Extraction de motifs fréquents dans des arbres attribués

Claude Pasquier*,**, Jérémy Sanhes*
Frédéric Flouvat*, Nazha Selmaoui-Folcher*

*Université de Nouvelle Calédonie
PPME, BP R4, F-98851 Nouméa, Nouvelle Caledonie
{jeremy.sanhes,frederic.flouvat, nazha.selmaoui}@univ-nc.nc,
http://ppme.univ-nc.nc
**Institut de Biologie Valrose (IBV)
UNS - CNRS UMR7277 - INSERM U1091, F-06108 Nice cedex 2
claude.pasquier@unice.fr
http://ibv.unice.fr

Résumé. L'extraction de motifs fréquents est une tâche importante en fouille de données. Initialement centrés sur la découverte d'ensembles d'items fréquents, les premiers travaux ont été étendus pour extraire des motifs structurels comme des séquences, des arbres ou des graphes. Dans cet article, nous proposons une nouvelle méthode de fouille de données qui consiste à extraire de nouveaux types de motifs à partir d'une collection d'arbres attribués. Les arbres attribués sont des arbres dans lesquels les nœuds sont associés à des ensembles d'attributs. L'extraction de ces motifs (appelés sous-arbres attribués) combine une recherche d'ensembles d'items fréquents à une recherche de sous-arbres et nécessite d'explorer un immense espace de recherche. Nous présentons plusieurs nouveaux algorithmes d'extraction d'arbres attribués et montrons que leurs implémentations peuvent efficacement extraire des motifs fréquents à partir de grands jeux de données.

1 Introduction

L'extraction de motifs fréquents est une tâche importante dans le domaine de la fouille de données. Initialement centrée sur la découverte d'ensemble d'items (itemsets) fréquents (Agrawal et al., 1993), les premiers travaux ont été étendus pour extraire des motifs structurels comme les séquences (Agrawal et Srikant, 1995), les arbres (Chi et al., 2004a) ou les graphes (Washio et Motoda, 2003).

Alors que l'extraction d'itemsets fréquents recherche les combinaisons fréquentes d'items, l'extraction de motifs structurels recherche des sous-structures fréquentes. La plupart des travaux existants se focalisent sur un seul type de problème (fouille d'itemsets ou fouille structurelle). Toutefois, afin de représenter des données plus complexes, il semble naturel de considérer des collections structurées d'itemsets. Dans cet article, nous introduisons le problème de fouille d'arbres attribués. Les arbres attribués sont des arbres dans lesquels les nœuds sont associés à des itemsets.

Les arbres attribués peuvent être utilisés dans de nombreuses applications de fouilles de données spatio-temporelles. Dans le cas d'études épidémiologiques, par exemple, l'espace géographique peut être découpé en zones qui sont représentées par les nœuds de l'arbre, les itemsets décrivent les caractéristiques de ces zones à un temps donné et les arêtes symbolisent des relations de voisinage avec d'autres zones au temps suivant. Les motifs fréquent trouvés sont ainsi susceptible de donner un éclairage nouveau sur les déterminants d'une pathologie. D'autres applications peuvent être imaginées dans divers domaines comme l'analyse des arbres de retweet, la fouille d'arbres phylogénétiques, la fouille de documents XML ou l'analyse de journaux d'activité Web (Web log analysis).

Les contributions clés de notre travail sont les suivantes : 1) nous présentons le problème de la fouille de sous-structures ordonnées et non ordonnées dans une collection d'arbres attribués, 2) nous définissons les formes canoniques des arbres attribués, 3) nous proposons une méthode permettant l'énumération des arbres attribués qui est basée sur la combinaison de deux opérations : l'extension de la structure arborescente et l'extension des itemsets associés aux nœuds, 4) nous présentons trois variantes d'un algorithme permettant d'extraire les motifs fréquents dans des arbres attribués, 5) nous montrons les résultats expérimentaux effectués sur plusieurs jeux de données artificiels et un jeu de données réel.

Cet article est organisé de la manière suivante : la section 2 présente les concepts de base et définit le problème, la section 3 propose un bref aperçu de l'état de l'art et met l'accent sur les quelques études qui combinent la fouille d'itemsets et la fouille structurelle, la section 4 décrit la méthode en insistant sur l'exploration de l'espace de recherche, le calcul des fréquences et l'élagage des candidats, la section 5 montre les résultats de la fouille de plusieurs jeux de données artificiels et réels et finalement, la section 6 conclut l'article et présente de possibles extensions de notre travail.

2 Concepts généraux et définition du problème

Dans cette section, nous introduisons les concepts et définitions nécessaires et présentons le problème de fouille d'arbres attribués.

2.1 Préliminaires

Soit $\mathcal{I} = \{i_1, i_2, .., i_n\}$ un ensemble d'items. Un **itemset** est un ensemble $\mathcal{P} \subseteq \mathcal{I}$. Les items appartenant à un itemset sont triés selon l'ordre lexicographique. L'ensemble $\mathcal{D}$ des itemsets présents dans une base de données est noté $\{\mathcal{P}1, \mathcal{P}2, ..., \mathcal{P}m\}$ avec $\forall \mathcal{P} \in \mathcal{D}, \mathcal{P} \subseteq \mathcal{I}$. $\mathcal{D}$ est une base de données de transactions.

Un **arbre** $S = (V, E)$ est un graphe orienté acyclique et connecté dans lequel V est l'ensemble des nœuds et $E = \{(u, v) | u, v \in V\}$ est l'ensemble des arêtes. Un nœud particulier $r \in V$ est considéré comme étant la racine, et pour chacun des autres nœuds $x \in V$, il existe un unique chemin allant de r à x. S'il existe un chemin allant d'un nœud u à un nœud v dans $S = (V, E)$, alors u est un *ancêtre* de v (v est un *descendant* de u). Si $(u, v) \in E$ (i.e. u est un ancêtre direct de v), alors u est un *parent* de v (v est un *enfant* de u). Dans un arbre ordonné, les fils de chaque nœuds sont ordonnés, sinon, l'arbre est non ordonné. Dans cette article, sauf spécification contraire, nous considérons que les arbres sont non ordonnés.

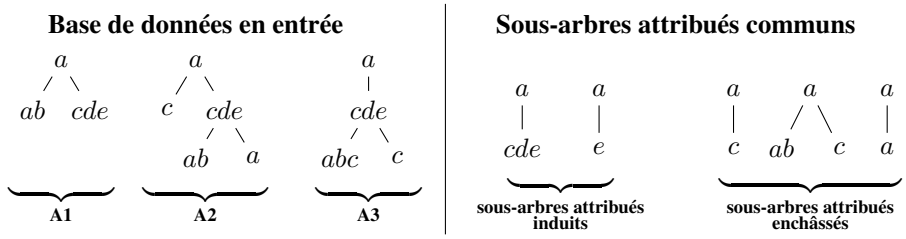

FIG. 1 – *Exemple d'une base de données d'arbres attribués avec quelques motifs communs.*

Un **arbre attribué** est un triplet $T = (V, E, \lambda)$ où (V, E) représente l'arbre sous-jacent et $\lambda : V \rightarrow \mathcal{D}$ est une fonction qui associe un itemset $\lambda(u) \in \mathcal{I}$ à chaque nœud $u \in V$. La taille d'un arbre attribué est le nombre des items associés à l'ensemble des nœuds.

Dans cet article, nous utilisons une représentation textuelle d'un arbre attribué basée sur celle définie par Zaki (2002) pour les arbres étiquetés. Notre représentation se distingue par l'écriture des nœuds qui est générée en listant tous les items présents dans l'itemset associé et par le fait que, par simplicité, nous omettons les \$s finaux. Par exemple, la représentation textuelle de l'arbre attribué $A2$ illustré dans la figure 1 est "$a\ c\ \$\ cde\ ab\ \$\ a$".

Les arbres attribués peuvent être vus comme des itemsets organisés selon une structure arborescente. L'**inclusion sur les arbres attribués** peut donc être définie en considérant soit l'inclusion d'itemsets, soit l'inclusion structurelle. Pour l'inclusion portant sur les itemsets, on considère que l'arbre attribué T_1 est **contenu dans** un autre arbre attribué T_2 si les deux arbres attribués ont la même structure et que, pour chaque nœud de T_1, l'itemset qui lui est associé est inclus dans l'itemset du nœud correspondant de T_2. Plus formellement, $T_1 = (V_1, E_1, \lambda_1)$ est contenu dans $T_2 = (V_2, E_2, \lambda_2)$, et on note $T_1 \sqsubset_I T_2$, *si* $V_1 = V_2$ et $E_1 = E_2$ et $\forall x \in V_1, \lambda_1(x) \subseteq \lambda_2(x)$. L'inclusion structurelle est quant à elle représentée par le concept classique de sous-arbre (Balcázar et al., 2010; Chi et al., 2004a; Hido et Kawano, 2005; Nijssen et Kok, 2003; Termier et al., 2004; Xiao et al., 2003; Zaki, 2002).

A partir de la définition précédente, nous généralisons la notion de sous-arbre attribué de la façon suivante. $T_1 = (V_1, E_1, \lambda_1)$ est un **sous-arbre attribué** d'un arbre attribué $T_2 = (V_2, E_2, \lambda_2)$ et on note $T_1 \sqsubset T_2$ *si* T_1 est un sous-arbre attribué isomorphe de T_2, i.e. il existe une correspondance $\varphi : V_1 \rightarrow V_2$ telle que $T_1 \neq T_2$ et $(u, v) \in E_1$ si $(\varphi(u), \varphi(v)) \in E_2$ et $\forall x \in V_1, \lambda_1(x) \subseteq \lambda_2(\varphi(x))$. Si T_1 est un sous-arbre attribué de T_2, on dit que T_2 est un super-arbre attribué de T_1. T_1 est un **sous-arbre attribué induit** de T_2 *ssi* T_1 est un sous-arbre attribué isomorphe de T_2 et φ préserve la relation parent-enfant. T_1 est un **sous-arbre attribué enchâssé** de T_2 *ssi* T_1 est un sous-arbre attribué isomorphe de T_2 et φ préserve la relation ancêtre-descendant. La figure 1 montre un exemple d'une base de données d'arbres attribués composée de trois arbres attribués différents ainsi que deux ensembles de motifs communs ; l'un contenant des arbres attribués induits et l'autre des arbres attribués enchâssés.

Tous les algorithmes de fouille d'arbres travaillant avec des arbres non-ordonnés doivent prendre en compte le problème d'isomorphisme. Pour éviter la génération redondante de solutions équivalentes, un arbre est choisi en tant que forme canonique et les formes alternatives sont écartées (Asai et al., 2002; Chi et al., 2004b; Nijssen et Kok, 2003; Xiao et al., 2003; Zaki, 2004). Dans de précédents travaux, les formes canoniques sont basées sur l'ordre lexi-

cographique des étiquettes des nœuds. Dans notre travail, nous définissons un ordre basé sur l'ordre des itemsets associés aux nœuds. Etant donné deux itemsets $\mathcal{P}$ et $\mathcal{Q}$ ($\mathcal{P} \neq \mathcal{Q}$), on dit que $\mathcal{P} < \mathcal{Q}$ ssi 1) $\forall i \in [1, \min(|\mathcal{P}|, |\mathcal{Q}|)] : \mathcal{P}_i \leq \mathcal{Q}_i$ et 2) si $\forall i \in [1, \min(|\mathcal{P}|, |\mathcal{Q}|)] : \mathcal{P}_i = \mathcal{Q}_i$, alors $|\mathcal{P}| > |\mathcal{Q}|$. De la définition précédente, un ordre, $\prec$, portant sur les arbres attribués peut être défini. A partir de cet ordre, une **forme canonique pour les arbres attribués isomorphes** est facilement définie en utilisant la méthode présentée par Chi et al. (2004a).

Le problème de la fouille d'arbres attribués est que le nombre de motifs fréquents est souvent très important. Dans des applications réelles, générer toutes les solutions peut s'avérer très coûteux ou même impossible. De plus, un nombre important de solutions contient des informations redondantes. Dans la figure 1, par exemple, l'arbre attribué "$a\ e$" est présent dans toutes les transactions mais le motif est déjà encodé dans "$a\ cde$".

Depuis la proposition de Mannila et Toivonen (2005) d'importants efforts on été consacrés à l'élaboration de représentations condensées qui résument toutes les solutions dans un ensemble restreint. L'ensemble des motifs fermés est un exemple d'une telle représentation condensée (Pasquier et al., 1999). Un arbre attribué T est un **arbre attribué fermé** si aucun de ses super-arbres attribués ne possède le même support que lui. Dans cet article, nous introduisons une autre représentation résumée des motifs qui est définie uniquement en fonction de la relation **contenu dans**. On dit que T est un **arbre attribué c-fermé** (contenu fermé) s'il n'est pas contenu (comme défini précédemment) dans un autre arbre attribué avec le même support.

2.2 Définition du problème

Soit une base de données $\mathcal{B}$ d'arbres attribués et un arbre attribué T, la **fréquence par transaction** de T est représentée par le nombre d'arbres attribués dans $\mathcal{B}$ pour lesquels T est un sous-arbre attribué. Un arbre attribué est fréquent si sa fréquence par transaction est supérieure ou égale à un seuil minimum. Le problème consiste à énumérer tous les motifs fréquents dans un ensemble d'arbres attribués.

3 Etat de l'art

Les premiers algorithmes de fouille d'arbres étiquetés sont dérivés de l'approche Apriori (Agrawal et al., 1993). Ceux-ci consistent en une succession d'itérations comprenant une génération des candidats suivie d'un calcul de fréquences à l'issue duquel les motifs non fréquents sont écartés. Deux stratégies sont possibles pour la génération des candidats : extension et jointure. Avec l'extension, un nouveau candidat est généré en ajoutant un nœud à un arbre fréquent (Asai et al., 2002; Nijssen et Kok, 2003). Avec la jointure, un nouveau candidat est créé en combinant deux arbres fréquents (Hido et Kawano, 2005; Zaki, 2004). La combinaison de ces deux principes a également été étudiée (Chi et al., 2004b).

D'autres algorithmes de fouille d'arbres sont dérivés de l'approche FP-growth (Han et al., 2000). Ces algorithmes, qui adoptent le principe de **pattern-growth**, permettent d'éviter le coûteux processus de génération de candidats. Cependant, cette approche ne peut pas être adaptée simplement au problème de la fouille d'arbres. Les implémentations existantes sont limitées dans le type d'arbres qu'elles peuvent manipuler (Xiao et al., 2003; Wang et al., 2004).

Trouver des représentations condensées des motifs fréquents est une extension naturelle de la fouille de motifs. Pour la fouille d'itemsets, la notion de fermeture a été formellement définie

par Pasquier et al. (1999). Plusieurs travaux ont exploré ce thème dans le contexte de la fouille d'arbres et ont proposé des méthodes de fouille adaptées ainsi que diverses implémentations (Chi et al., 2004c; Termier et al., 2004, 2008). A notre connaissance, aucune méthode n'a été proposée dans le cadre général des arbres attribués.

Récemment, nous avons vu un intérêt croissant pour la fouille d'itemsets organisés selon une certaine structure. Miyoshi et al. (2009) travaillent sur des graphes étiquetés avec des attributs quantitatifs associés aux nœuds. Ce type de structure permet de résoudre le problème en combinant un algorithme "classique" d'exploration de sous-graphe pour le graphe étiqueté, et un algorithme existant de fouille d'itemsets pour les attributs quantitatifs. Cette approche ne peut pas être utilisée si les attributs associés aux nœuds ont tous la même importance. Fukuzaki et al. (2010) étudient des graphes dans lesquels les nœuds sont associés à des itemsets. Notre travail diffère de cette étude dans le sens où nous recherchons des motifs dans lesquels les itemsets associés aux nœuds ne sont pas forcément identiques.

4 Fouille d'arbres attribués fréquents

Nous nous intéressons principalement à l'identification de sous-arbres attribués induits, ordonnés ou non. Bien que l'on se focalise sur les sous-arbres attribués induits, nous définissons une méthode générale qui est capable d'extraire également les sous-arbres attribués enchâssés.

4.1 Énumération des arbres attribués

En utilisant l'opérateur $\prec$, il est possible de construire un arbre des candidats Q représentant l'ensemble de l'espace de recherche (Ayres et al., 2002) de la manière suivante. La racine de Q est étiquetée avec $\emptyset$. Récursivement, pour chaque nœud terminal $n \in Q$, des fils n' sont ajoutés tel que $n \prec n'$. Les fils d'un nœud $n \in Q$ sont générés soit par extension d'arbre, soit par extension d'itemset.

4.1.1 Extension de la structure arborescente

Pour l'extension de la structure arborescente (**extension d'arbre**), nous utilisons une variante de la technique connue sous le nom de **rightmost path extension** (Asai et al., 2002; Nijssen et Kok, 2003). Un arbre attribué T peut être étendu pour générer de nouveaux arbres attribués de deux façons différentes. Dans le premier cas, un nouveau fils N est ajouté au nœud terminal situé le plus à droite de T. Dans le second cas, un nouveau frère N est ajouté à l'un des nœuds situés sur le chemin le plus à droite de T (Chehreghani, 2011). Dans l'approche classique, N représente n'importe quel nœud valide figurant dans la base de données. Dans notre approche, de nouveaux nœuds N sont créés à partir des nœuds valides Q de la base de données. Dans les faits, chaque nœud Q, associé à un itemset de taille k, génère un ensemble de k nœuds $\mathcal{N} = \{N_1, .., N_k\}$ qui sont utilisés pour l'extension d'arbre. Chaque N_i est associé à un itemset de taille 1 ; l'unique item étant le $i$ème item de $\lambda(Q)$. Par exemple, dans la figure 1, les nœuds qui peuvent être utilisés pour l'extension du motif "$a\ cde$" sont "ab", "a" (de $A2$), "abc" et "c" (de $A3$). A partir du nœud "abc", trois extensions sont générées ("a", "b" et "c") alors que le nœud "ab" génère "a" et "b". Les nœuds "a" et "c" génèrent respec-

tivement les extensions "a" et "c". Trois nouveaux candidats différents sont ainsi obtenus en ajoutant chacune de ces extensions au motif candidat : "$a\ cde\ a$", "$a\ cde\ b$" et "$a\ cde\ c$"

Pour les arbres ordonnés, la complétude et la non-redondance de cette méthode a été démontrée (Asai et al., 2002). Pour les arbres non ordonnés, la méthode peut générer des motifs redondants sous la formes d'arbres isomorphes. Les candidats dupliqués sont détectés et écartés avant la phase d'extension au moyen d'un test de canonicité.

4.1.2 Extension des itemsets associés aux nœuds

Pour l'extension des itemsets associés aux nœuds (**extension d'itemset**), nous utilisons une variante de la méthode présentée par Ayres et al. (2002). Grâce à cette variante, un nouvel item I est ajouté à l'itemset associé au nœud terminal le plus à droite de l'arbre attribué candidat T. Les items utilisés pour l'extension d'itemset sont dérivés de l'itemset associé à ce nœud dans la base de données. La contrainte est que le nouvel item doit être lexicographiquement situé après n'importe lequel des items associés au nœud terminal le plus à droite de T.

4.2 Calcul des fréquences

Nous organisons nos données dans une structure stockant toute l'information nécessaire au processus de fouille. Notre structure est une extension de la représentation verticale introduite par Zaki (2002, 2004). Brièvement, chaque sous-arbre attribué candidat est associé à son motif et à un ensemble de données annexes permettant de localiser précisément toutes ses occurrences dans la base de données. Les premiers candidats, composés d'un nœud unique associé à un seul item, sont générés en parcourant la base de données. En utilisant uniquement cette structure, il est facile de calculer le nombre d'occurrences de chaque motif. De plus, cette même structure est suffisante pour générer toutes les extensions possibles d'un motif donné. Quand un motif de taille k est traité, toutes ses occurrences sont étendues par les méthodes d'extension d'arbre et d'extension d'itemset décrites plus haut pour générer de nouveaux candidats de taille $(k+1)$ qui sont eux même stockés dans la structure de données.

4.3 Parcours de l'espace de recherche

Plusieurs techniques peuvent être utilisées pour élaguer l'espace de recherche.

4.3.1 Élagage des candidats

Le principe, énoncé par Agrawal et al. (1993) il y a une vingtaine d'années, peut être appliqué à la fouille d'arbres attribués : i) tout sous-motif d'un motif fréquent est fréquent, et ii) tout super-motif d'un motif non fréquent est non fréquent. Comme la fréquence suit une fonction anti-monotone (l'extension d'un motif ne peut pas donner un nouveau motif avec une fréquence supérieure), il est possible d'arrêter l'exploration d'une branche lorsque la fréquence d'un candidat est inférieure au support minimum. Par exemple, dans la figure 1, durant la fouille des arbres attribués, lorsque l'on détermine que la fréquence du motif "$a\ c\ a$" est inférieure au support minimum, il n'est pas nécessaire de générer des extensions de ce motif (e.g. "$a\ c\ ab$", "$a\ c\ a\ \$\ b$", "$a\ c\ a\ \$\ \$\ c$"). Dans le cas de la fouille d'arbres attribués non ordonnées, l'extension est également stoppée si le candidat examiné n'est pas sous forme canonique.

4.3.2 Énumération des arbres attribués c-fermés

En énumérant uniquement les arbres attribués qui ne sont pas **contenus** dans un autre arbre attribué ayant le même support, l'espace de recherche peut être considérablement réduit. L'énumération d'arbres attribués c-fermés nécessite de stocker tous les motifs fréquents identifiés ainsi que leur fréquence par transaction et leur nombre total d'occurrences.

Soit T un arbre attribué candidat, $\mathcal{T}$ l'ensemble de tous les sous-arbres attribués fréquents identifiés précédemment et $\mathcal{X}$ l'ensemble de tous les candidats générés par l'extension de T. Nous distinguons deux sous ensembles de $\mathcal{X}$: $\mathcal{X}_I$, l'ensemble des motifs générés par extension d'itemset de T et $\mathcal{X}_T$, l'ensemble motifs obtenus par extension d'arbre de T. Nous définissons deux fonctions : f_t qui donne la fréquence par transaction d'un sous-arbre attribué et f_o qui retourne son nombre total d'occurrences. T est un arbre attribué c-fermé si $\nexists T' \in \mathcal{T} U \mathcal{X}_I$ tel que $T \sqsubset_I T'$ et $f_t(T') = f_t(T)$. Cependant, identifier une extension d'itemset de T avec la même fréquence par transaction que T ne permet pas de stopper l'exploration des autres candidats de $\mathcal{X}$. La condition supplémentaire suivante doit également être satisfaite : $\exists T' \in \mathcal{T} U \mathcal{X}_I : T \sqsubset_I T'$ et $f_o(T') = f_o(T)$.

Dans la figure 1, par exemple, le premier candidat à être examiné est "a" avec une fréquence par transaction de 3. Par extension d'itemset, nous construisons $\mathcal{X}_I = \{$"ab", "ac"$\}$. Le candidat "ab" a une fréquence par transaction de 3, donc, comme "a" $\sqsubset_I$ "ab", "a" n'est pas c-fermé. Cependant, le motif "a" apparaît 7 fois dans la base de données alors que la fréquence d'occurrence du candidat "ab" est 3. Les 4 occurrences où "a" apparaît dans un itemset qui ne contient pas "b" peuvent éventuellement générer des motifs qui eux sont c-fermés. C'est effectivement le cas dans la figure 1 où l'extension du nœud droit du motif "a" génère le candidat "$a\ e$" avec un fréquence par transaction de 3.

4.3.3 Énumération d'arbres attribués fermés

On dit que T est un arbre attribué fermé si $\nexists T' \in \mathcal{T} U \mathcal{X}$ tel que $T \sqsubset T'$ et $f_t(T') = f_t(T)$. L'extension de T peut être stoppée si $\exists T' \in \mathcal{T} U \mathcal{X} : T \sqsubset T'$ et $f_o(T') = f_o(T)$. Il faut également supprimer les arbres attribués non fermés stockés dans $\mathcal{T}$, i.e. tous les motifs qui sont des sous-arbres attribués de T avec la même fréquence par transaction. Le test de fermeture nécessite d'effectuer plusieurs tests d'isomorphisme de sous-arbres qui sont des opérations coûteuses.

4.4 Description des algorithmes

La figure 2 illustre la structure de l'algorithme IMIT. Pour commencer, un ensemble de sous-arbres attribués de taille 1 est construit en parcourant la base de données (ligne 1). Chaque candidat est alors traité dans une boucle. La fonction $GetFirst$ retourne le plus petit candidat par rapport à l'opérateur $\prec$ (ligne 3). Le candidat traité est supprimé de la liste des candidats (ligne 4). S'il est fréquent et s'il est sous forme canonique (ligne 5), il est ajouté à la liste des solutions (ligne 6) et toutes ses extensions sont ajoutées à la liste des candidats (ligne 7).

Cet algorithme, qui implémente la méthode d'élagage proposée dans la section 4.3.1 est suffisant pour énumérer toutes les solutions mais son espace de recherche est immense. Pour limiter les redondances dans l'ensemble des solutions, nous avons développé un algorithme d'extraction des sous-arbres attribués fermés appelé IMIT_CLOSED reprenant les principes

$IMIT(\mathcal{D}, minSup)$
1: $\mathcal{C} \leftarrow \{all\ asubtrees\ of\ size\ 1\ in\ \mathcal{D}\}$
2: **while** $\mathcal{C} \neq \emptyset$ **do**
3: $T \leftarrow getFirst(\mathcal{C})$
4: $\mathcal{C} \leftarrow \mathcal{C} \setminus \{T\}$
5: **if** $isCanonical(T)$ **and** $f_t(T) \geq minSup$ **then**
6: $\mathcal{T} \leftarrow \mathcal{T} \cup \{T\}$
7: $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{X}$
8: **end if**
9: **end while**
10: $printSolutions(\mathcal{T})$

FIG. 2 – *L'algorithme IMIT.*

exposés dans la section 4.3.3 (figure 3). Comme cela est montré dans la section 5, l'algorithme est coûteux et ne passe pas à l'échelle.

Forts de cette constatation, nous avons développé IMIT_CONTENT_CLOSED, un algorithme permettant l'extraction des sous-arbres attribués c-fermés dont le principe est présenté dans la section 4.3.2. Faute de place, cet algorithme n'est pas présenté ici. Cependant, il peut être facilement déduit de l'algorithme IMIT_CLOSED (figure 3) en remplaçant $\sqsubset$ par $\sqsubset_I$ et en supprimant les lignes 12 à 14. L'utilisation de la relation $\sqsubset_I$ à la place de $\sqsubset$ permet de se limiter aux tests d'inclusion d'itemsets qui sont beaucoup moins coûteux que les tests d'isomorphismes. Les lignes 12 à 14 suppriment de l'ensemble des solution trouvées précédemment celles qui contiennent un sous-arbre attribué du candidat actuellement examiné. Ce test n'est pas nécessaire dans le cas de l'extraction de motifs c-fermés. Les expérimentations montrent que ce troisième algorithme constitue le meilleur compromis entre la non redondance des solutions et le temps d'exécution.

5 Résultats experimentaux

Tous les algorithmes sont implémentés en C++ avec la STL. Les expérimentations ont été effectuées sur un ordinateur avec Ubuntu 12.04 LTS basé sur un processeur Intel©Core™i5-2400 @ 3.10GHz avec 8 Gb de mémoire. Tous les temps d'exécution incluent la phase de prétraitement et l'affichage des résultats.

5.1 Jeux de données artificiels

Nous avons modifié le programme proposé par Zaki (2002) pour générer des arbres attribués avec différentes tailles d'itemsets. Nous avons utilisé les mêmes paramètres que Zaki (2002) sauf pour le nombre de sous-arbres que nous avons fixé à 10000. Nous avons construit cinq jeux de données en faisant varier la taille des itemsets. Dans T10K, tous les nœuds sont associés à des itemsets de taille 1. Dans T10K-3 et T10K-5, les nœuds sont associés respectivement à des itemsets de taille 3 et 5. Dans T10K-1/10, les nœuds sont associés à des itemsets de taille variant de 1 à 10 alors que dans T10K-1/20, la taille des itemsets varie de 1 à 20.

$IMIT\_CLOSED(\mathcal{D}, minSup)$
1: $\mathcal{C} \leftarrow \{all\ asubtrees\ of\ size\ 1\ in\ \mathcal{D}\}$
2: **while** $\mathcal{C} \neq \emptyset$ **do**
3: $T \leftarrow getFirst(\mathcal{C})$
4: $\mathcal{C} \leftarrow \mathcal{C} \setminus \{T\}$
5: **if** $isCanonical(T)$ **and** $f_t(T) \geq minSup$ **then**
6: **if** $\nexists T' \in \mathcal{T} : T \sqsubset T'\ and\ f_t(T') = f_t(T)$ **and** $\nexists T' \in \mathcal{X}_I : f_t(x) = f_t(T)$ **then**
7: $\mathcal{T} \leftarrow \mathcal{T} \cup \{T\}$
8: **end if**
9: **if** $\nexists T' \in \mathcal{T} : T \sqsubset T'\ and\ f_o(T') = f_o(T)$ **then**
10: $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{X}$
11: **end if**
12: **for all** $T' \in \mathcal{T}\ such\ that\ T' \sqsubset T\ and\ f_o(T') = f_o(T')$ **do**
13: $\mathcal{T} \leftarrow \mathcal{T} \setminus \{T'\}$
14: **end for**
15: **end if**
16: **end while**
17: $printSolutions(\mathcal{T})$

FIG. 3 – *L'algorithme IMIT_CLOSED.*

5.2 Données de journaux d'activité Web

Nous avons élaboré un jeu de données à partir du journal d'activité Web de notre université. Cependant, au lieu d'étiqueter les nœuds avec les URLs des pages consultées, nous leur avons associé un ensemble de mots clés portant sur le contenu des pages. Cette approche permet de capturer les habitudes de navigation des utilisateurs même dans le cas où la structure du site Web change. Le jeu de données est composé de 126 396 arbres annotés avec des itemsets de taille 10 (10 mots clés par page).

5.3 Évaluation des performances

La figure 4 montre les temps d'exécution d'IMIT_CONTENT_CLOSED pour la fouille de motifs c-fermés induits et non-ordonnés sur les 5 jeux de données artificiels. A titre de comparaison, nous avons ajouté dans la figure les temps d'exécution de SLEUTH (Zaki, 2004), une implémentation de référence du paradigme d'extension de classes d'équivalence, pour fouiller T10K. IMIT_CONTENT_CLOSED est environ deux fois plus lent que SLEUTH pour toutes les valeurs de support, sauf pour les plus petites pour lesquelles SLEUTH est pénalisé par le coût de la jointure sur des millions de motifs fréquents. Bien qu'IMIT_CONTENT_CLOSED soit en général plus lent que SLEUTH, les résultats sont satisfaisants car notre algorithme est conçu pour fouiller des arbres attribués. Il est normal d'être moins performant que des implémentations dédiées à la fouille d'arbres étiquetés.

La figure montre également que la fouille d'arbres attribués demande beaucoup plus de ressources que la fouille d'arbres étiquetés ; et la différence est largement sous-estimée car seuls les motifs c-fermés sont listés. Extraire tous les motifs génère un grand nombre de solutions

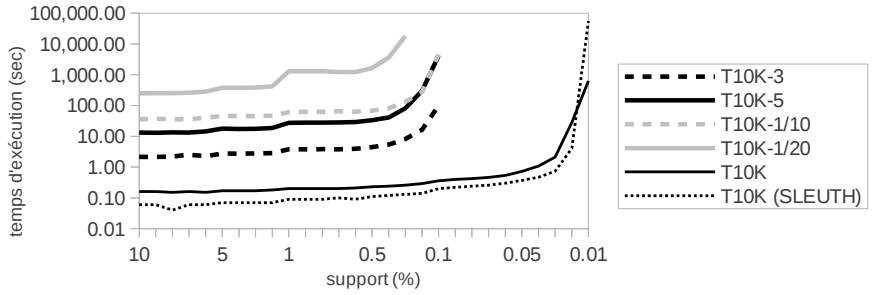

FIG. 4 – *Temps d'exécution d'IMIT_CONTENT_CLOSED pour extraire les motifs c-fermés, induits et non-ordonnés à partir de cinq jeux de données artificiels.*

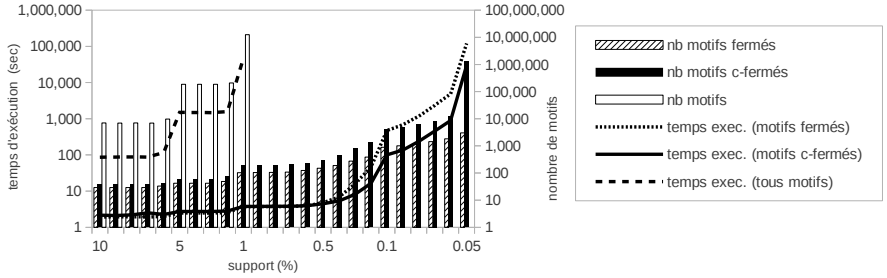

FIG. 5 – *Fouille de motifs induits et non-ordonnés avec 3 versions d'IMIT sur les données T10K-3.*

et prend énormément de temps. Pour donner une idée, la fouille du jeu de données T10K-3 avec un support minimum de 1% génère plus de 12 millions de motifs en 15 heures (figure 5). Rechercher uniquement les sous-arbres c-fermés permet de réduire à la fois le nombre de motifs et le temps d'exécution. Ainsi, avec un support minimum de 1% , 200 motifs c-fermés sont identifiés en 4 secondes. Comme illustré dans la même figure, la recherche de motifs fermés permet de réduire encore le nombre de motifs. Avec un support de 1%, par exemple, le nombre de motifs tombe à 103. Cependant, à cause des tests coûteux d'isomorphisme d'arbres, en contrepartie, les performances s'effondrent lorsque les motifs sont nombreux.

La figure 6 montre les temps d'exécution et le nombre de motifs c-fermés trouvés dans les données de journaux d'activité Web. Ce jeu de données réel est plus important que ceux utilisés précédemment et la fouille ne peut être réalisée raisonnablement avec un support inférieur à 10%. La fouille avec un support de 6% retourne 360 motifs en 6 heures.

6 Conclusion et perspectives

Dans cet article, nous avons introduit le problème de la fouille d'arbres attribués. Nous avons exploré des méthodes permettant d'énumérer tous les motifs fréquents ou seulement les fermés. Ces méthodes se sont révélées inefficaces à cause, dans le premier cas, du nombre

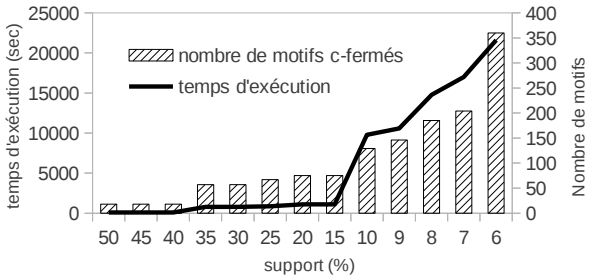

FIG. 6 – *Utilisation d'IMIT_CONTENT_CLOSED pour extraire les motifs c-fermés induits et non-ordonnés des données de journaux d'activité Web.*

important de motifs retournés, et dans le second cas, du coût des tests d'isomorphisme. Nous avons finalement proposé d'énumérer les motifs c-fermés qui constituent une représentation concise des sous-arbres attribués fréquents définie en fonction de l'inclusion d'itemsets. Cette représentation permet de réduire drastiquement à la fois le nombre de motifs retournés et le temps d'exécution. Nous avons évalué les performances d'IMIT_CONTENT_CLOSED, l'algorithme de fouille des motifs c-fermés, et montré qu'il permet de fouiller des jeux de données contenant plusieurs milliers d'arbres attribués.

Un futur travail possible consiste à développer un algorithme permettant de fouiller de manière efficace les sous-arbres attribués fermés. Une autre possibilité est de développer des méthodes similaires pour fouiller des structures plus complexes comme les graphes attribués.

Remerciements. Ce travail a été financé par le contrat ANR-2010-COSI-012 FOSTER.

Références

Agrawal, R., T. Imieliński, et A. Swami (1993). Mining association rules between sets of items in large databases. *SIGMOD Rec. 22*(2), 207–216.

Agrawal, R. et R. Srikant (1995). Mining sequential patterns. In *ICDE'95*, pp. 3–14.

Asai, T., K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, et S. Arikawa (2002). Efficient substructure discovery from large semi-structured data. In *SDM'02*, pp. 158–174.

Ayres, J., J. Flannick, J. Gehrke, et T. Yiu (2002). Sequential pattern mining using a bitmap representation. In *KDD'02*, pp. 429–435.

Balcázar, J. L., A. Bifet, et A. Lozano (2010). Mining frequent closed rooted trees. *Mach. Learn. 78*(1-2), 1–33.

Chehreghani, M. H. (2011). Efficiently mining unordered trees. In *ICDM'11*, pp. 111–120.

Chi, Y., R. R. Muntz, S. Nijssen, et J. N. Kok (2004a). Frequent subtree mining - an overview. *Fundam. Inf. 66*(1-2), 161–198.

Chi, Y., Y. Yang, et R. R. Muntz (2004b). Hybridtreeminer : An efficient algorithm for mining frequent rooted trees and free trees using canonical form. In *SSDBM'04*, pp. 11–20.

Chi, Y., Y. Yang, Y. Xia, et R. R. Muntz (2004c). Cmtreeminer : Mining both closed and maximal frequent subtrees. In *PAKDD'04*, pp. 63–73.

Fukuzaki, M., M. Seki, H. Kashima, et J. Sese (2010). Finding itemset-sharing patterns in a large itemset-associated graph. In *PAKDD'10*, pp. 147–159.

Han, J., J. Pei, et Y. Yin (2000). Mining frequent patterns without candidate generation. *SIGMOD Rec. 29*(2), 1–12.

Hido, S. et H. Kawano (2005). Amiot : Induced ordered tree mining in tree-structured databases. In *ICDM'05*, pp. 170–177.

Mannila, H. et H. Toivonen (2005). Multiple uses of frequent sets and condensed representations. In *KDD'05*, pp. 189–194.

Miyoshi, Y., T. Ozaki, et T. Ohkawa (2009). Frequent pattern discovery from a single graph with quantitative itemsets. In *ICDM Workshops*, pp. 527–532.

Nijssen, S. et J. N. Kok (2003). Efficient discovery of frequent unordered trees. In *First International Workshop on Mining Graphs, Trees and Sequences (MGTS'03)*.

Pasquier, N., Y. Bastide, R. Taouil, et L. Lakhal (1999). Discovering frequent closed itemsets for association rules. In *ICDT'99*, pp. 398–416.

Termier, A., M.-C. Rousset, et M. Sebag (2004). Dryade : A new approach for discovering closed frequent trees in heterogeneous tree databases. In *ICDM'04*, pp. 543–546.

Termier, A., M.-C. Rousset, M. Sebag, K. Ohara, T. Washio, et H. Motoda (2008). Dryade-parent, an efficient and robust closed attribute tree mining algorithm. *IEEE Trans. on Knowl. and Data Eng. 20*(3), 300–320.

Wang, C., M. Hong, J. Pei, H. Zhou, W. Wang, et B. Shi (2004). Efficient pattern-growth methods for frequent tree pattern mining. In *PAKDD'04*, pp. 441–451.

Washio, T. et H. Motoda (2003). State of the art of graph-based data mining. *SIGKDD Explor. Newsl. 5*(1), 59–68.

Xiao, Y., J.-F. Yao, Z. Li, et M. H. Dunham (2003). Efficient data mining for maximal frequent subtrees. In *ICDM'03*, pp. 379–386.

Zaki, M. J. (2002). Efficiently mining frequent trees in a forest. In *KDD'02*, pp. 71–80.

Zaki, M. J. (2004). Efficiently mining frequent embedded unordered trees. *Fundam. Inf. 66*(1-2), 33–52.

Summary

Frequent pattern mining is an important data mining task with a broad range of applications. Initially focused on the discovery of frequent itemsets, studies were extended to mine structural forms like sequences, trees or graphs. In this paper, we introduce a new data mining method that consists in mining new kind of patterns in a collection of attributed trees. Attributed trees are trees in which vertices are associated to itemsets. Mining this type of patterns, which combines tree mining and itemset mining, requires the exploration of a huge search space. We present new algorithms for attributed trees mining and show that their implementations can efficiently list frequent patterns from large datasets.