Extraction de motifs dans des graphes orientés attribués en présence d'automorphisme

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

*Université de Nouvelle Calédonie
PPME, BP R4, F-98851 Nouméa, Nouvelle Caledonie
{frederic.flouvat,jeremy.sanhes,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. Les graphes orientés attribués sont des graphes orientés dans lesquels les nœuds sont associés à un ensemble d'attributs. De nombreuses données, issues du monde réel, peuvent être représentées par ce type de structure, mais encore peu d'algorithmes sont capables de les traiter directement. La fouille des graphes attribués est difficile, car elle nécessite de combiner l'exploration de la structure du graphe avec l'identification d'itemsets fréquents. De plus, du fait de l'explosion combinatoire des itemsets, les isomorphismes de sous-graphes, dont la présence impacte énormément les performances des algorithmes de fouille, sont beaucoup plus nombreux que dans les graphes étiquetés.

Dans cet article, nous présentons une nouvelle méthode de fouille de données qui permet d'extraire des motifs fréquents à partir d'un ou de plusieurs graphes orientés attribués. Nous montrons comment réduire l'explosion combinatoire provoquée par les isomorphismes de sous-graphes en traitant de manière particulière les motifs automorphes.

1 Introduction

Les graphes orientés sont des structures adaptées à la modélisation d'un grand nombre de données complexes présentes dans le monde réel. Les réseaux de régulation génétique, par exemple, sont des graphes orientés où les nœuds représentent des gènes et les arcs, des relations d'inhibition ou d'activation. Le Web est modélisé par un graphe orienté dans lequel les nœuds sont des pages et les arcs des liens hypertextes. La circulation des emails dans une organisation, les graphes sociaux dans lesquels un individu peut suivre les activités d'autres personnes, les réseaux de citations dans lesquels un article cite d'autres articles sont également représentés par des graphes orientés.

Du fait de ces nombreuses applications, ces structures sont beaucoup étudiées en théorie des graphes, et, plus récemment, dans le contexte de la fouille de données. La plupart des

travaux portent sur des graphes non étiquetés ou dont les nœuds ne sont associés qu'à une unique étiquette. Cependant, dans les graphes modélisant des données réelles, les nœuds, qui représentent des objets, peuvent être associés à de nombreuses caractéristiques. Les graphes dans lesquels les nœuds sont annotés avec des ensembles d'attributs (ou itemsets) sont appelés graphes attribués et à l'heure actuelle, encore peu de travaux se consacrent à leur analyse.

L'un des problèmes phares abordés dans la fouille de graphes étiquetés concerne l'identification de sous-graphes qui apparaissent avec une fréquence minimum spécifiée par l'utilisateur. Ces motifs récurrents peuvent apporter un éclairage particulier sur la manière dont les nœuds sont organisés. Fouiller des graphes attribués est d'un grand intérêt, car cela permet d'identifier des motifs structurels, mais également, de mettre en évidence des liens entre les attributs des nœuds.

Deux raisons principales rendent la fouille des graphes attribués très difficile. En premier lieu, il est nécessaire de combiner de manière non triviale l'exploration de la structure du graphe avec l'identification d'itemsets fréquents associés aux nœuds. La seconde raison est que, comme pour la fouille des graphes étiquetés, les performances sont très impactées par la présence de sous-graphes isomorphes (Yan et Han, 2002). Comme indiqué par (Yan et Han, 2002), dans le cas de graphes creux avec des labels diversifiés, le nombre de sous-graphes isomorphes est la plupart du temps faible. Cela n'est plus le cas dans le cas des graphes attribués où la combinatoire sur les itemsets génère fréquemment un nombre important d'isomorphismes de sous-graphes.

Les contributions clés de notre travail sont les suivantes : i) nous présentons le problème de la fouille de structures fréquentes dans un ou plusieurs graphes attribués orientés, puis, ii) nous montrons comment l'identification des motifs fréquents peut être réalisée en adoptant une stratégie d'exploration de l'arbre couvrant de chaque nœud. iii) Nous définissons une nouvelle forme canonique adaptée au parcours de l'espace de recherche, iv) nous détaillons la manière d'étendre les motifs avec des arcs réentrants et v) nous traitons de la prise en compte des cycles. vi) Nous analysons le cas particulier des motifs automorphes et nous montrons comment l'explosion combinatoire provoquée par l'isomorphisme de sous-graphes peut être considérablement limitée. Avant de conclure, vii) nous présentons les résultats obtenus en analysant plusieurs jeux de données artificiels et réels avec l'algorithme AADAGE (Automorphism Aware Directed Attributed Graph Explorer) que nous avons développé.

2 Etat de l'art

De nombreux algorithmes ont été proposés pour fouiller des graphes étiquetés. La quasi-totalité des solutions proposées découlent de techniques développées pour la fouille d'itemsets et sont basées sur une même idée qui consiste à explorer l'espace des solutions en faisant grossir des sous-graphes candidats et en éliminant les motifs infréquents (Agrawal et al., 1993; Jiang et al., 2013). Là où les méthodes se différencient, c'est dans leur manière de parcourir l'espace de recherche et d'éviter la génération de solutions redondantes. En effet, contrairement à la fouille d'itemsets où il est facile de générer toutes les solutions de manière unique, dans la fouille de graphes, il est très fréquent de construire plusieurs fois le même motif.

L'une des méthodes privilégiées pour éviter d'explorer plusieurs fois une même partie de l'espace de recherche est d'ajouter des arcs et des nœuds aux motifs candidats de manière ordonnée et d'utiliser une représentation canonique des sous-graphes. L'idée sous-jacente à

l'utilisation des formes canoniques est de n'étendre qu'une seule des solutions isomorphes. Pour éviter d'effectuer des tests coûteux d'isomorphisme complets, une représentation canonique adaptée à la manière d'étendre les sous-graphes est choisie. Cette forme canonique permet de décider rapidement si un motif a déjà été examiné et donc, ne doit pas être étendu. La solution la plus utilisée est de créer une séquence de tous les arcs contenus dans le sous-graphe dans l'ordre ou les arcs ont été ajoutés (Borgelt, 2007; Yan et Han, 2002).

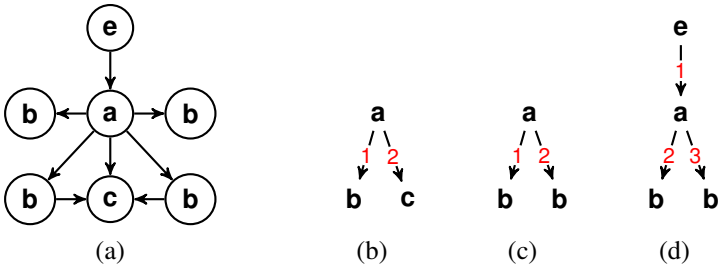


FIG. 1: Exemple d'un graphe étiqueté orienté (a) et de 3 motifs (b, c et d) présents dans le graphe (les nombres affichés sur les arcs indiquent leur ordre de création).

L'utilisation des formes canoniques, si elle permet d'éviter d'explorer plusieurs fois le même sous-graphe, ne dispense pas de la nécessité d'identifier toutes les manières de générer les motifs. Ainsi, si l'on ordonne les nœuds dans l'ordre lexicographique de leurs étiquettes, le motif de la figure 1(b), est canonique, mais il est présent 4 fois dans le graphe initial (a). Les motifs (c) et (d), qui sont eux aussi canoniques, peuvent être générés de 12 façons différentes (dans le graphe initial, il y a 12 manières possibles de choisir 2 nœuds b parmi les 4 fils étiquetés b du nœud a). Il est nécessaire de tenir compte de toutes ces formes car la configuration choisie conditionne les extensions futures du motif ; par exemple, la possibilité ou non d'étendre chacun des nœuds b avec un nœud étiqueté c. Les motifs qui possèdent de nombreux isomorphismes de sous-graphes avec le graphe analysé posent des difficultés à tous les algorithmes existants, car le problème d'isomorphismes de sous-graphes en lui même est NP-complet. Les motifs (c) et (d), qui possèdent des automorphismes posent le plus de problèmes et contribuent de manière importante à la dégradation des performances des algorithmes, car le nombre d'isomorphismes de sous-graphes avec le graphe initial est augmenté.

Dans le cas des graphes étiquetés creux, ce nombre reste toutefois relativement raisonnable. Au contraire, dans les graphes attribués, puisque plusieurs attributs sont associés aux nœuds, la probabilité d'observer des sous-graphes partageant un ou plusieurs attributs identiques est grandement augmentée. Le problème devient alors très présent. Une manière de le contourner est d'éliminer du graphe les items les plus fréquents mais l'on se prive de tous les motifs contenant cet attribut, y compris certains qui peuvent potentiellement être intéressants.

Certaines études traitent spécifiquement de l'analyse des graphes dans lesquels les nœuds sont associés à des itemsets comme les travaux de Miyoshi et al. (2009) ou ceux de Fukuzaki et al. (2010). Les problèmes abordés dans ces études sont différents du notre dans le sens où ils traitent de l'analyse de graphes non orientés et ont pour but de rechercher des sous-graphes partageant les mêmes itemsets.

Dans notre travail, nous cherchons à identifier la présence récurrente de certains attributs associés à des nœuds connectés. Dans ses objectifs, notre travail est plus proche de la fouille

de séquence ou d'arbres attribués. Dans une précédent étude Pasquier et al. (2013), nous avons travaillé sur l'identification de sous structures reflétant des évolutions d'itemsets. Cependant, l'algorithme que nous avions proposé ne s'appliquait que sur des jeux de données prenant la forme d'arbres attribués.

3 Concepts généraux et définition du problème

Dans cette section, nous introduisons les concepts et définitions nécessaires et présentons le problème de la fouille de graphes attribués orientés.

3.1 Préliminaires

Soit $\mathcal{I} = \{i_1, i_2, .., i_n\}$ un ensemble d'items. Un **itemset** est un ensemble $\mathcal{P} \subseteq \mathcal{I}$. Nous considérons que les items appartenant à un itemset sont triés selon l'ordre lexicographique et peuvent être accédés par leur index (p.ex. $\mathcal{I}_2$ pour référencer le deuxième item de $\mathcal{I}$).

Un **graphe attribué orienté** $G = (V, E, \lambda)$ sur un ensemble d'items $\mathcal{I}$ consiste en un ensemble de nœuds V, un ensemble (orienté) d'arcs $E \subseteq \{(u, v) \in V^2 / u \neq v\}$ et une fonction $\lambda(v) : V \rightarrow pow(\mathcal{I})$ qui associe un ensemble d'items $i \in \mathcal{I}$ à chaque nœud $v \in V$. La notation $pow(\mathcal{I})$ représente l'ensemble des parties de $\mathcal{I}$. Pour un arc $(u, v) \in E$, u est le parent de v et v est le fils de u. Il existe un cycle dans le graphe orienté si un chemin peut être trouvé entre un nœud et lui même. Un graphe attribué orienté est **enraciné** si l'on peut trouver un nœud v tel qu'il existe un chemin entre v et n'importe quel autre nœud du graphe.

Deux graphes attribués $G_1 = (V_1, E_1, \lambda_1)$ et $G_2 = (V_2, E_2, \lambda_2)$ sont **isomorphes** *ssi* il existe une bijection $\varphi : V_1 \rightarrow V_2$ telle que : (i) $\forall v \in V_1 : \lambda_1(v) = \lambda_2(\varphi(v))$, (ii) $\forall (u, v) \in E_1 \Leftrightarrow (\varphi(u), \varphi(v)) \in E_2$. Un **automorphisme** est un isomorphisme pour lequel $G_1 = G_2$.

Un graphe attribué $G_1 = (V_1, E_1, \lambda_1)$ est un **sous graphe attribué** de $G = (V, E, \lambda)$ *ssi* il existe une correspondance $\varphi : V_1 \rightarrow V$ telle que : (i) $\forall v \in V_1 : \varphi(v) \in V$, (ii) $\forall v \in V_1 : \lambda_1(v) \subseteq \lambda(\varphi(v))$, et (iii) $\forall (u, v) \in E_1 : (\varphi(u), \varphi(v)) \in E$.

Chaque graphe attribué G_1 qui est isomorphique à G définit un **plongement** (embedding en anglais) de G_1 dans G. A noter que, dans certains cas, G_1 peut être plongé plusieurs fois dans G en effectuant des permutations dans la **mise en correspondance** des nœuds.

3.2 Définition du problème

Les données à analyser sont disponibles soit sous la forme d'une base de données $\mathcal{B}$ de graphes orientés attribués, soit sous la forme d'un unique graphe orienté attribué G, non nécessairement connexe. Dans le premier cas, il est possible de calculer, pour chaque motif identifié P, une **fréquence par transaction** qui est donnée par le nombre de graphes dans $\mathcal{B}$ pour lesquels P est un sous-graphe attribué. Dans le second cas, nous utilisons la mesure de fréquence définie par Bringmann et Nijssen (2008) qui est égale au nombre minimal d'occurrences dans G de chacun des nœuds du motif. Nous pouvons avec cette mesure calculer une fréquence relative qui est donnée par la fréquence absolue divisée par le nombre de nœuds du graphe. Nous considérons qu'un motif est fréquent si sa fréquence est supérieure ou égale à un seuil minimum. Le problème consiste à énumérer tous les motifs fréquents présents dans un jeu de données.

4 Forme canonique des graphes orientés attribués

La forme canonique que nous utilisons n'est pas basée sur une séquence d'arcs, mais sur une liste de nœuds dans l'ordre dans lequel ils ont été ajoutés à l'arbre couvrant du sous-graphe en effectuant un parcours en profondeur d'abord. Pour identifier de manière non ambiguë chaque arbre couvrant, il est suffisant d'identifier chaque nœud par ses attributs et sa profondeur. Comme nous devons également représenter les liens réentrants, nous utilisons un troisième attribut qui contient l'index du nœud pointé si celui-ci est déjà présent dans le motif.

4.1 Ordre portant sur les itemsets

Dans notre application, les nœuds sont associés à des itemsets et il convient de définir un ordre total sur les itemsets. Etant donné deux itemsets, $\mathcal{I}$ et $\mathcal{J}$ ($\mathcal{I} \neq \mathcal{J}$), on dit que $\mathcal{I} < \mathcal{J}$ ssi $\exists k \in [1, \min(|\mathcal{I}|, |\mathcal{J}|)]$ tel que (i) $\forall i < k : \mathcal{I}_i = \mathcal{J}_i$ et (ii) $\mathcal{I}_k < \mathcal{J}_k$ ou $k = |\mathcal{J}|$

4.2 Ordre portant sur les nœuds

Le code d'un nœud est un triplet $(d, \mathcal{Q}, p)$ où d représente la profondeur du nœud dans l'arbre couvrant, $\mathcal{Q}$ représente l'ensemble des items associés au nœud et p vaut 0 si le nœud est utilisé pour la première fois ou est égal au numéro d'ordre du nœud si celui-ci est déjà présent dans le motif. Pour comparer deux nœuds, il suffit de comparer leurs codes sous forme de triplets. Soit $T_1 = (d_1, \mathcal{Q}_1, p_1)$ et $T_2 = (d_2, \mathcal{Q}_2, p_2)$, on dit que $T_1 < T_2$ ssi l'une des assertions suivantes est vraie : (i) $d_1 > d_2$, (ii) $d_1 = d_2$ et $\mathcal{Q}_1 < \mathcal{Q}_2$, (iii) $d_1 = d_2$ et $\mathcal{Q}_1 = \mathcal{Q}_2$ et $p_1 < p_2$.

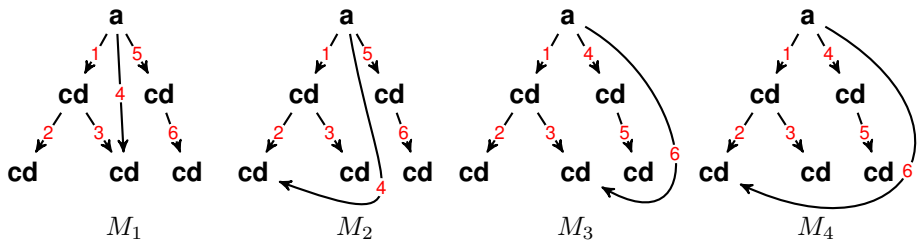


FIG. 2: Exemple de quatre motifs isomorphes (les nombres affichés sur les arcs indiquent leur ordre de création).

4.3 Ordre portant sur les sous-graphes attribués

Étant donné un motif M de racine $r(M)$, nous définissons un code sur la racine de ce motif $code(r(M))$, obtenu en concaténant les codes des nœuds présents dans le motif dans l'ordre dans lequel ils ont été ajoutés. La forme canonique d'un sous-graphe attribué est déterminée simplement en sélectionnant le plus petit code.

Soit deux motifs M_1 et M_2 représentés respectivement par les codes $C_1 = \{c_1^1, ..., c_1^m\}$ et $C_2 = \{c_2^1, ..., c_2^n\}$, on dit que $C_1 < C_2$ ssi l'une des assertions suivantes est vraie : (i)

$\exists k \in [1, min(m,n)]$ tel que $\forall i < k : c_1^i = c_2^i$ et $c_1^k < c_2^k$, (ii) $\forall i < min(m,n) : c_1^i = c_2^i$ et $m > n$.

La figure 2 présente quatre motifs isomorphes d'un même graphe orienté attribué. Leurs codes sont respectivement représentés par les séquences suivantes :

$code(r(M_1))$ = (0,a,0) (1,cd,0) (2,cd,0) (2,cd,0) (1,cd,4) (1,cd,0) (2,cd,0)

$code(r(M_2))$ = (0,a,0) (1,cd,0) (2,cd,0) (2,cd,0) (1,cd,3) (1,cd,0) (2,cd,0)

$code(r(M_3))$ = (0,a,0) (1,cd,0) (2,cd,0) (2,cd,0) (1,cd,0) (2,cd,0) (1,cd,4)

$code(r(M_4))$ = (0,a,0) (1,cd,0) (2,cd,0) (2,cd,0) (1,cd,0) (2,cd,0) (1,cd,3)

En utilisant l'ordre des triplets définie précédemment, on trouve que $code(r(M_4)) < code(r(M_3)) < code(r(M_2)) < code(r(M_1))$, donc le motif M_4 est sous forme canonique.

5 Parcours de l'espace de recherche

Tous les items présents dans le graphe de départ sont listés et constituent les motifs initiaux qui vont être progressivement étendus. L'arbre de recherche est construit en utilisant l'ordre défini précédemment sur les graphes orientés attribués, en effectuant un parcours en profondeur d'abord.

5.1 Enumération des arbres couvrants

L'utilisation de la stratégie définie par Pasquier et al. (2013) permet d'énumérer tous les arbres couvrants. La génération de nouveaux motifs est effectuée par extension des nœuds figurant dans le chemin le plus à droite du motif examiné (rightmost path extension) (Chi et al., 2004). Deux types d'extensions sont possibles : l'**extension d'itemset** permet d'ajouter un nouvel item à l'itemset associé au nœud terminal le plus à droite, l'**extension de structure** consiste à ajouter un fils à l'un des nœuds composant le chemin droit. La méthode est complète, mais peut générer des motifs redondants qui prennent la forme d'arbres isomorphes. A chaque génération, les motifs non canoniques sont écartés. L'inconvénient de cette approche est qu'elle ne permet de ne trouver que des motifs enracinés.

5.2 Identification des motifs canoniques et automorphes

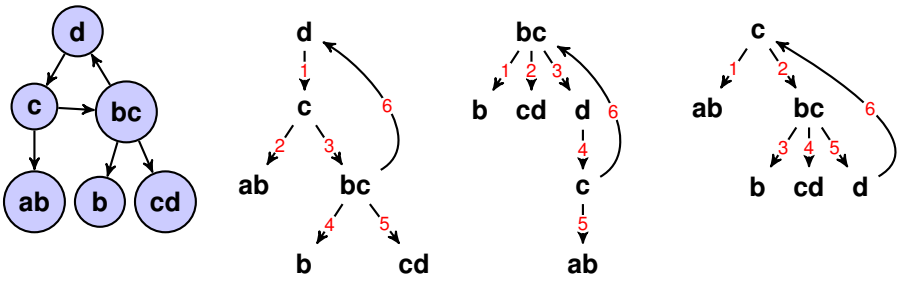
Le code sur les graphes défini précédemment possède, comme la plupart des codes utilisés par les algorithmes de fouille de graphes, la propriété suivante : chaque préfixe d'un code canonique est lui même canonique (Huan et al., 2003; Yan et Han, 2002). L'extension d'un motif ne change pas l'ordre des triplets lors de la génération d'un motif canonique. Ainsi, chaque motif canonique ne peut être obtenu qu'en ajoutant un nouveau triplet ou en ajoutant un item à l'itemset associé au dernier triplet.

En utilisant la stratégie d'extension du chemin droit, il est possible de vérifier la canonicité d'un motif en ne testant que les nœuds appartenant au chemin droit. Soit les fonctions l et p qui, appliquées à un nœud retournent respectivement son dernier et son avant-dernier fils. Il est possible de déterminer la canonicité d'un motif de la manière suivante : pour tous les nœuds n composant le chemin droit, si $\exists p(n)$ et $code(p(n)) \leq code(l(n))$, alors, le motif est sous forme canonique. Il est, de la même manière, assez aisé d'identifier les extensions qui

produisent des motifs automorphes. S'il existe un noeud n dans le chemin droit du motif tel que n a plus d'un fils et $code(p(n)) = code(l(n))$, alors le motif possède des automorphismes. Sur un tel motif, on appelle **nœud de séparation** le nœud du chemin droit, le plus proche de la racine, qui possède plus d'un fils. Ainsi, les motifs illustrés en figures 1(c) et 1(d) possèdent des automorphismes. Dans ces deux figures, le nœud de séparation est le nœud a.

5.3 Prise en compte des liens réentrants

A partir de la stratégie d'énumération des arbres couvrants décrite précédemment, il est possible de définir une méthode permettant d'énumérer tous les sous-graphes. Pour cela, il faut identifier lorsqu'une extension structurelle ajoute un nœud qui est déjà présent dans le motif examiné. Lorsque ce cas se produit, le nœud est ajouté dans le motif comme s'il s'agissait d'une extension structurelle normale, mais l'on indique qu'il s'agit d'une réutilisation d'un nœud déjà présent en mémorisant l'index du nœud pointé. L'exploration de l'espace de recherche est stoppée à partir de ce nœud. En effet, du fait de notre stratégie d'exploration, nous sommes certains que tous les nœuds faisant partie du chemin le plus à droite du motif examiné sont étendus lors de la phase de génération en cours. Toutes les structures ne faisant pas partie du chemin le plus à droite ont quant à elles, déjà été complètement explorées. Concrètement, la réutilisation d'un nœud déjà présent est reflétée dans le code du graphe par l'ajout d'un triplet qui fait référence à un autre triplet. Cela ne génère aucun changement concernant les propriétés du code, la complétude de la méthode et la manière d'éliminer les formes non canoniques.



Graphe G Motif M_1 Motif M_2 Motif M_3

FIG. 3: Illustration de l'élagage des motifs non minimaux avec cycle.

5.4 Prise en compte des cycles

L'extension d'un motif peut donner lieu à la création d'un cycle si et seulement si le nouveau nœud ajouté pointe sur l'un des nœuds déjà présents dans le chemin droit. Lorsqu'un cycle est créé, on procède de la même manière que pour les autres liens réentrants : un nouveau nœud est ajouté avec l'index du nœud réutilisé, puis les extensions à partir de ce nœud sont stoppées. La complétude de la méthode n'est pas affectée par cette opération. Il y a cependant un problème au niveau de la redondance des solutions si le nœud réutilisé est la racine du motif car dans ce cas, de nombreux graphes isomorphes sont générés en débutant l'exploration à partir des autres nœuds appartenant au chemin droit.

Nous résolvons ce problème en générant tous les motifs ayant pour racine l'un des nœuds du chemin droit. En comparant les codes des motifs ainsi obtenus, nous en choisissons un comme motif canonique (le plus petit code) et ne continuons l'exploration que pour ce motif. Dans la figure 3, M_1 représente un motif trouvé dans le graphe G. Le nœud d, qui est à la racine du motif vient d'être ajouté ce qui crée un cycle. Le cycle est composé des 3 nœuds appartenant au chemin droit (d, c et bc). Le motif enraciné en d est connu, il reste donc à générer deux autres motifs en fixant arbitrairement leur racine à bc (motif M_2) et c (motif M_3). Le motif qui a le plus petit code est bien évidemment celui enraciné en bc donc nous stoppons ici l'exploration du motif M_1.

6 Traitement des motifs automorphes

L'utilisation de formes canoniques permet d'élaguer l'espace de recherche et cette optimisation est utilisée dans notre algorithme. Cependant, cela ne permet pas de résoudre le problème posé par la présence de motifs automorphes.

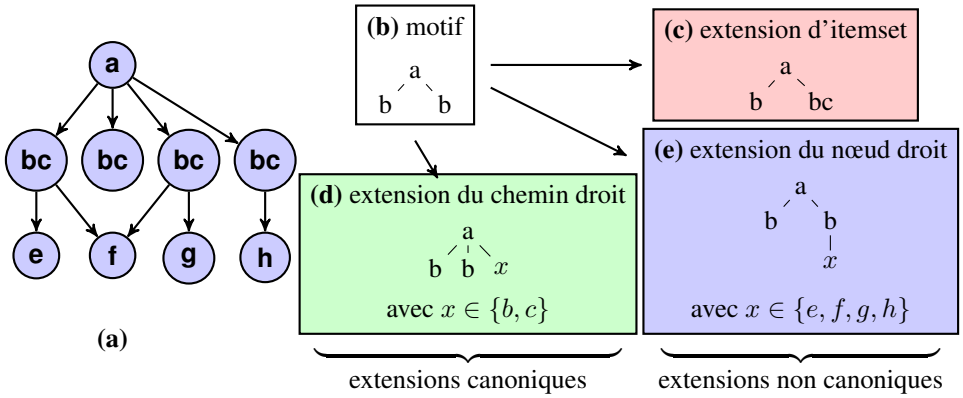


FIG. 4: Illustration des extensions d'un motif canonique possédant des automorphismes.

Prenons par exemple le graphe orienté attribué représenté dans la figure 4(a). Ce graphe contient plusieurs sous-structures automorphes qui rendent l'élagage basé sur l'utilisation des formes canoniques moins opérant. Lorsque le motif illustré dans la figure 4(b) est analysé, il est conservé et étendu puisqu'il est sous forme canonique. Or, ce motif peut être obtenu de douze façons différentes. L'algorithme naïf consiste, pour chacune de ces formes, à étendre le motif en utilisant comme défini précédemment, l'extension d'itemset ou l'extension de structure.

L'extension d'itemset génère le motif illustré en (c). Ce motif, qui n'est pas sous forme canonique, sera écarté. L'extension de structure à partir du nœud b génère l'un des motifs décrits en (e). Dans tous les cas, le motif obtenu n'est pas sous forme canonique donc il sera écarté. La seule opération qui permet de générer un motif canonique est l'extension de structure à partir du nœud de séparation (d).

Lorsque l'on identifie un nouveau motif possédant des automorphismes, on sait que l'on ne pourra pas faire d'extension sur le dernier nœud, ni sur tous les nœuds situés après le nœud de séparation. En effet, pour un nœud de séparation n, on a $code(p(n)) = code(l(n))$ donc,

n'importe quelle extension effectuée sur le motif de racine $l(n)$ rendra son code inférieur à $p(n)$, donc le motif ne sera plus canonique. Nous pouvons utiliser ce fait pour éviter de générer inutilement des motifs. Ceci est une première optimisation, mais son impact est limité.

La seconde optimisation consiste à supprimer certaines manière d'obtenir des motifs automorphes qui ne permettent pas de générer de nouveaux motifs canoniques. Dans la figure 1, par exemple, le motif (d), qui est obtenu de 12 façons différentes, ne peut être étendu que par l'ajout d'un troisième fils au nœud de séparation a. Quelle que soit la manière dont le motif (d) a été obtenu, les possibilités d'extension sont inchangées. Il est donc possible, dans ce cas, sans omettre de solution, de ne conserver qu'une seule mise en correspondance du motif ; n'importe laquelle.

Avec le graphe tel que présenté dans la figure 4(a), la situation est plus compliquée car il est possible, à partir du motif, d'ajouter un fils c au nœud a. Dans ce cas, il sera possible lors de la génération suivante, d'appliquer n'importe quel type d'extension sur tous les nœuds du chemin droit. Dans l'exemple, on pourrait en particulier ajouter e, f, g ou h en tant que fils du nœud c. Pour que tous les motifs possibles puissent être générés, il faut conserver les possibilités d'étendre le motif avec 2 nœuds bc parmi les 3 nœuds bc ayant un fils, soit $\binom{3}{2}$ = 3 formes différentes.

Dans un graphe réel, la situation est souvent beaucoup moins simple et déterminer quelles formes il est possible de supprimer n'est pas une tâche facile. Pour éviter d'effectuer des calculs portant sur la structure du graphe qui nuiraient aux performances de l'algorithme, on choisit de ne conserver qu'une seule forme parmi celles utilisant les mêmes nœuds. On passe ainsi à un nombre de formes déterminé par un nombre d'arrangements possibles à un nombre de formes basé sur le nombre de combinaisons. Cette solution n'est pas optimale mais elle est suffisante pour limiter l'explosion combinatoire. Sur une configuration en étoile dans laquelle un nœud possède n fils avec une même étiquette, il existe $\sum_{i=1}^{n} {}^{n}P_k$ manières différentes de générer des motifs inclus. Après optimisation, ce nombre chute à $\sum_{i=0}^{n-1} (n-i)\binom{n}{i}$.

7 Résultats expérimentaux

La méthode décrite dans l'article a été implémentée en C++ avec la STL. Les expérimentations ont été effectuées sur un ordinateur avec Ubuntu 13.04 basé sur un processeur Intel©Core™i5-2400 @ 3.10GHz avec 12 Gb de mémoire. Tous les temps d'exécution incluent la phase de prétraitement et la sauvegarde des résultats.

7.1 Jeux de données artificiels

Un ensemble de 1000 graphes orientés comprenant chacun 1000 nœuds et 5000 arcs ont été générés avec le programme de Johnsonbaugh et Kalin (1991). Cet ensemble de graphes a servi de base à la création de cinq jeux de données de graphes orientés attribués, nommés $A1$ à $A5$, dans lesquels les nœuds ont été associés à des itemsets de taille variant de 1 à 5. Pour simuler le fait que, comme dans beaucoup de jeux de donnés réels, la plupart des attributs sont rares, mais un petit nombre est très fréquent, nous avons attribué à chaque item une valeur entière aléatoire distribuée suivant une loi puissance (concrètement, nous avons utilisé une distribution de Pareto de paramètre $\alpha = 0.05$).

7.2 Réseau de citations de PubMed Central

PubMed Central est une base de données bibliographique d'articles scientifiques dans le domaine des sciences de la vie. Les articles en Open Source sont au nombre de 322 526 mais seulement 58 728 d'entre eux citent au moins un autre article, lui même Open Source. Nous avons construit un graphe de citations où chaque nœud représente un article, chaque arc, un lien de citation et les attributs associés aux nœuds identifient les mots-clés de l'article. Le graphe est très creux puisqu'il ne contient que 60 012 arcs, cependant, certains attributs se retrouvent fréquemment associés à des sous-ensembles de nœuds connectés.

7.3 Jeux de données Google+ et Twitter

Nous avons utilisé les jeux de données Google+ et Twitter construits par McAuley et Leskovec (2012). Dans chaque jeu de données, un nœud représente un individu et les attributs associés constituent les caractéristiques de cette personne. Il existe un arc entre deux personnes lorsque l'une d'elles suit l'activité de l'autre. Les jeux de données sont conséquents (107 614 nœuds et 13 673 453 arcs pour Google+, 81 306 nœuds et 1 768 149 arcs pour Twitter).

7.4 Évaluation des performances

Les résultats de notre algorithme sont présentés dans la figure 5. La comparaison avec des algorithmes existants n'est possible que sur des graphes dans lesquels les nœuds ne sont associés qu'à une seule étiquette. Pour le jeu de données artificiel $A1$, le seul qui corresponde à un graphe étiqueté, une comparaison de AADAGE avec l'implémentations de gSpan réalisées dans le cadre du projet ParSeMiS (Wörlein et al., 2005) (une implémentation qui permet de fouiller les sous-graphes orientés) est présentée (figure 5(a)). Les deux algorithmes, qui sont paramétrés pour énumérer l'ensemble des sous-graphes orientés, enracinés et connectés, génèrent exactement les mêmes motifs. Pour les plus petites valeurs de support, la stratégie de filtrage des mises en correspondance permet de faire la différence avec gSpan. Les données Google+ et Twitter contiennent beaucoup de nœuds, beaucoup d'attributs et surtout quelques attributs très fréquents (par exemple, l'attribut "$sexe = masculin$", présent dans 52% des nœuds) dont la seule présence génère tellement de motifs qu'elle fait échouer la fouille. Dans la figure 5(d), les deux attributs de genre ont été enlevés du jeu de données Google+. Le jeu Twitter n'a pas été modifié. Malgré sa taille, le jeu de données Google+ a pu être fouillé en fixant un support minimum de 3%. Le jeu de données Twitter, pourtant plus modeste, est traité jusqu'à un support de 5%. La caractéristique commune de ces deux jeux de données est qu'ils sont traités relativement rapidement jusqu'à un seuil précis correspondant à la prise en compte d'un nouvel attribut fréquent qui fait exploser le nombre de motifs.

Avec le jeu de données de PubMed Central, il a été possible d'utiliser un support très faible. Cela a permis d'identifier quelques motifs intéressants. Un motif, par exemple, concerne un groupe d'articles annotés avec "$p53$" qui est le nom d'un oncogène. Ces articles sont cités par d'autres articles annotés "$rapamycin$" (un médicament immunosuppresseur élaboré en 1975) qui sont eux-même cités par des articles annotés "$aging$" (des effets de la rapamycine sur la sénescence cellulaire ont été découverts en 2009) et "$mTOR$" (un gène qui est inhibé par la rapamycine). Ce motif représente de manière très condensée le cheminement de certaines recherches portant sur le vieillissement.

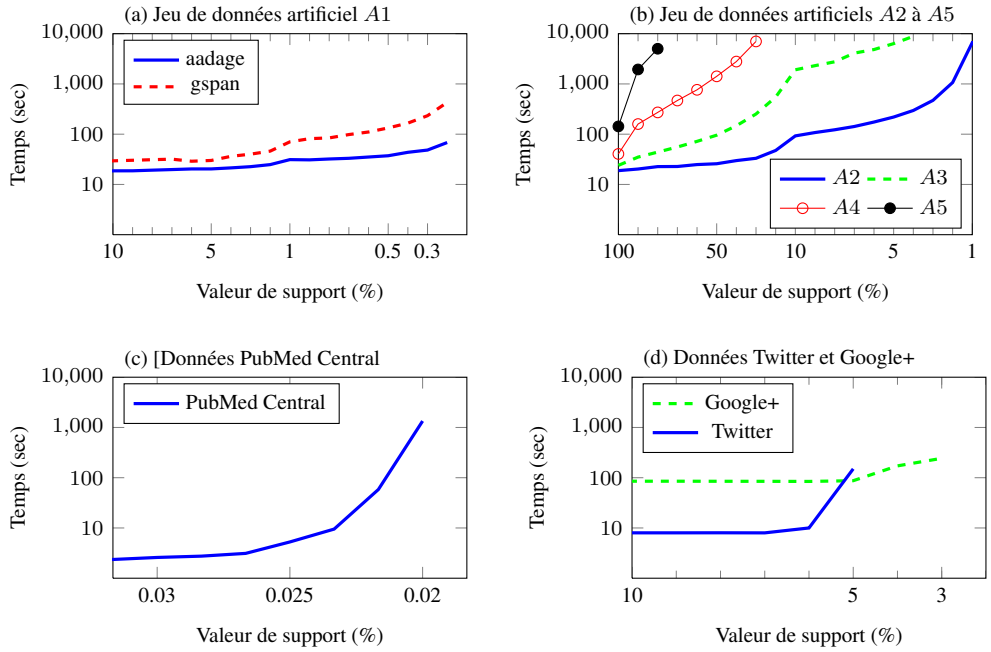


FIG. 5: Performances de l'algorithme sur divers jeux de données.

8 Conclusion et perspectives

Nous avons, dans cet article présenté des méthodes permettant de fouiller des graphes attribués orientés et montré leur efficacité lors de l'analyse de plusieurs jeux de données. Nous nous sommes attachés aux configurations de graphes dans lesquelles, comme dans les données réelles, certains attributs sont partagés par un nombre important de nœuds. Nous avons montré que, selon la structure du motif en cours d'analyse, il n'est pas toujours nécessaire d'étendre toutes les formes que peut prendre le motif. Nous nous sommes intéressés plus particulièrement aux motifs automorphes, mais il existe d'autres motifs pour lesquels des optimisations peuvent être effectuées. Par exemple, lorsque le motif de la figure 1(b), qui ne possède pas d'automorphisme, est trouvé 12 fois sur le graphe de la figure 4(a), ont voit assez facilement que certaines mises en correspondance peuvent être écartées. Nous sommes convaincus que l'analyse détaillée des motifs et de leurs mises en correspondance peut révéler d'autres configurations permettant de réduire l'espace de recherche et ainsi de traiter des jeux de données plus importants ou plus denses. C'est une piste que nous avons l'objectif d'explorer.

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.

Borgelt, C. (2007). Canonical forms for frequent graph mining. In R. Decker et H.-J. Lenz (Eds.), *Advances in Data Analysis*, pp. 337–349. Springer Berlin Heidelberg.

Bringmann, B. et S. Nijssen (2008). What is frequent in a single graph ? In *PAKDD'08*, pp. 858–863.

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

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.

Huan, J., W. Wang, et J. Prins (2003). Efficient mining of frequent subgraphs in the presence of isomorphism. In *ICDM'05*, pp. 549–552.

Jiang, C., F. Coenen, et M. Zito (2013). A survey of frequent subgraph mining algorithms. *Knowledge Eng. Review 28*, 75–105.

Johnsonbaugh, R. et M. Kalin (1991). A graph generation software package. *SIGCSE Bull. 23*(1), 151–154.

McAuley, J. J. et J. Leskovec (2012). Learning to discover social circles in ego networks. In *Neural Information Processing Systems*, pp. 548–556.

Miyoshi, Y., T. Ozaki, et T. Ohkawa (2009). Frequent pattern discovery from a single graph with quantitative itemsets. In *ICDMW'09*, pp. 527–532.

Pasquier, C., J. Sanhes, F. Flouvat, et N. Selmaoui-Folcher (2013). Frequent Pattern Mining in Attributed trees. In *PAKDD'13*, pp. 26–37.

Wörlein, M., T. Meinl, I. Fischer, et M. Philippsen (2005). A quantitative comparison of the subgraph miners mofa, gspan, ffsm, and gaston. In *PKDD'05*, pp. 392–403.

Yan, X. et J. Han (2002). gspan : Graph-based substructure pattern mining. In *ICDM'02*, pp. 721–724.

Summary

Attributed directed graphs are directed graphs in which nodes are associated with sets of attributes. Many data from the real world can be naturally represented by this type of structure, but few algorithms are able to directly handle these complex graphs. Mining attributed graphs is a difficult task because it requires combining the exploration of the graph structure with the identification of frequent itemsets. In addition, due to the combinatorics on itemsets, subgraph isomorphisms, whose presence has a significant impact on the performance of mining algorithms are much more numerous than in labeled graphs. In this paper, we present a new data mining method that can extract frequent patterns from one or more directed attributed graphs. We show how to reduce the combinatorial explosion induced by subgraph isomorphisms in giving special treatment to automorphic patterns.