis des motifs graduels : On appelle treillis des motifs graduels, le treillis induit par l'ensemble des motifs graduels muni de la relation d'inclusion. L'ensemble des noeuds du treillis est l'ensemble des motifs graduels. Un arc qui part d'un motif A vers un motif B traduit l'inclusion de A dans B.
Le treillis des motifs graduels représente l'espace de recherche des motifs graduels et fréquents. On peut exploiter le lemme 1 pour réduire l'espace de recherche de moitié, par exemple au treillis de motifs graduels ayant au moins un terme positif.
Treillis des motifs graduels ayant au moins un terme positif : Le treillis des motifs graduels ayant au moins un terme positif est le sous treillis du treillis des motifs graduels qui porte uniquement sur les motifs graduels ayant au moins un item graduel qui traduit une variation croissante. Un tel motif est de la forme $A_1^{\geq}\{A_i^{*i}\}, i = 2...k$.

2.2. Techniques d'extraction des motifs graduels

L'approche de régression linéaire proposée par Hüllermeier permet d'extraire des règles graduelles dont le support et la confiance sont supérieurs au seuil fixé par l'utilisateur. Cette approche ne considère que des données floues et les règles dont la prémisse et la conclusion soient de tailles inférieures ou égales à deux. Toutefois le concept de T-Norme qui fait partie de cette approche permet de s'affranchir de la limite de la taille de la prémisse et de la conclusion des règles. Dans l'approche Berzal et al.[4], le poids d'un motif graduel, encore appelé Support Graduel (SG) est égal au nombre de couples d'objets distincts qui vérifient l'ordre induit par le motif divisé par le nombre total de couples d'objets distincts de la base de données. Ainsi, SG(M) = $\frac{\sum_{(o,o')\in\mathcal{D}x\mathcal{D}} |o\preceq_M o'|}{|\mathcal{D}|(|\mathcal{D}|-1)}$, où M est un motif graduel et $\mathcal{D}$ la base de données. L'approche de Laurent et al [10]. est une amélioration de l'approche de Berzal et al[4] qui exploite le fait que si un couple d'objets distincts (o,o') vérifie l'ordre induit par un motif graduel alors son complémentaire (o',o) ne le vérifie pas. Dans cette approche, nous avons : SG(M) = $\frac{\sum_{(o,o')\in\mathcal{D}x\mathcal{D}} |o\preceq_M o'|}{(|\mathcal{D}|(|\mathcal{D}|-1))/2}$. Dans l'approche dite des chemins maximaux[6, 5] le support graduel d'un motif graduel M est égal à la longueur d'un chemin maximal associé à M divisé par le nombre total d'objets de la base de données . Dans cette approche nous avons : SG(M) = $\frac{\max_{D\in\mathcal{L}(M)} |D|}{|\mathcal{D}|}$.

2.3. Présentation de l'algorithme Grite

L'algorithme grite est un algorithme de découverte des motifs graduels sur la base des chemins maximaux. Il est inspiré du principe Apriori qui consiste à générer les motifs candidats et à ne retenir que ceux dont le support est supérieur au seuil de support minimal. À cet effet il génère d'abord les motifs candidats de taille 1, ensuite ceux de taille 2 et ainsi de suite. Il utilise la propriété d'anti-monotonie du support pour élagager l'espace de recherche. Dans cet algorithme, les deux opérations fondamentales sont la génération des candidats et le calcul du support. L'opération la plus sollicitée et la plus gourmande en temps CPU est le calcul du support puisqu'il est effectué pour chaque candidat et implique à chaque fois la recherche du plus long chemin du candidat courant.
Matrice d'adjacence : La matrice d'adjacence d'un motif graduel M est une matrice binaire qui associe à chaque couple d'objet (o,o') la valeur 1 si le couple d'objet respecte

l'ordre induit par le motif M et 0 sinon.
La matrice d'adjacence d'un motif induit un graphe de dépendance dont les noeuds sont des objets et la présence d'une dépendance entre deux noeuds est donnée par les entrés de la matrice d'adjacence.
Noeud père et Noeud fils : Étant donnée la matrice d'adjacence Adj_M d'un motif M, si $Adj_M[o, o'] = 1$ alors o est père de o' et o' est un fils de o.
Pour les définitions ci-dessous $\mathcal{O}$ est l'ensemble des objets de la base de données.
Noeud isolé : C'est un noeud qui n'est pas lié à un autre noeud, i.e qu'il n'a ni père ni fils. Étant donnée la matrice d'adjacence Adj_M d'un motif M, l'ensemble des noeuds isolés est :$\{o \in \mathcal{O} | \forall o' \in \mathcal{O}, Adj_M[o, o'] = 0 \wedge Adj_M[o', o] = 0\}$.
Racine : C'est un noeud n'ayant pas de père, mais qui est lié à tous les autres noeuds. Étant donnée la matrice d'adjacence Adj_M d'un motif M, l'ensemble des noeuds racines est : $\{o \in \mathcal{O} | \forall o' \in \mathcal{O}, Adj_M[o, o'] = 1 \wedge Adj_M[o', o] = 0\}$.
Feuille : C'est un noeud n'ayant pas de fils, mais qui n'est pas isolé. Étant donné la matrice d'adjacence Adj_M d'un motif M. l'ensemble des feuilles est : $\{o \in \mathcal{O} | \forall o' \in \mathcal{O}, Adj_M[o, o'] = 0, \exists o'' \in \mathcal{O} | Adj_M[o'', o] = 1\}$.

Lemme 3. *Jointure[6] : Soient 2 motifs graduels s et s', la relation suivante est vérifiée :* $MG_{s''} = MG_s AND MG_{s'}$.

L'algorithme appelé Jointure proposé dans[6] décrit le calcul de la jointure de deux matrices d'adjacence $\mathcal{M}_s$ et $\mathcal{M}_{s'}$ de deux motifs graduels s et s'. La matrice obtenue, notée $\mathcal{M}_{ss'}$, est la matrice d'adjacence de ss'. Elle est définie par : $\mathcal{M}_{ss'}[o, o'] = \mathcal{M}_s[o, o'] \wedge \mathcal{M}_{s'}[o, o'] \; \forall (o, o') \in \mathcal{O}X\mathcal{O}$. Cet algorithme normalise la matrice résultante $\mathcal{M}_{ss'}$ en y supprimant tous les objets isolés. L'algorithme appelé *RecursiveCorvering* dans [6] décrit le calcul récursif de la distance maximale entre un noeud donné et une feuille quelconque. Cet algorithme prend en entrée un objet *node* $\in \mathcal{O}$ et un vecteur, appelée *Memory*, de taille $|\mathcal{O}|$ dont les index sont les objets de $\mathcal{O}$. On suppose que *Memory*[o] = -1 $\forall o \in \mathcal{O}$ avant le premier appel de RecursiveCorvering. Pendant l'exécution de l'algorithme, *Memory*[o] contient la distance maximale courante entre o et une feuille quelconque. À la fin de tous les appels récursifs induits par l'appel initial de RecursiveCorvering, *Memory* contient la distance maximale entre chaque objet de $\mathcal{O}$ et une feuille quelconque. L'algorithme *RecursiveCorvering* est exploité pour calculer le support graduel d'un motif en l'appelant pour chaque noeud racine du graphe de précédence induit par ledit motif, et calcule par ailleurs le maximum des valeurs retournées par ces différents appels. Le principal défaut de cet algorithme provient de ce qu'il effectue deux balayages de la descendance de l'argument *node*. Le premier balayage est orienté des pères vers les fils, et le second est orienté des fils vers les pères. Étant donné que cet algorithme implémente l'opération la plus sollicitée et la plus gourmande en CPU, comme mentionnée plus haut, il est souhaitable de l'optimiser de manière à effectuer un seul balayage.

3. Un nouvel algorithme appelé Sgrite

L'algorithme Sgrite est basé sur le principe *Apriori* comme l'algorithme grite. En plus de l'exploitation du lemme sur l'anti-monotone du support comme grite pour élaguer l'espace de recherche, il utilise le lemme sur l'égalité des supports d'un motif et de son complémentaire pour renforcer l'élagage. Son espace de recherche est le treillis au premier terme positif. Sgrite considère uniquement les opérateurs de comparaison stricts (<,>). Pour des raisons d'espace nous avons omis les preuves des lemmes.

3.1. Amélioration du calcul du support d'un motif

Nous proposons dans cette section deux classes d'algorithmes de calcul du support graduel qui éffectuent un seul balayage du graghe de précédence. Ce qui est une amélioration par rapport à l'algorithme récursif de calcul de support graduel utilisé dans Grite[5] qui éffectue deux balayages du graphe de précédence. Ici, Chaque algorithme prend en entrée une matrice d'adjacence, un objet *node* $\in \mathcal{O}$ et un vecteur, appelé *Memory*, de taille $|\mathcal{O}|$ dont les index sont les objets de $\mathcal{O}$. On suppose que *Memory*[o] = -1 $\forall o \in \mathcal{O}$ avant l'exécution de chaque algorithme. Pendant l'exécution de chaque algorithme, *Memory*[o] contient la distance maximale courante entre o et une feuille quelconque. À la fin de l'exécution de chaque algorithme *Memory*[o] contient la distance maximale finale entre o et une feuille quelconque.

Le principe de la première classe d'algorithmes est de mettre à jour les valeurs des parents d'un noeud dès que la valeur de ce noeud est mise à jour. Ce principe peut être appliqué soit en effectuant un parcours en largeur du graphe de précédence, soit en effectuant un parcours en profondeur dudit graphe. La version présentée par l'algorithme 1, donné en annexe, est basée sur un parcours en profondeur et non-récursif (de manière itérative). Cet algorithme calcule la distance maximale entre un noeud donné et une feuille quelconque. L'algorithme commence par affecter la valeur 1 à chaque feuille, ensuite il initialise la pile de noeuds dont on a récemment mise à jour les valeurs à la liste des feuilles. À chaque itération de la boucle on dépile un noeud au sommet de la pile et on se sert de sa valeur pour mettre à jour les valeurs de ses parents. Tout parent dont on a mis à jour la valeur est empilé en sommet de pile. L'algorithme se termine dès que la pile est vide.

Le principe de la deuxième classe d'algorithmes est d'utiliser uniquement la valeur finale d'un noeud pour mettre à jour les valeurs des noeuds de ses parents. Comme le premier principe, le deuxième principe peut être appliqué soit en effectuant un parcours en largeur du graphe de précédence, soit en effectuant un parcours en profondeur dudit graphe. L'algorithme *RecursiveCorvering*, utilisé dans grite, est une implémentation récursive du deuxième principe. L'algorithme 2, donné en annexe, présente la version itérative basée sur un parcours en largeur. Cet algorithme commence par affecter la valeur 1 à chaque feuille, puis il initialise la file des noeuds dont-on connait la distance maximale par rapport à une feuille quelconque à l'ensemble des feuilles. À chaque étape de la boucle, on retire un noeud de la file et on se sert de la valeur dudit noeud pour mettre à jour les valeurs de ses parents. Un parent dont on connait la valeur de tous ses fils est ajouté dans la file. Dès que la file est vide l'algorithme se termine. La version itérative basée sur un parcours en profondeur est obtenue en remplaçant la file dans l'algorihtme 2 par une pile.

Lemme 4. *Véracité des algorithmes : Les algorithmes retournent effectivement la valeur de la distance maximale entre l'argument node et une feuille quelconque du graphe de précédence induit par la matrice d'adjacence* Adj_M.

Les illustrations Fig. 4 , Fig. 5 en annexe présentent la trace d'exécution de *Recursive-Corvering* de grite[5] et sgrite voir Algo. 1 en annexe ; montrent en effet que sgrite utilise moins d'opérations, 15 contre 24 pour grite pour la détermination du support graduel.

3.2. Amélioration du calcul des supports de plusieurs motifs

Étant donnés trois motifs M_1, M_2 et M= M_1M_2, nous montrons dans cette section comment calculer le support graduel de M en exploitant les valeurs de la mémoire *Memory*

obtenues après le calcul du support graduel de M_1. L'approche proposée ici procède en quatre étapes. Avant de présenter ces étapes nous présentons le lemme suivant.

Lemme 5. *Considérons un motif M= M_1M_2. Nous avons les propriétés suivantes : (1.) Tout noeud isolé de M_1 (resp. M_2) est un noeud isolé de M. (2.) Tout noeud racine de M est un noeud racine de M_1 (resp. M_2). (3.) Tout noeud feuille de M_1 (resp. M_2) est soit un noeud isolé de M, soit une feuille de M. (4.) Tout noeud père de M est un noeud père de M_1 (resp. M_2). (5.) Le nombre de fils d'un noeud père de M est inférieur ou égal à son nombre de fils dans M_1. (6.) Tout noeud racine de M_1 (resp. M_2) est soit un noeud isolé de M, soit un noeud racine de M. (7.) Tout noeud interne de M_1 (resp. M_2) est dans M, soit un noeud isolé, soit une feuille, soit un noeud interne, soit une racine.*

Étape 1 : l'objectif de cette étape est de déterminer tous les noeuds de M, qui permettent de retrouver chaque noeud de M dont on doit recalculer la distance maximale par rapport à une feuille quelconque dans M. Ces noeuds sont appelés noeuds générateurs. Nous notons *Générateur(M,M_1)* l'ensemble des noeuds générateurs. Soit *Feuilles(M,M_1)* l'ensemble des noeuds feuilles de M qui ne sont pas des feuilles dans M_1. Il s'agit des noeuds feuilles de M qui sont des noeuds pères dans M_1. Soit *Pères(M,M_1)* l'ensemble des noeuds pères de M dont le nombre de fils dans M_1 est strictement supérieur au nombre de fils dans M. Nous avons *Générateur(M,M_1) = Pères(M,M_1)$\cup$Feuilles(M,M_1)*. Nous notons *Noeuds(M,M_1)* l'ensemble des noeuds de M dont on doit recalculer les valeurs dans M. Nous avons le lemme suivant.

Lemme 6. *Considérons deux motifs M et M_1 tels que $M_1 \subset M$. Pour tout noeud o du graphe de précédence induit par le motif M nous avons : si o$\notin$ Noeuds(M,M_1) alors la distance maximale qui sépare o d'une feuille quelconque dans M est égale à celle trouvée dans M_1.*

Étape 2 : l'objectif de cette étape est d'initialiser à zéro la valeur de chaque noeud dont on doit recalculer la distance maximale par rapport à une feuille quelconque dans M. L'algorithme 3, donné en annexe, permet d'atteindre cet objectif. Il prend en entrée les matrices d'adjacence de M et de M_1, et la mémoire *Memory* obtenue après le calcul du support de M_1. Il commence par initialiser la pile à l'ensemble des noeuds générateurs et l'ensemble courant des noeuds dont on doit affecter la valeur zéro est initialisé à l'ensemble vide. À chaque itération de la boucle la plus externe on dépile un noeud de la pile, on lui affecte la valeur zéro, on l'ajoute à l'ensemble des noeuds dont on doit recalculer la valeur dans M, et on empile tous ses parents. l'algorithme se termine dès que la pile est vide. L'algorithme retourne l'ensemble Noeuds(M,M_1).

Étape 3 : l'objectif de cette étape est d'affecter la valeur finale 1 à chaque feuille de M qui appartient à l'ensemble *Noeuds(M,M_1)* et d'affecter à chaque noeud père de M qui apartient à *Noeuds(M,M_1)* la valeur maximale de ses fils augmentée de 1 comme valeur courante. Ceci permet d'exploiter les valeurs des noeuds dont on connait la valeur finale dans M pour mettre à jour les valeurs de leurs pères qui appartiennent à l'ensemble *Noeuds(M,M_1)*. L'algorithme 4, donné en annexe, permet de réaliser la troisième étape.

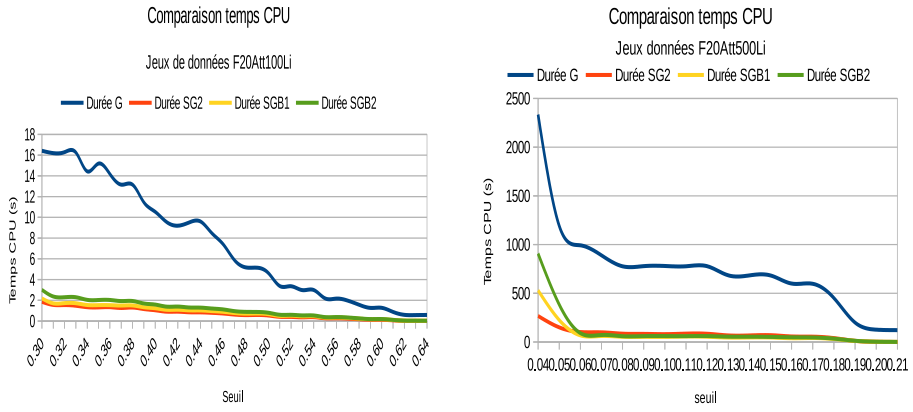
Étape 4 : l'objectif de cette étape est de calculer la valeur finale des noeuds pères de l'ensemble Noeuds(M,M_1). L'algorithme 5, donné en annexe, calcule le support graduel de M en exploitant la mémoire Memory obtenue après le calcul du support de M_1. Il est déduit de l'algorithme 1 en remplaçant l'instruction 1 de cet algorithme par une instruction qui initialise la pile à l'ensemble $Noeuds(M, M_1)$ et en exploitant les résultats des algorithmes 3 et 4 . La mémoire Memory passée en paramètre est celle obtenue après le calcul du support de M_1, c'est-à-dire que $\mathcal{M}emory[o]$ contient la distance maximale

entre o et une feuille quelconque de M_1. À la fin de l'exécution $\mathcal{M}emory[o]$ contient la distance maximale entre o et une feuille quelconque de M.

Lemme 7. *L'algorithme 5, donné en annexe, parcours uniquement les noeuds de l'ensemble* $Noeuds(M, M_1)$ *et les fils des noeuds de l'ensemble Pères*(M, M_1).

4. Expérimentation

Cette section compare expérimentalement les performances de grite et de Sgrite. Nous avons utilisé trois jeux de données. Les deux premiers sont des données de tests appelées *F20Att100Li* ayant 20 attributs, 100 transactions et F20Att500Li ayant 20 attributs, 500 transactions et issues du site`https://github.com/bnegreve/paraminer/tree/master/data/gri`. Le dernier jeu de données constitues des données météorologiques issues du site `http://www.meteo-paris.com/ile-de-france/station-meteo-paris/pro/`. Les expérimentations sont effectuées sur un Intel core i7, 2 Ghz x 8, 8 GB de mémoire vive, sur ubuntu 16.04 LTS. Nous avons considéré grite et trois versions de Sgrite, notamment SG2, SGB2 et SGB1. Dans la version SG2, le calcul du support graduel est basé sur l'algorithme 1. Dans la version SGB2, le calcul du support graduel est basé sur l'algorithme 2. Dans la version SGB1, le calcul du support graduel est basé sur l'algorithme obtenu en remplaçant la file de l'algorithme 2 par une pile afin de remplacer le parcours en largeur par un parcours en profondeur. Pour chaque jeu de données, nous avons exécuté grite et les 3 versions de Sgrite en faissant varier le seuil du support minimum dans l'intervalle [0,1[et en relevant à chaque fois le temps d'exécution. Ces résultats expérimentaux confirment que sgrite est plus performant que grite.



(a) base test, 100 Tr. et 50 It.

(b) base test, 500 Tr. et 50 It.

Figure 2 – Différents temps CPU (Tr. (resp. It.) désigne Transactions (resp. Items))

L'écart entre les temps d'extraction croit à l'avantage de Sgrite quand le seuil de support minimal décroît. Pour des valeurs basses de seuil de support minimal Sgrite est deux à quatre fois plus rapide que grite. Ceci provient de ce que les algorithmes de calcul des supports des motifs graduels sont optimisés dans Sgrite (voir Fig.2a,Fig. 3, Fig. 2b). De

plus, le fait que RecursiveCorvering[5] soit récursive, et Sgrite est une approche ascendante itérative contribue encore à l'optimisation. Le fait, aussi que SG2 soit meilleur que SGB1 et SGB2, excepté pour le jeu de données *F50Att500Li* 2b suggère que les approches bloquantes peuvent s'accélérer pour des jeux de données denses et de grande taille, mais SG21 reste meuilleur(voir Fig. 3 et 2a) en ce qui concerne les jeux moins denses.

5. Conclusion

Nous avons présenté dans ce papier une approche d'amélioration des performances des algorithmes de découverte des motifs graduels en réduisant la charge de calcul liée au calcul des supports graduels. Nous avons montré comment calculer le support d'un motif en effectuant un seul balayage de son graphe de précédence. Nous avons aussi montré comment exploiter les traitements effectués lors du calcul du support graduel d'un motif pour accélérer les calculs des supports de ses sur-motifs par la relation d'inclusion. Les expérimentations effectuées confirment l'intérêt de l'approche proposée dans ce papier.

Références

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM, 1993.

[2] Sarra Ayouni. Etude et extraction de regles graduelles floues : définition dâalgorithmes efficaces. *These de doctorat, Université Montpellier*, 2, 2012.

[3] Kenmogne Edith Belise, Nkambou Roger, Tadmon Calvin, and Engelbert Mephu Nguifo. A parallel pattern-growth algorithm. 2018.

[4] F. Berzal, J. C. Cubero, D. SÃ¡Nchez, M. A. Vila, and J. M. Serrano. An alternative approach to discover gradual dependencies. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 15(5) :559–570, 2007.

[5] Lisa Di Jorio. *Recherche de motifs graduels et application aux données médicales*. PhD thesis, Montpellier 2, 2010.

[6] Lisa Di-Jorio, Anne Laurent, and Maguelonne Teisseire. Mining frequent gradual itemsets from large databases. In *International Symposium on Intelligent Data Analysis*, pages 297–308. Springer, 2009.

[7] Didier Dubois and Henri Prade. Gradual inference rules in approximate reasoning. *Information Sciences*, 61(1-2) :103–122, April 1992.

[8] Edith Belise Kenmogne. In *Algorithmes des motifs séquentiels et temporels : Etat de l'art et étude comparative*. Université de Dschang, 2011.

[9] Edith Belise Kenmogne. *Contribution to the sequential and parallel discovery of sequential patterns with an application to the design of e-learning recommenders*. PhD thesis, Université of Dschang, October 2018.

[10] Anne Laurent, Marie-Jeanne Lesot, and Maria Rifqi. Extraction de motifs graduels par corrélations d'ordres induits. *Rencontres sur la Logique Floue et ses Applications, LFAâ2010*, 2010.

[11] Benjamin Negrevergne, Alexandre Termier, Marie-Christine Rousset, and Jean-François Mehaut. ParaMiner : a Generic Parallel Pattern Mining Algorithm. Research Report RR-LIG-012, 2011.

[12] Amal Oudni. *Fouille de données par extraction de motifs graduels : contextualisation et enrichissement*. PhD thesis, Université Pierre et Marie Curie-Paris VI, 2014.

6. Annexe

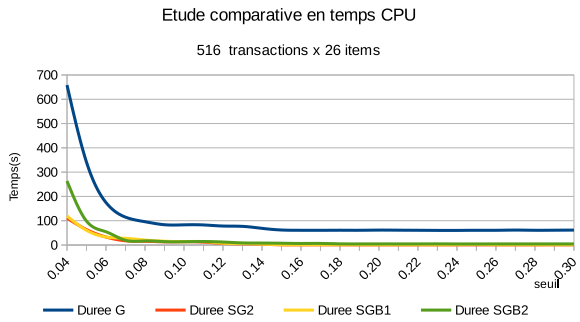


Figure 3 – Comparaison des temps d'exécution sur des données météorologiques constituées de 516 transactions et de 26 items

Algorithm 1 Algorithme ComputeSupportSGriteBootumToUp.

Require: $\mathcal{A}dj_m$; $\mathcal{M}emory$
Ensure: $support$;
1: $stack \leftarrow \mathcal{LF}s \leftarrow getLeafs(\mathcal{A}dj_m)$;
2: initializeLeafPosirionToOne($\mathcal{M}emory$) ;
3: $e \leftarrow pop(stack)$;
4: **while** $stack \neq \emptyset$ **do**
5: $\mathcal{H}_p$ = parents($\mathcal{A}dj_m$, e) ;
6: **for all** $p \in \mathcal{H}_p$ **do**
7: **if** $\mathcal{M}emory[p] < \mathcal{M}emory[e] + 1$ **then**
8: $\mathcal{M}emory[p] \leftarrow \mathcal{M}emory[e] + 1$;
9: $push(stack, p)$;
10: **end if**
11: **end for**
12: $e \leftarrow pop(stack)$;
13: **end while**
14: **return** $\mathcal{M}emory$;

Algorithm 2 Algorithme ComputeSupportBlockingSGrite2

Require: $\mathcal{A}dj_m$; $\mathcal{M}emory$
Ensure: $support$;
1: $\mathcal{LF}s \leftarrow getAllLeafs(\mathcal{A}dj_m)$;
2: initializeLeafValuesToOne($\mathcal{M}emory, \mathcal{LF}s$) ;
3: $queueOfKnowMaxDistNodes \leftarrow \mathcal{LF}s$
4: $e \leftarrow getAndRemoveFirstElement(queueOfKnowMaxDistNodes)$;
5: **while** $queueOfKnowMaxDistNodes \neq \emptyset$ **do**
6: $parentSet$ = setOfParentsOfANode($\mathcal{A}dj_m$, e) ;
7: **for all** $p \in parentSet$ **do**
8: **if** $p \notin getTheKeysSetOfAMap(mapNodeToSons)$ **then**
9: $addAnEntryToAMap(mapNodeToSons, p, getSons(p))$;
10: **end if**
11: $removeAnElementFromASet(e, get(p, mapNodeToSons))$;
12: **if** $\mathcal{M}emory[p] < \mathcal{M}emory[e] + 1$ **then**
13: $\mathcal{M}emory[p] \leftarrow \mathcal{M}emory[e] + 1$;
14: **if** $get(p, mapNodeToSons) = \emptyset$ **then**
15: $addAnElement(setOfKnowMaxDistNodes, p)$;
16: **end if**
17: **end if**
18: **end for**
19: $e \leftarrow getAndRemoveFirstElement(queueOfKnowMaxDistNodes)$;
20: **end while**
21: **return** $\max(\mathcal{M}emory)$;

Algorithm 3 InitializeToZero.

Require: $\mathcal{A}dj_{M_1}$; $\mathcal{A}dj_M$; $\mathcal{M}emory$
Ensure: $updateSet$;
1: $updateSet \leftarrow \emptyset$;
2: $stack \leftarrow$ Générateur(M,M_1) ;
3: **while** $stack \neq \emptyset$ **do**
4: $node \leftarrow pop(stack)$;
5: $addToSet(updateSet, node)$;
6: $\mathcal{M}emory[node] = 0$;
7: $\mathcal{H}_p$ = parents($\mathcal{A}dj_M$, node) ;
8: **for all** $p \in \mathcal{H}_p$ **do**
9: $push(stack, p)$;
10: **end for**
11: **end while**
12: **return** $updateSet$;

Algorithm 4 InitFromSons.

Require: $Feuille(M, M_1)$; $\mathcal{A}dj_M$; $\mathcal{M}emory$;$Noeuds(M, M_1)$
Ensure: $Memory$;
1: $internalNode(M, M_1) \leftarrow Noeuds(M, M_1) - Feuille(M, M_1)$;
2: **for all** $node \in Feuille(M, M_1)$ **do**
3: $\mathcal{M}emory[node] \leftarrow 1$;
4: **end for**
5: **for all** $node \in internalNode(M, M_1)$ **do**
6: $\mathcal{M}emory[node] = \max(getSons(node)) + 1$;
7: **end for**
8: **return** $Memory$;

Algorithm 5 Algorithme ComputeSupportSGrite

Require: $Feuille(M, M_1); \mathcal{A}dj_{M_1}; \mathcal{A}dj_M; \mathcal{M}emory$
Ensure: $support$;
1: $Noeuds(M, M_1) \leftarrow InitializeToZero(\mathcal{A}dj_{M_1}, \mathcal{A}dj_M, \mathcal{M}emory)$;
2: $stack \leftarrow Noeuds(M, M_1)$;
3: $\mathcal{M}emory \leftarrow InitFromSons(Feuille(M, M_1), \mathcal{A}dj_M, \mathcal{M}emory, Noeuds(M, M_1))$;
4: $e \leftarrow pop(stack)$;
5: **while** $stack \neq \emptyset$ **do**
6: $\mathcal{H}_p$ = parents($\mathcal{A}dj_m$, e);
7: **for all** $p \in \mathcal{H}_p$ **do**
8: **if** $\mathcal{M}emory[p] < \mathcal{M}emory[e] + 1$ **then**
9: $\mathcal{M}emory[p] \leftarrow \mathcal{M}emory[e] + 1$;
10: $push(stack, p)$;
11: **end if**
12: **end for**
13: $e \leftarrow pop(stack)$;
14: **end while**
15: **return** $\mathcal{M}emory$;

Opérations	O1	O2	O3	O4	O5	O6	O7	O8
0	Initialisation de Memory à -1 et empiler O7 et O8							
1	-1	-1	-1	-1	-1	-1	**1**	**1**
2	Depiler O8 et parents(O8)={**O6**}							
3	-1	-1	-1	-1	-1	**2**	1	1
4	Depiler O6 et parents(O6)={**O4**}							
5	-1	-1	-1	**3**	-1	2	1	1
6	Depiler O4 et parents(O4)={**O5**}							
7	-1	-1	-1	3	**4**	2	1	1

Opérations	O1	O2	O3	O4	O5	O6	O7	O8
8	Depiler O5 et parents(O5)={**O2**}							
9	-1	**5**	-1	3	4	2	1	1
10	-1	**5**	-1	3	4	2	1	1
11	Depiler O2 et parents(O2)={**O1,O3**}, Depiler O3							
12	-1	5	**6**	3	4	2	1	1
13	Depiler O1 et parents(O1)={**}**							
14	**6**	5	6	3	4	2	1	1
15	**Depiler O7, la pile est vide**							

Figure 4 – Trâce de sgrite voir Algo. 1 pour le motif $A^{>}S^{>}$ et le graphe Fig. 1

Opérations	O1	O2	O3	O4	O5	O6	O7	O8
0	Initialisation de Memory à -1							
1	-1	-1	-1	-1	-1	-1	-1	-1
2	Appel 1 sur O1							
3	Appel 2 sur O2							
4	Appel 3 sur O5							
5	Appel 4 sur O4							
6	Appel 5 sur O6 puis sur O7							
7	Appel 6 sur O8							
8	-1	-1	-1	-1	-1	-1	-1	**1**
9	retour sur appel 6							
10	-1	-1	-1	-1	-1	**2**	-1	**1**
11	retour sur appel 5 sur O6							
12	Appel 7 sur O7							
13	-1	-1	-1	-1	-1	2	**1**	1

Opérations	O1	O2	O3	O4	O5	O6	O7	O8
14	retour sur appel 4 sur O4							
15	-1	-1	-1	**3**	-1	2	1	1
16	retour sur appel 3 sur O5							
17	-1	-1	-1	3	**4**	2	1	1
18	retour sur appel 2 sur O2							
19	-1	**5**	-1	3	4	2	1	1
20	retour sur appel 1 sur O1							
21	**6**	5	-1	3	4	2	1	1
22	Appel 8 sur O3							
23	6	5	**6**	3	4	2	1	1
24	retour sur appel 8							

Figure 5 – Trâce de RecursiveCorvering pour le motif $A^{>}S^{>}$ et le graphe Fig. 1