HAL
archives-ouvertes.fr

AUSMS : Un Environnement pour l'Extraction de Sous-Structures Fréquentes dans une Collection d'Objets Semi-Structurés

Pierre-Alain Laur, Pascal Poncelet

▶ **To cite this version:**

Pierre-Alain Laur, Pascal Poncelet. AUSMS : Un Environnement pour l'Extraction de Sous-Structures Fréquentes dans une Collection d'Objets Semi-Structurés. 02193, 2002, pp.13. <lirmm-00090371>

HAL Id: lirmm-00090371
https://hal-lirmm.ccsd.cnrs.fr/lirmm-00090371

Submitted on 30 Aug 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

AUSMS : un environnement pour l'extraction de sous-structures fréquentes dans une collection d'objets semi-structurés

P.A. Laur
LIRMM
161 rue Ada
34392 Montpellier Cedex 5 France
laur@lirmm.fr

P. Poncelet
Ecole des Mines d'Alès
Site EERIE
Parc Scientifique Georges Besse
30035 Nîmes cedex 1
Pascal.Poncelet@ema.fr

RESUME. La recherche de connaissances dans les données structurées a fait l'objet de nombreux travaux ces dernières années. Cependant, la plupart des approches proposées s'intéressent à des structures plates et, avec la popularité du Web, le nombre de documents semi-structurés disponibles augmente très rapidement. La structure de ces objets est irrégulière et il est judicieux de penser qu'une requête sur la structure des documents est aussi importante qu'une requête sur les données. De plus, les données manipulées ne sont pas statiques parce que de nouvelles mises à jour sont constamment réalisées. Le problème de maintenir de telles sous-structures devient aussi prioritaire que de les rechercher car, au fur et à mesure des mises à jour, les sous-structures trouvées peuvent devenir invalides. Dans cet article, nous proposons un système appelé AUSMS (Automatic Update Schema Mining System), permettant de collecter les données, de rechercher les sous-structures fréquentes et de maintenir les connaissances extraites suite aux évolutions des sources.

ABSTRACT. Mining knowledge from structured data has been extensively addressed in the few past years. However, most proposed approaches are interested in flat structures. With the growing popularity of the Web, the number of semi-structured documents available fastly increases. Structure of these objects is irregular and it is clever to think that a query on documents structure is almost as important as a query on data. Moreover, manipulated data is not static because new updates are constantly realised. Problem of maintaining such sub-structures is then prior on researching them because, every time data is updated, found sub-structures could become invalid. In this paper we propose a system, called A.U.S.M.S. (Automatic Update Schema Mining System), allowing data retrieval, researching frequent sub-structures and maintaining extracted knowledge after sources evolutions.

MOTS-CLES : découverte de sous- structures fréquentes, données semi-structurées, web mining.

KEY WORDS: frequent sub-structures mining, semi-structured data, web-mining.

1 Introduction

La recherche de connaissances dans des données structurées a fait l'objet de nombreux travaux de recherche ces dernières années. La plupart des approches proposées s'intéressent à des structures plates ou fortement structurées. Avec la popularité du Web, le nombre de documents semi-structurés disponibles augmente très rapidement. Contrairement aux applications de bases de données traditionnelles où l'on décrit d'abord la structure des données, i.e. le type ou le schéma, où l'on crée ensuite les instances de ces types, dans les données semi-structurées, les données n'ont pas de schéma prédéfini et chaque objet contient sa propre structure [Bune97]. La majorité des documents « en ligne », tels que les fichiers HTML/XML, Latex, Bibtex, ou SGML sont semi-structurés. Par conséquent, la structure des objets est irrégulière et il est judicieux de penser qu'une requête sur la structure des documents est aussi importante qu'une requête sur les données [WaLi99]. Cependant malgré cette irrégularité structurelle, il peut exister des similitudes structurelles parmi les objets semi-structurés. Il est même assez fréquent de constater que des objets semi-structurés qui décrivent le même type d'information ont des structures similaires. L'analyse de telles structures implicites dans des données semi-structurées peut alors fournir des informations importantes pour : optimiser les évaluations de requêtes, obtenir des informations

générales sur le contenu, faciliter l'intégration de données issues de différentes sources d'information, améliorer le stockage, faciliter la mise en place d'index ou de vues et aider à la classification de documents semi-structurés. Les domaines d'applications sont très nombreux et regroupent par exemple la bio-informatique, le Web Content Mining et le Web Usage Mining. Dans ce dernier cas, les sous-structures découvertes dans les navigations des différents utilisateurs constituent une connaissance très utile pour optimiser dynamiquement l'organisation hypertexte d'un serveur ou elles peuvent être utilisées par un serveur de Proxy dans le but d'améliorer l'accès aux pages.
Récemment de nouvelles approches ont été définies dans ce contexte. Des approches très efficaces ont été proposées pour rechercher de telles sous-structures [WaLi97,WaLi98,MiSh01,AsAb02,Zaki02]. Malheureusement, les données manipulées ne sont pas statiques parce que de nouvelles mises à jour sont constamment réalisées. Le problème de maintenir de telles sous-structures devient alors très important car, au fur et à mesure des mises à jour, les sous-structures trouvées peuvent devenir invalides.

Dans cet article nous nous intéressons à l'extraction de telles sous-structures avec une attention particulière pour leur évolution au cours du temps. Nous proposons un système, appelé AUSMS (Automatic Update Schema Mining System), permettant de collecter les données, de rechercher les sous-structures fréquentes, de maintenir les connaissances extraites suites aux évolutions des sources.

L'article est organisé de la manière suivante. Dans la section 2 nous présentons les problématiques de la recherche de sous-structures fréquentes et de la maintenance des données. La section 3 présente l'architecture fonctionnelle du système en détaillant les différentes étapes. Nous présentons également quelques expériences menées avec le prototype sur des jeux de données réelles issus du Web. Un bref état de l'art sur les approches d'extraction et de maintenance des connaissances est proposé au paragraphe 4. Enfin, dans le paragraphe 5, nous concluons en évoquant les suites de ce travail.

2 Problématique

Dans cette section, nous donnons les définitions formelles liées au problème de la recherche de sous-structures fréquentes dans des objets semi-structurés.

2.1 Définitions

L'objectif de notre proposition est de découvrir des similarités structurelles parmi un ensemble d'objets semi-structurés. Nous considérerons pour la suite un arbre comme un graphe connecté acyclique et une forêt comme un graphe acyclique. Dans notre contexte un graphe cyclique peut être transformé en graphe acyclique qui peut lui même être représenté par un arbre en répliquant les sous fils partagés [WaLi99]. Une forêt est donc une collection d'arbres où chaque arbre est un composant connecté de la forêt. Un arbre ordonné est un arbre enraciné dans lequel les enfants de chaque nœud sont ordonnés. L'ordre est donné en fonction du type d'application et il suit soit l'ordre lexicographique (set-of), soit l'ordre imposé (list-of). Pour exprimer les différences d'ordre nous utiliserons respectivement les notations « *{}* » pour représenter un « ensemble de » et « <> » pour représenter une « liste de ». Nous considérerons pour la suite que nous manipulons des arbres labellisés et ordonnés avec une racine commune. Le fait de nous intéresser à des arbres avec racines et deux types d'ordres nous permet de nous intéresser à différents types de jeux de données classiques comme par exemple les données issues de pages Web.

Exemple : considérons la figure 1 où nous avons deux types d'ordre : le premier est lexicographique (*address*, *id*) où *address* possède les fils suivants: *city*, *street*, *zipcode*, le second est imposé par l'application *name*, *firstname* (en pointillé sur la figue). La représentation de l'arbre est la suivante : *[racine :{address : {city, street, zipcode}, id : <name, firstname>}]*.

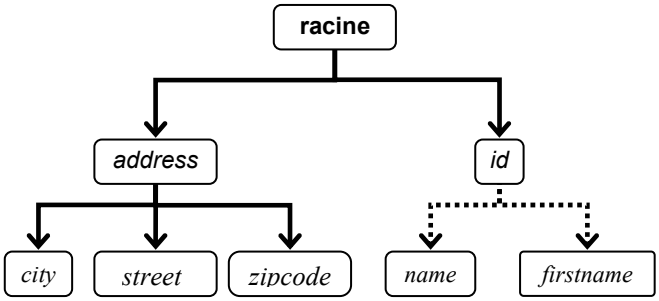


Figure 1 – Un exemple d'arbre

Considérons un nœud x dans un arbre T de racine r. Un nœud y du chemin unique de la racine r à x est appelé un ancêtre de x et est noté $y \leq_k x$, où k est la longueur du chemin de y à x. Si y est un ancêtre de x, alors x est un descendant de y (chaque nœud est à la fois son ancêtre et son descendant). Si $y \leq_1 x$, i.e. y est un ancêtre direct, alors y est appelé le parent de x et x le fils d'y.

Soit T un arbre tel que $T = (N, B)$, où N est l'ensemble des nœuds labellisés et B l'ensemble des branches. Nous considérons qu'un arbre $S=(N_s, B_s)$ est un arbre imbriqué dans T, noté $S \leq_{ST} T$ si et seulement si : $N_s \leq N$ et $b=(n_x, n_y) \in B_s$ si et seulement si $n_y \leq_1 n_x$, i.e. n_x est le fils de n_y dans T. Si $S \leq_{ST} T$, nous disons que T contient S ou que S est une sous-structure de S.

Exemple : par exemple, l'arbre *{address : {city, street, zipcode}, category, name}* est un sous arbre de *{address : {city, street, zipcode}, category, name, nearby : {category, name, price}}*. Cependant l'arbre *{address : {city, street, zipcode}, category, name, price}* n'est pas un sous arbre car l'élément *price* n'est pas au même niveau dans le graphe comme l'indique la figure 2.

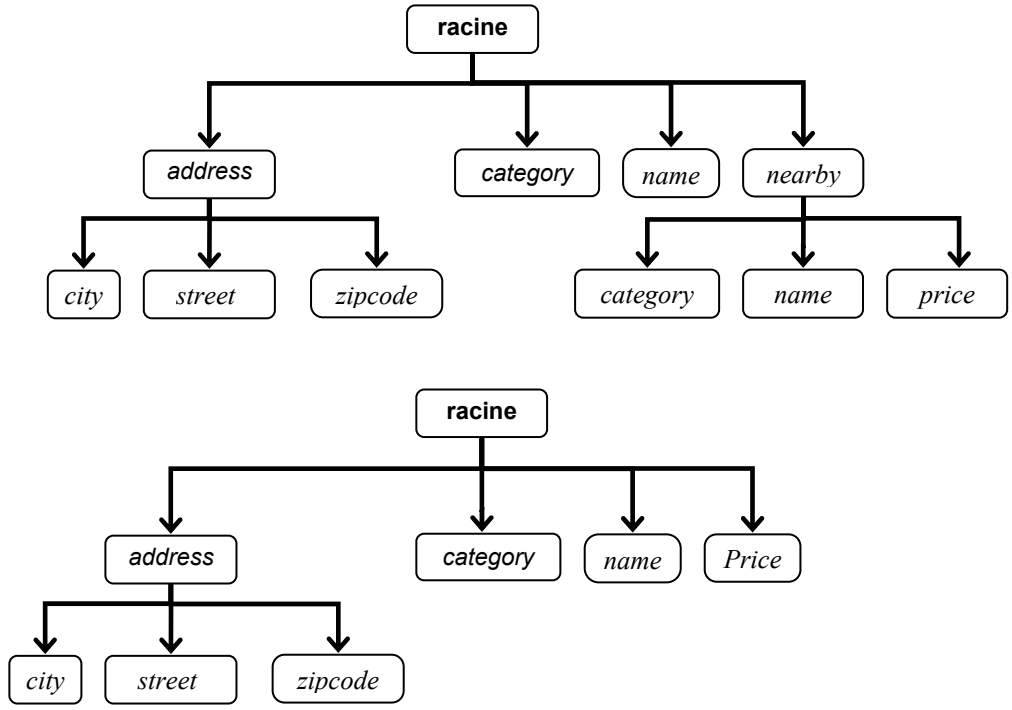


Figure 2 – Inclusion de sous arbre

2.2 Problématique

Soit *DB* une base de données d'arbres appelés aussi structures, i.e. une forêt où chaque arbre T est composé d'un identifiant et d'une structure incluse dans la forêt. Tous les arbres sont triés soit par ordre lexicographique, soit par un ordre imposé (list-of). La figure 3 illustre un exemple d'une base de données. Soit *supp(p)*, une valeur de support pour une structure correspondant au nombre d'occurrences de cette structure dans la base de données *DB*. En d'autres termes, le support d'une structure p est défini comme le pourcentage de tous les arbres de la base qui contiennent p. Un arbre de la base contient p si et seulement si p est une sous-structure de l'arbre. Afin de décider si une structure est fréquente ou non, une valeur de support minimal est spécifiée par l'utilisateur (*minSupp*) et une structure est fréquente si la condition $supp(p) \geq minSupp$ est vérifiée.

Etant donnée une base de données *DB* d'arbres, le problème de la recherche de régularités dans des données semi-structurées consiste donc à rechercher toutes les structures maximales qui sont dans *DB* et dont le support est supérieur à *minSupp*.

Trans_id	Structure
t_1	*[person :{identity :{address, name}}]*
t_2	*[person :{identity : { address : <street , zipcode >, company, director : <name, firstname >, name}}]*
t_3	*[person : {identity : { address : <street, zipcode>, id }}]*
t_4	*[person :{identity : { address , company, name}}]*
t_5	*[person : { identity : {address, name}}]*
t_6	*[person : {identity : { address : < street , zipcode>, director : <name, firstname >, name }}]*

Figure 3 – Une exemple de base de données de sous tructures

Exemple : pour illustrer le problème de la recherche de régularités dans des données semi-structurées, considérons la base DB de la figure 3. Supposons que la valeur du support spécifiée par l'utilisateur soit 50%, i.e. pour être fréquente, une sous-structure doit apparaître dans au moins trois arbres. Les seules structures fréquentes dans DB sont les suivantes : *[identity : {address, name}]* et *[identity : {address : <street, zipcode>}]*. Le premier car il est vérifié dans t_1 mais également dans t_4 et t_5. Par contre, la structure *[identity : {address : <street, zipcode>, director : <name, firstname>, name}]* est vérifiée par les t_2 et t_6 mais n'est pas fréquente puisque le nombre d'arbres qui possède cette structure est inférieur à la contrainte du support minimal.

Considérons à présent l'évolution des sources de données. Soit *db* la base de données incrément où de nouvelles informations sont ajoutées ou supprimées. Soit $U=DB \cup db$, la base de données mise à jour contenant toutes les structures de *DB* et *db*. Soit L^{DB}, l'ensemble de toutes les sous-structures fréquentes dans *DB*. Le problème de la maintenance des connaissances est de rechercher les sous-structures fréquentes dans *U*, noté L^U, en respectant la même valeur du support. En outre, la maintenance doit tenir compte des connaissances extraites précédemment de manière à éviter de relancer des algorithmes d'extraction quand les données sont mises à jour.

3 Le Système A.U.S.M.S.

Le but du système A.U.S.M.S. (Automatic Update Schema Mining System) est de proposer un environnement de découverte et d'extraction de connaissances pour des données semi-structurées depuis la récupération des informations jusqu'a la mise à jour des connaissances extraites. Ces principes généraux illustrés par la figure 4 sont assez similaires à ceux d'un processus d'extraction de connaissances. La démarche se décompose en trois phases principales. Tout d'abord à partir de fichiers de données semi-structurés brutes, un pré-traitement est nécessaire pour éliminer les données inutiles et assurer leur transformation. Dans la seconde phase, un algorithme d'extraction de connaissances est utilisé pour extraire les sous-structures fréquentes. De manière à permettre la maintenance des connaissances extraites, les informations obtenues lors de la phase d'extraction sont maintenues dans une base de données. Enfin, l'exploitation par l'utilisateur des résultats obtenus est facilitée par un outil de visualisation des sous-structures fréquentes.

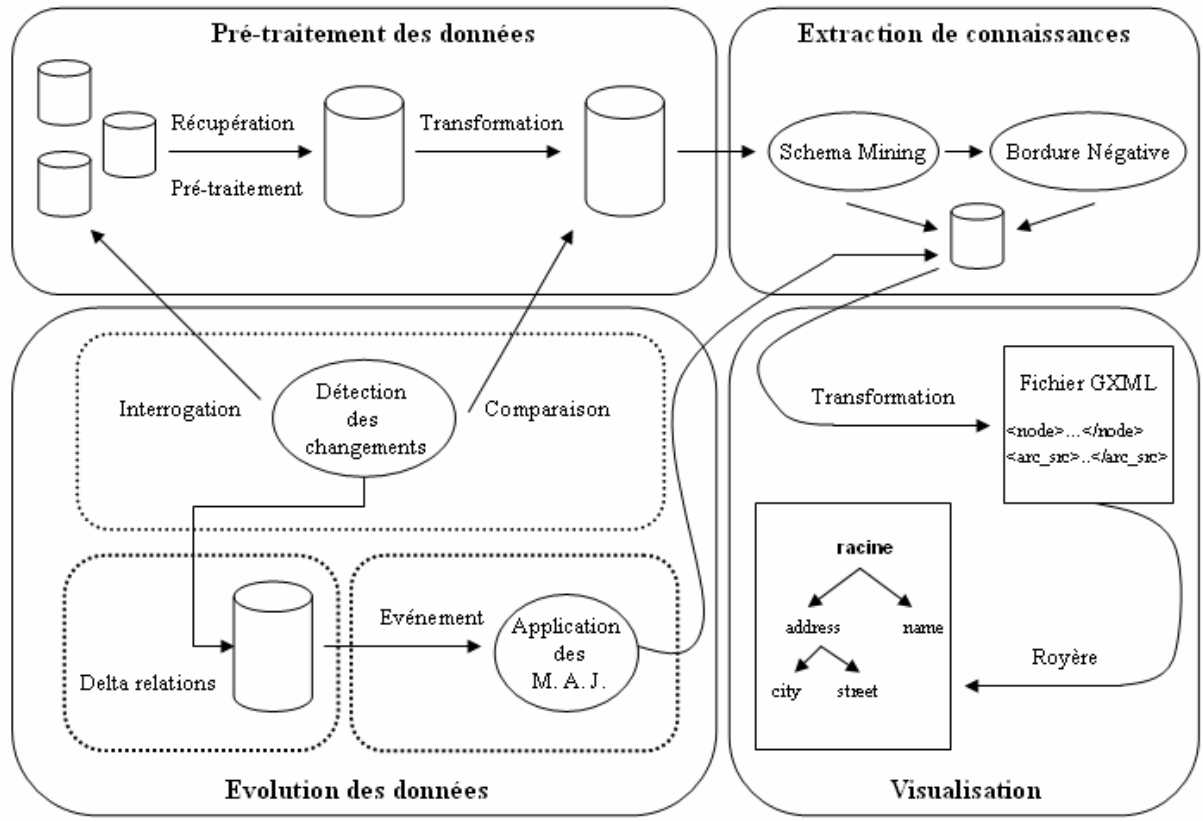


Figure 4 – Architecture générale

Les différentes phases introduites sont détaillées dans les paragraphes suivants.

3.1 Pré-trairement des données

A partir des sources, une étape d'extraction et de transformation est réalisée et les données extraites sont stockées dans une base de données. Dans le cadre de données issues du Web une étape de filtrage est réalisée de manière à éliminer les données qui ne sont pas utiles pour l'analyse : image, sons, vidéo, …. En outre, en fonction du point de vue de l'utilisateur les sous-structures « non intéressantes » sont également supprimées.
Pour chaque arbre extrait, nous lui associons un identifiant qui servira de clé principale et chaque structure extraite est transformée avant d'être stockée. De manière à conserver les niveaux d'imbrications des différents arbres, la transformation est opérée de la manière suivante :

- Prise en compte de la profondeur d'imbrication et du type complexe : chaque élément extrait est considéré et pour prendre en compte le niveau d'imbrication d'un élément, un entier décrivant la profondeur d'imbrication de cet élément dans la structure complexe, est ajouté à l'élément.
- Création de listes d'ensembles d'éléments simples : quand deux éléments sont au même niveau et si le premier est directement suivi par le second, nous les regroupons dans un même ensemble autrement ils sont inclus dans deux ensembles séparés. La notion d'ordre de la valeur « liste-de » est, par contre, prise en compte en créant de nouveaux ensembles entre les éléments. La transaction composite qui résulte de l'union de ces ensembles obtenus à partir des transactions initiales décrit une séquence d'éléments simples modifiés et l'ordre de cette séquence peut alors être perçu comme une navigation en « profondeur d'abord » des transactions.

A l'issue de cette phase, les différentes structures sont stockées dans la base de données

Exemple : Pour illustrer la phase de transformation, considérons les deux arbres suivants : $t_1 = \{a, \{c, \{d, f\}\}\}$ et $t_2 = \{a, <e, \{b,f\},d,<,h\ g>>,c\}$. Pour l'illustration, chaque élément est précédé respectivement de la lettre *S* pour « set-of » et la lettre *L* pour « list-of ». Dans le premier arbre, étant donné que tous les éléments simples apparaissent dans un ensemble de valeurs, le symbole *S* leur est affecté. Concernant le niveau d'imbrication de la transaction nous affectons à chaque élément simple son niveau par rapport à l'ensemble le plus haut de la hiérarchie. Les éléments simples *a*, *b*, *c*, *d* et *f* sont alors transformés de la manière suivante : Sa_1, Sc_2, Sd_3, Sf_3.

En affectant chaque élément simple dans un nouvel ensemble et en parcourant en profondeur d'abord la structure t_1, nous obtenons la transformation suivante : $t_1 = (Sa_1)\ (Sc_2)\ (Sd_3\ Sf_3)$. En appliquant le même principe pour t_2, nous obtenons : $t_2 = (Sa_1)\ (Le_2)\ (Sb_3\ Sf_3)\ (Ld_2)\ (Lh_3)\ (Lg_3)\ (Sc_1)$. Nous pouvons constater que la modification de t_2 respecte l'ordre de parcours en profondeur d'abord. Examinons en détail la partie « list-of » < *g, h*> de t_2. Comme les éléments *g* et *h* sont ordonnés à cause de la valeur « set-of », nous considérons que même s'ils interviennent au même niveau, et même si *h* suit directement *g*, ils ne peuvent pas être regroupés dans un même ensemble.

3.2 Extraction de connaissances

Nous avons montré dans [Laur00,LaMa00] qu'il existait une bijection entre la problématique de la recherche de sous-structures telle que nous l'avons définie et celle de la recherche de motifs séquentiels définie dans [AgSr95]. Pour rechercher les structures fréquentes dans la base obtenue lors de la phase précédente, nous utilisons un algorithme largement inspiré de ceux définis pour la recherche des motifs séquentiels et dont les principes généraux sont expliqués ci dessous :

Algorithme d'extraction
input : un support minimal (minSupp), une base de données DB
output : l'ensemble L des structures fréquentes maximales qui vérifient la contrainte de support minimal et un graphe G constituant la bordure négative
k = 1 ;
C1 = {{i}/ i ∈ ensemble d'éléments atomiques transformés par la phase précédente}
while ($C_k \neq \varnothing$) **do**
    for each d ∈ D **do** VerifyCandidate (d,k) ;
        L_k = { c ∈C_k/support (c) ≥ minSupp} ;
        k += 1 ;
        CandidateGeneration (k) ;
        GenerateBN(G,k) ;
return L^{DB} où L^{DB} est l'union de j=0 à k des L_j

De manière générale, l'algorithme effectue un parcours de DB pour déterminer quels sont les éléments qui interviennent assez régulièrement pour être retenus. A partir des structures de taille 1 qui vérifient le support, nous générons des structures de taille 2 qui sont appelées des structures candidates. Un nouveau parcours sur la base permet de retenir toutes les structures candidates de taille 2 dont le nombre d'occurrences est supérieur au support minimal. Ensuite, l'algorithme continue de la manière suivante : à chaque étape k, la base est parcourue pour compter le support des candidats (procedure VerifyCandidate). A partir des candidats dont le nombre d'occurrences est supérieur au support minimal, l'ensemble des structures fréquentes est construit : L^k. A partir de cet ensemble, de nouveaux candidats peuvent être construits (procedure CandidateGeneration). L'algorithme s'arrête quand la procédure de génération des candidats fournit un ensemble vide ou que la procédure VerifyCandidate retourne un ensemble vide de fréquents.

Dans le but d'améliorer la procédure de génération des candidats ainsi que la gestion des éléments candidats, nous utilisons une représentation par bitmap inspirée de [ArGe02]. Cette structure offre l'avantage de diminuer considérablement l'espace de stockage et nous offre la possibilité de générer facilement les candidats. En outre elle est particulièrement adaptée à la recherche de structures longues.
Lors de la recherche de candidats nous générons également la bordure négative [MaTo96]. Celle-ci est composée de toutes les structures qui ne sont pas fréquentes mais dont les sous-structures sont fréquentes. Cette bordure sera utilisée dans la phase suivante de prise en compte de l'évolution des données sources.

Exemple : considérons la figure 5 représentant le treillis associé à la base exemple. Pour un support minimal de 50 %, au niveau 1, seuls les éléments A_1, A_2 et B_3 sont fréquents et peuvent être utilisés pour créer des structures plus complexes. Nous stockons donc dans la bordure négative, les éléments B_2, C_3 et D_2. Au niveau 2, seuls $(A_1)\ (A_2)$, $(A_1)\ (B_2)$, $(A_2\ B_2)$ sont fréquents, nous conservons dans la bordure négative les éléments dont les éléments du niveau précédent étaient fréquents.

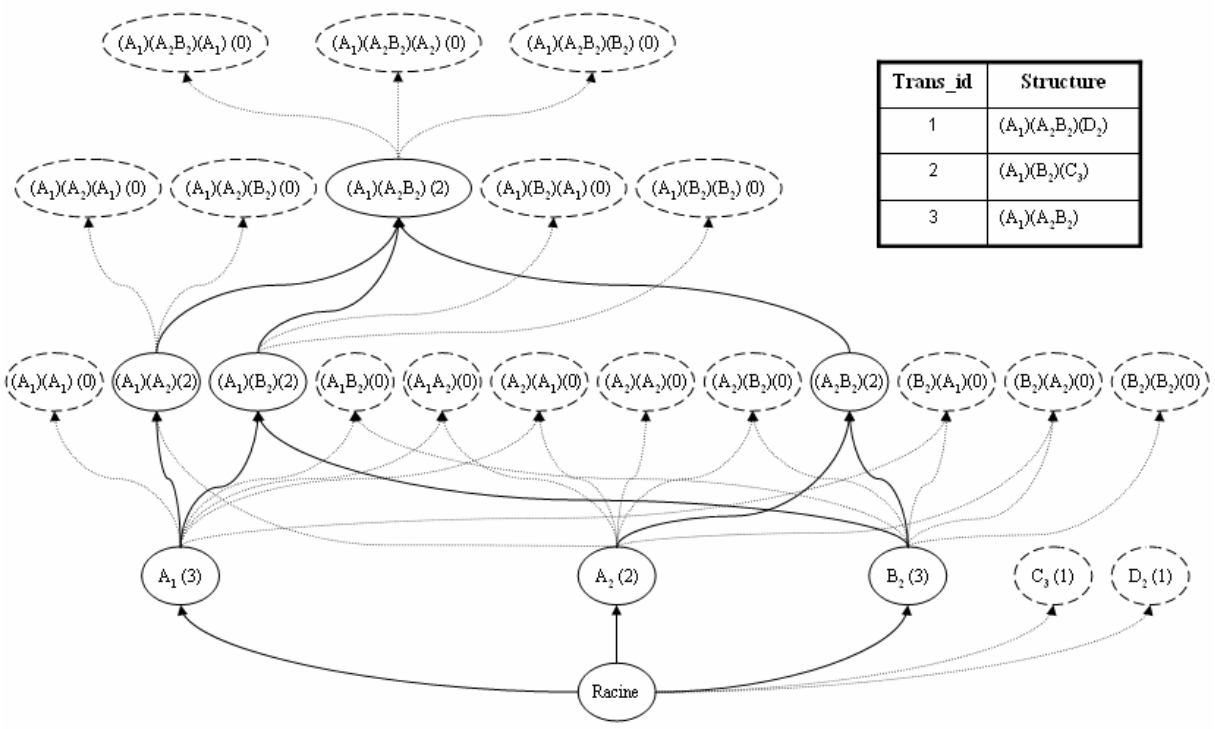


Figure 5 – Un exemple de bordure négative

3.3 Prise en compte de l'évolution des données sources

La bordure négative obtenue lors de l'étape précédente nous permet de tenir compte des mises à jour et de maintenir la connaissance extraite. En effet, pour éviter de ré-appliquer l'algorithme précédent lors de chaque mise à jour, nous stockons dans la bordure négative l'information minimale pour calculer rapidement les sous-structures fréquentes. La prise en compte de l'évolution des données sources suit les principes généraux qui sont expliqués ci-dessous :

Algorithme de mise à jour
input : S l'ensemble des sources de données, BN la bordure négative, BN^{Limit}, L^{DB} l'ensemble des structures fréquentes, minSupp le support minimal spécifié par l'utilisateur
output : les sources S mises à jour, BN mise à jour et L^{DB} mis à jour
while t ∈ delay **do**
foreach s ∈ S **do**
if $s_{new} \neq s_{old}$ **then**
updateDeltaRelation (Δ_s, opmaj, t)
enddo

$$\Delta_S \leftarrow \bigcup_{i=0}^{S} \Delta_s$$

if Validate(Δ_S, BN^{Limit}) **then**
Update (L^{DB})

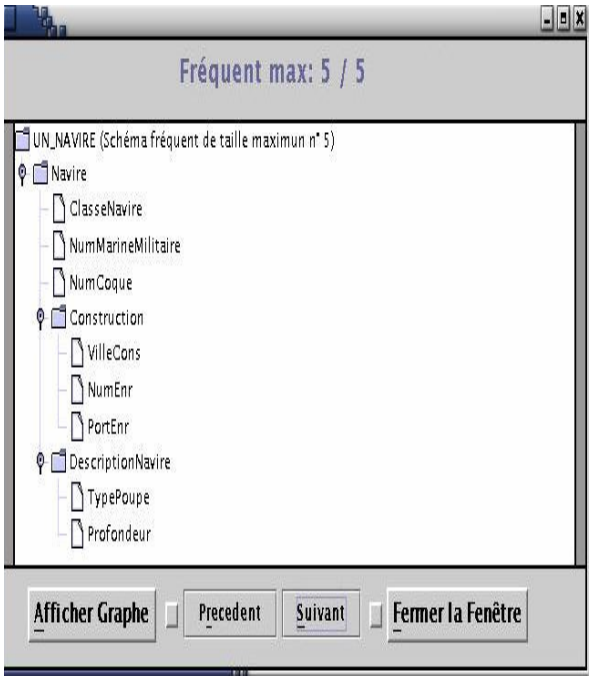
A partir d'un délai spécifié par l'utilisateur (delay), les sources de données sont comparées (s_{old} représente les données sources initiales, i.e. lors de la dernière analyse et s_{new} représente les données en cours d'analyse). Cette opération est effectuée dans le système AUSMS par un agent qui agit soit de manière temporelle (durée écoulée depuis le dernier déclenchement), soit de manière directe (choix de l'utilisateur). L'agent est chargé de comparer les données sources et de propager les modifications. Ainsi, si la source de données a été modifiée, les mises à jour sont stockées dans l'ensemble Δs qui gère l'historique des modifications (procedure UpdateDeltaRelation).

Cette procédure, inspirée des delta relations des règles actives, permet de refléter les effets de bord des modifications de la structure, i.e. elle ne contient que l'effet de bord résultant des modifications [ChAb98].

A partir des informations contenues dans l'ensemble Δ_S, une comparaison est effectuée, par la procédure Validate (Δ_S, BN^{Limit}) , avec les éléments contenus dans la bordure négative qui sont susceptibles de changer rapidement, i.e. ceux qui peuvent devenir fréquents ou non à un élément près. Cette procédure tient également compte de l'ajout ou de la suppression de nouvelles sources qui engendrent bien sûr une modification de la valeur de support. Si l'une des conditions est vérifiée alors les modifications sont apportées directement dans la bordure négative pour mettre à jour l'ensemble des structures fréquentes (procedure Update (L^{DB})). Faute de place, nous ne détaillons pas ici cet algorithme (le lecteur intéressé peut se reporter à [Laur02]) mais nous en donnons les principes généraux ci-dessous.
La première étape consiste à reporter les modifications dans la bordure négative dès que des structures sont ajoutées ou supprimées. En effet, une telle opération a pour effet de modifier le calcul de la valeur du support sur toute la base. Pour chaque structure, nous examinons donc la valeur dans la bordure négative et si celle-ci est inférieure au support, les branches de l'arbre issues de cette structure sont élaguées. Autrement les autres éléments sont ré-examinés et la bordure négative est mise à jour en fonction de leur fréquence. Lorsque les opérations consistent en l'ajout ou la suppression d'élément dans des structures existantes, nous analysons la bordure négative en commençant par le niveau 1 de manière à valider ou non la fréquence d'apparition des éléments. S'ils deviennent fréquents les différents niveaux du treillis sont construits récursivement avec ceux qui étaient déjà fréquents. S'ils ne deviennent plus fréquents les différentes branches du treillis issus de la sous-structure sont élaguées. A l'issue de cette phase, les éléments fréquents sont extraits et L^{DB} est mis à jour.

3.4 Visualisation

Alors que les modules précédents sont chargés de fournir et de maintenir des sous-structures fréquentes, ce module permet de visualiser ces structures et offre un formalisme pour les décrire. Pour cela, nous utilisons, en premier lieu, GraphXML [HeMa00] qui est un langage de description de graphe en XML spécialement étudié pour des systèmes de visualisation et de dessin. GraphXML, en plus de fournir un langage de description permet à l'utilisateur de rajouter de nombreuses informations aux graphes manipulés : date, couleur des arcs, information semi-structurée (pour un arc, graphe, nœud).
En second lieu, de manière à visualiser à la fois les structures extraites mais également leur apparition dans les données sources, nous utilisons « Graph Visualisation Framework » [MaHe00] qui propose un ensemble de classes java pour visualiser et manipuler les structures. Ce système, via une application nommée Royère, permet l'affichage des structures décrites au format GraphXML.




```
<?xml version="1.0"?>
<!DOCTYPE GraphXML SYSTEM "file:GraphXML.dtd" [
<!ENTITY % admissibleProperties "
support CDATA #IMPLIED
nombreTid CDATA #IMPLIED
dateCalcul CDATA #IMPLIED
nomBase CDATA #IMPLIED ">
]>
<GraphXML>
<graph isDirected="true" isForest="true"
isAcyclic="true" "referredlayout=
"ReingoldTilford" id="Frequent" version="1.0"
vendor="ausms@lirmm.fr" isPlanar="true">
<support>85</support> <nombreTid>728</nombreTid>
<dateCalcul>031002</dateCalcul>
<nomBase>navire</nomBase>

<node name="1">
<label>Navire</label>
</node>
<node name="2">
<label>ClasseNavire</label>
</node>
<edge source="1" target="2">
</edge>
        … … … … …
<node name="12" class="std">
<label>Profondeur</label>
</node>
<edge source="10" target="12">
</edge>
</graph> </GraphXML>
```

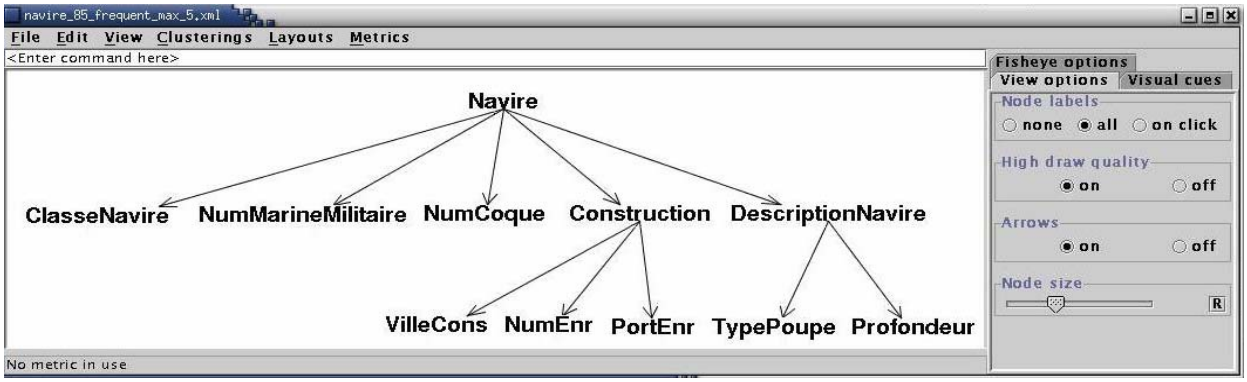


Figure 6 – Exemples de structures extraites

Considérons la figure 6 qui représente des copies d'écran de structures visualisées. Nous trouvons en haut et à gauche une structure fréquente issue de la recherche de sous-structures fréquentes à au moins 85% d'une base de données d'historique de navires. A droite nous avons la description au format GraphXML de cette structure et en dessous la structure affichée via Royère.

3.5 Illustration

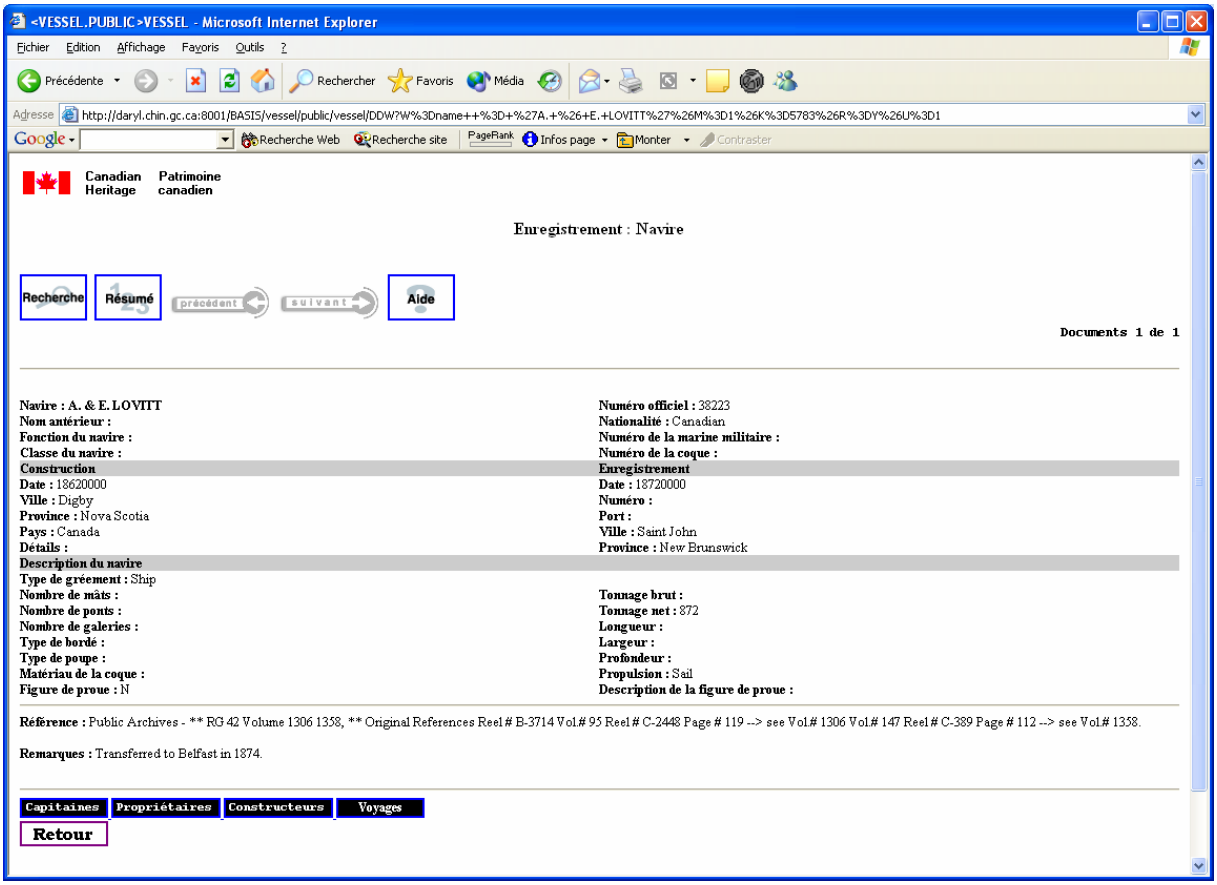


Figure 7 – Une base de données exemple

Pour présenter l'utilité de notre approche, nous illustrons ci-dessous une application sur la recherche de sous-structures fréquentes dans un ensemble de documents semi-structurés. Les données utilisées sont issues d'une base de données canadienne située à l'adresse http://daryl.chin.gc.ca:8001/basisbwdocs/sid/title1f.html . Cette

base a été créée pour répondre aux besoins des gestionnaires des ressources culturelles chargés de l'information sur les épaves archéologiques. Elle contient des renseignements sur des navires qui ont été immatriculés au Canada ou qui ont navigué dans les eaux canadiennes. Elle se divise en cinq sous bases : Navires (728 transactions), Capitaines (473 transactions), Propriétaires (485 transactions), Constructeurs navals (474 transactions), Voyages (4215 transactions). Pour chacune des ces bases, des structures différentes existent. Par exemple, la figure 7 illustre une partie de la base Navire qui contient de nombreuses informations comme par exemple : l'identification du navire, l'identification antérieure, des informations sur le tonnage, la voilure.... De manière générale, les informations contenues dans la base ont une profondeur variant de 2 à 10. Dans la base de données, il y a en outre des informations incomplètes, i.e. tous les champs ne sont pas renseignés.

Pour intégrer cette base à notre prototype nous avons écrit dans un premier temps des analyseurs capables, après rapatriement des sources, de convertir celle-ci en une base de structure sur laquelle nous pouvons appliquer les principes présentés dans la section 3. Ainsi, pour un support de 85 %, nous avons trouvé 5 fréquents de taille maximum (l'un des fréquents est présenté dans la figure 6). Pour un support de 100% sur cette même sous base il n'existe plus qu'un seul fréquent de taille maximum : *[Navire :{ClasseNavire, Construction :{NumEnr}, DescriptionNavire{TypePoupe}, NumCoque, NumMarineMilitaire}]*. Ce schéma représente les informations disponibles pour la totalité des navires. Nous apprenons par exemple que pour les capitaines, seule l'information relative au nom est disponible pour la totalité de ceux-ci. Ce n'est que pour un support de 60% que l'on obtiendra deux schémas de fréquents maximum différents : *[Capitaine :{Nom, Pays}]* et *[Capitaine : {Nom, NumeroCertificat}]*. La totalité des informations n'est renseignée que pour 119 capitaines (soit un support de 25%). Pour tester la mise à jour des connaissances en fonction des modifications des données sources, nous avons simulé l'ajout à certaines sources de nouvelles informations comme par exemple la connaissance des pays d'origine des capitaines. Après mise à jour des éléments de la bordure négative, nous avons trouvé une seule nouvelle sous-structure fréquente maximale pour un support de 60% : *[Capitaine :{Nom, Pays, NumeroCertificat }]*.

4 Travaux antérieurs

A notre connaissance, il existe peu de travaux concernant la recherche de régularités structurelles dans de grandes bases de données. Néanmoins, notre approche est très proche de celle proposée dans [WaLi97, WaLi98, WaLi99] pour la recherche d'association structurelle dans des données semi-structurées. Les auteurs proposent une approche très efficace et des solutions basées sur une nouvelle représentation de l'espace de recherche. En outre, en proposant des optimisations basées sur des stratégies d'élagages, ils améliorent considérablement l'étape de génération des candidats. De la même manière, l'approche proposée dans [MiSh01] est assez similaire à l'approche précédente et utilise un motif d'arbre particulier appelé tag tree patterns. Dans [AsAb02], les auteurs proposent un algorithme appelé Find-Freq-Trees qui utilise également une approche basée sur une recherche par niveau comme dans l'algorithme Apriori [AgIm93] et étend la proposition en améliorant la technique d'énumération définie dans [Baya98] de manière à découvrir des sous-structures dans de longues séquences. Enfin dans [Zaki02], l'auteur propose deux algorithmes TreeMinerH et TreeMinerV pour la recherche d'arbres fréquents dans une forêt. TreeMinerH reprend le principe du parcours en largeur de A-priori en améliorant la génération et le comptage des candidats à l'aide des classes d'équivalences, d'une structure d'arbre préfixé et de « scope list ». Quand à TreeMinerV, il propose de voir un arbre comme une structure verticale et associe à cette vision une méthode de parcours en profondeur très efficace pour la recherche de séquence longue. Dans ces deux algorithmes, la génération et le comptage des candidats sont effectués par des opérations ensemblistes sur les « scope list », la structure préfixée permet de réduire le nombre de transactions à parcourir dans la base de données. Outre la prise en compte des évolutions, notre approche a cependant certaines différences. Nous nous intéressons à la recherche de toutes les structures incluses dans la base alors qu'ils ne s'intéressent qu'à la recherche de tree-expression qui sont définis comme des arbres allant de la racine à une feuille terminale de l'arbre. Avec cette définition, ils ne peuvent pas trouver des régularités de la forme : *[identity : {address : <street, zipcode>}]* qui seraient fréquentes mais seraient incluses dans une transaction plus longue qui elle n'est pas fréquente. L'algorithme de recherche de sous-structure utilisé est basé sur une représentation par vecteurs de bits qui nous permet également de travailler sur de longues structures.
D'autres méthodes de recherche pour des structures d'arbre ou de graphes sont également proposées mais ne sont pas directement applicables à la fouille de données semi structurées. Ainsi, les auteurs de [WaSh96], proposent un algorithme de découverte de structures communes approximatives et l'appliquent à la découverte d'applications génomiques. Dehaspe et al. [DeTo98] présentent un algorithme efficace pour résoudre le problème de la découverte de sous-structures fréquentes dans des graphes labellisés. Leur approche est basée sur l'utilisation d'ILP. Dans [MaHo99], un algorithme est également proposé pour extraire des motifs d'un graphe dirigé.

En ce qui concerne la maintenance des sous-structures fréquentes extraites, il n'existe pas à notre connaissance de travaux dans ce domaine. Nous avons montré que la recherche de sous-structures pouvait se rapprocher de celle de motifs séquentiels. Nous examinerons donc dans la suite les travaux effectués autour de ce domaine. Proche des motifs séquentiels et à l'origine de nombreuses approches, [ChHa96] propose un algorithme appelé FUP pour une fouille de données incrémentales dans le cas des règles d'associations. Cependant, la problématique de mise à jour incrémentale dans le cadre des motifs séquentiels est beaucoup plus complexe que celle des règles d'associations dans la mesure où l'espace de recherche, i.e. le nombre de combinaisons est beaucoup plus grand. Dans [PaZa99], les auteurs proposent un algorithme appelé ISM (Incremental Sequence Mining) basé sur l'approche SPADE [Zaki98], qui permet une mise à jour des séquences fréquentes quand de nouveaux clients et de nouvelles transactions sont ajoutés à la base de données. L'approche proposée construit un treillis de séquence qui contient tous les fréquents et les éléments de la bordure négative [MaTo96]. Quand de nouvelles informations arrivent, elles sont ajoutées à ce treillis. Le problème de cette approche est évidemment la taille croissante de la bordure négative qui dans notre cas est minimisée car basée sur des vecteurs de bits. Dans [MaPo00], l'algorithme ISE (Incremental Sequence Extraction) a été proposé pour la recherche de motifs fréquents, il génère des candidats dans toute la base de données en attachant les séquences de la base de données incrémentale à ceux de la base originale. Cette approche évite de garder les séquences contenues dans la bordure négative et le recalcul de ces séquences quand la base de données initiale à été mise à jour. Cependant, en ne conservant pas la bordure négative, il est nécessaire de parcourir plus souvent la base pour rechercher les candidats. Dans [ZhXu02] l'algorithme proposé utilise à la fois les notions de bordure négative de la base de données originelle et des notions de suffixes et préfixes contrairement à ISE. Pour contrôler la taille de cette bordure négative, ils introduisent un support minimum pour ces éléments réduisant ainsi la taille de celle-ci. De plus cet algorithme réalise une extension par préfixe et par suffixe (à l'aide de la bordure négative). Le problème de cet algorithme réside dans le choix de la valeur du support minimum pour la bordure négative.

5 Conclusion

Dans cet article, nous avons proposé une architecture fonctionnelle, AUSMS, d'un système d'extraction et de maintenance des connaissances dans des bases d'objets semi-structurées. L'originalité de l'approche réside dans la mise en œuvre d'algorithmes efficaces pour extraire les sous-structures fréquentes dans la base d'objets semi-structurées mais aussi dans la prise en compte des données manipulées. L'utilisation de vecteurs de bits pour l'extraction et pour la gestion de la bordure négative nous permet également d'optimiser le stockage et la recherche des structures. Les tests que nous avons menés sur des bases issues du Web ont montré que l'approche retenue était très utile pour aider l'utilisateur final dans l'analyse des différents éléments manipulés et offrait des solutions pour la recherche d'informations générales sur les sources de données, pour aider à l'interrogation de bases contenant des données semi-structurées et pour aider à construire des vues et des index.

Nous sommes actuellement en train d'étudier l'application d'AUSMS dans la gestion de données issues de serveurs Web (Web Usage Mining) où nous souhaitons analyser le comportement complet des utilisateurs. Même si actuellement, une entrée dans un fichier « access log » est automatiquement ajoutée à chaque fois qu'une requête pour une source atteint le serveur, elle n'enregistre cependant pas certains comportements de l'utilisateur comme un retour en arrière fréquent ou le rechargement d'une page lorsque les pages sont cachées par le navigateur ou un Proxy. Par exemple, le fait qu'un utilisateur soit obligé de régulièrement revenir en arrière peut indiquer une mauvaise conception de la navigation du serveur et de telles informations sont importantes pour améliorer la conception d'un site. Pour cela, nous avons mis au point une application locale à l'utilisateur et qui stocke son comportement pour le transmettre à une base de données. L'idée consiste donc à coupler les informations obtenues dans le fichier log du navigateur à la base de données pour servir d'entrée au processus d'AUSMS. Un autre axe de recherche que nous menons actuellement concerne la mise en parallèle des algorithmes d'extraction de manière à optimiser la recherche des sous-structures fréquentes.

6 Références

[AgIm93] R. Agrawal, T. Imielinski, and A. swami, “Mining Association Rules between Sets of Items in Large Databases “, Proceedings of the International Conference on Management of Data (SIGMOD’93), pp. 207-216, Washington DC, USA, May 1993.

[AgSr95] R. Agrawal and R. Srikant, “Mining Sequential Patterns”, Proceedings of International Conference on Data Engineering (ICDE'95), pp. 3-14, Tapei, Taiwan, March 1995.

[ArGe02] J. Ares, J. Gehrke, T. Yiu and J. Flannick, “ Sequential Pattern Using Bitmap Representation “, Proceedings of Principles and Practice of Knowledge Discovery in Data (PKDD’02), Edmonton, Canada, July 2002.

[AsAb02] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto and S. Arikawa, “Efficient substructure discovery from Large Semi-structured Data”, Proceedings of the International Conference on Data Mining (ICDM’02), Washington DC, USA, April 2002.

[Baya98] R. J. Bayardo Jr, “Efficiently Mining Long Patterns from Databases”, Proceedings of the International Conference on Management of Data (SIGMOD’98), pp. 85-93, Seattle, USA , June 1998.

[Bune97] P. Buneman, “Semistructured Data”, Proceedings of Symposium on Principles of Database Systems (PODS’97), pp. 117-121, Tucson, USA, May 1997.

[ChAb98] S. Chawathe, S. Abiteboul and J. Widom, “Representing and Querying Changes History in Semistructured Data ", Proceedings of the International Conference on Data Engineering (ICDE98), Orlando, USA, February 1998.

[ChHa96] D. W. Cheung, J. Han, V. Ng and C. Y. Wong, “Maintenance of Discovered Association Rules in Large Databases: an Incremental Update Technique”, Proceedings of the International Conference on Data Engineering (ICDE’96), pp. 116-114, New Orleans, USA, February 1996.

[DeTo98] L. Dehaspe, H. Toivonen and R. D. King, “Finding Frequent Substructures in Chemical-compounds”, Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD’98), pp. 30-36, New York, USA, August 1998.

[HeMa00] I. Herman and M.S. Marshall, “GraphXML An XML based graph interchange format”, Centre for Mathematics and Computer Sciences (CWI), Technical Report INS-R0009, pp. 52-62, Amsterdam, 2000.

[LaMa00] P.A. Laur, F. Masseglia and P. Poncelet, “A General Architecture for Finding Structural Regularities on the Web”, Proceedings of the International Conference on Artificial Intelligence (AIMSA'00), September 2000.

[Laur00] P.A. Laur, “Schema Mining: vers une approche efficace“, Mémoire de DEA, LIRMM, Montpellier, France 2000.

[Laur02] P.A. Laur, « Mise à jour des motifs séquentiels : une approche basée sur la bordure négative » , rapport interne, LIRMM, Montpellier, 2002.

[MaHe00] M. Marshall, I. Herman and G. Melancon, “An Object-oriented Design for Graph Visualization”, Technical Report INS-R00001, Centre for Mathematics and Computer Sciences, Amsterdam, 2000.

[MaHo99] T. Matsuda, T. Horiuchi, H. Motoda, T. Washio, K. Kumazawa and N. Arai. “Graph Based Induction for General Graph Structured Data”. In Proceedings of the DS’99 Conference, pp. 340-342, 1999.

[MaPo00] F. Masseglia, P. Poncelet and M. Teisseire, “Incremental Mining of Sequential Patterns in Large Database”, Actes des 16ièmes Journées Bases de Données Avancées (BDA'00), Blois, France, Octobre 2000.

[MaTo96] H. Mannila and H. Toivonen. « On an Algorithm for Finding all Interesting Sequences ». In Proceedings of the 13th European Meeting on Cybernetics and Systems Research, Vienna, Austria, April 1996.

[MiSh01] T. Miyahara, T. Shoudai, T. Uchida, K. Takahashi and H. Ueda, "Discovery of Frequent Tree Structured Patterns in Semistructured Web Documents ", Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'01), pp. 47-52, Hong Kong, China, April 2001.

[PaZa99] S. Parthasarathy and M. J. Zaki, "Incremental and Interactive Sequence Mining", Proceedings of the Conference on Information and Knowledge Management (CIKM'99), pp. 251-258, Kansas City, USA, November 1999.

[WaLi97] K. Wang and H. Liu, "Schema Discovery for Semi-structured Data ", Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD'97), pp. 271-274., Newport Beach, USA, August 1997.

[WaLi98] K. Wang and H. Liu, "Discovering Typical Structures of Documents: A Road Map Approach", Proceeding of the Conference on Research and Development in Information Retrieval (SIGIR'98), pp. 146-154, Melbourne, Australia, August 1998.

[WaLi99] K. Wang and H. Liu, "Discovering Structural Association of Semistructured Data", In IEEE Transactions on Knowledge and Data Engineering , pp. 353-371, January 1999.

[WaSh96] J.TL. Wang, B.A. Shapiro, D. Shasha, K. Zhang and C.Y Chang, "Automated Discovery Structures" In Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD'96), pp. 70-75, 1996

[Zaki98] M. Zaki, "Scalable Data Mining for rules", PHD Dissertation, University of Rochester-NewYork, 1998.

[Zaki02] M. Zaki, "Efficiently Mining Frequent Trees in a Forest ", Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD'02), Edmonton, Canada, July 2002.

[ZhXu02] Q. Zheng, K. Xu, S. Ma and W. Lu, "The Algorithms of Updating Sequential Patterns", Proceedings of the International Conference on Data Mining (ICDM'02), Washington DC, USA, April 2002.