AUSMS: An Environment for Frequent Sub-Substructures Extraction in a Semi-Structured Object Collection

Pierre-Alain Laur, Maguelonne Teisseire, Pascal Poncelet

AUSMS: An environment for frequent sub-structures extraction in a semi-structured object collection

P.A Laur [1] – M. Teisseire [1] – P. Poncelet [2]

[1] LIRMM, 161 rue Ada, 34392 Montpellier cedex 5, France
{laur,teisseire}@lirmm.fr
[2] EMA/LGI2P, Ecole des Mines d'Alès
Site EERIE, Parc Scientifique Georges Besse, 30035 Nîmes cedex 1, France
Pascal.Poncelet@ema.fr

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 is rapidly increasing. Structure of these objects is irregular and it is judicious to assume that a query on documents structure is almost as important as a query on data. Moreover, manipulated data is not static since it is constantly being updated. The problem of maintaining such sub-structures then becomes as much of a priority as 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), which enables us to retrieve data, identify frequent sub-structures and keep up-to-date extracted knowledge after sources evolutions.

1. Introduction

The search for knowledge in structured data has been extensively addressed in the few past years. Most of the proposed approaches concern flat or highly structured structures. With the growing popularity of the World Wide Web, the number of semi-structured documents available is rapidly increasing. However in spite of this structural irregularity, structural similarities among semi-structured objects can exist and it is frequently noted that semi-structured objects which describe the same type of information have similar structures. The analysis of such implicit structures in semi-structured data can then provide significant information : to optimize requests evaluations, to obtain general information on the contents, to facilitate data integration resulting from various information sources, to improve storage, to facilitate index or views and to contribute to semi-structured documents classification. Applications fields are very numerous and gather, for example: bio-informatics,

Web Content Mining and Web Usage Mining. Recently, new approaches were proposed to discover such sub-structures [4, 8, 11, 14, 15]. Unfortunately, handled data are not static because new updates are constantly carried out. The problem of keeping such sub-structures up to date becomes very significant. As updates are carried out, the previously found sub-structures can become invalid. In this article we are interested in extraction of such sub-structures with a detailed attention for their evolution. We propose a system, called AUSMS (Automatic Update Schema Mining System), which allows collecting data, finding frequent sub-structures, and maintaining extracted knowledge during sources evolutions.
The article is organized in the following way. In section 2, we present the problems of searching frequent sub-structures and data maintenance. Section 3 presents the functional architecture of the system by detailing the various stages. A related work is proposed in section 4. Lastly, in section 5, we conclude.

2. Problem Statement

In this section, we give the formal definitions related to the problem of searching frequent sub-structures in semi-structured objects.
The goal of our proposal is to discover structural similarities among a set of semi-structured objects. Since in our context a cyclic graph can be transformed into an acyclic graph [14], we consider in the following a tree as an acyclic connected graph and a forest as a collection of trees where each tree is a connected component of the forest (rooted tree). Furthermore we consider that handled trees are ordered tree (a rooted tree in which the children of each node are ordered). The order is given according to the type of application and it follows either the lexicographical order (set-of), or the imposed order (list-of). To express the differences between orders in the following, we will respectively use the notations "{}" to represent a "set of" and "< >" to represent a "list of". Due to lack of space, we do not define formally the inclusion of a structure in a tree. Nevertheless, we illustrate this notion by the following example

Example: for example, let us consider figure 1 and the structure [*address*: {*city*, *street*, *zipcode*}, *category*, *name*]. This structure is a sub-structure of the tree [*root*:{*address*: {*city*, *street*, *zipcode*}, *category*,

name, *nearby*: {*category*, *name*, *price*}}]. However the structure [*address*: {*city*, *street*, *zipcode*}, *category*, *name*, *price*] is not a sub-structure of the tree since the element *price* is not on the same level in the graph.

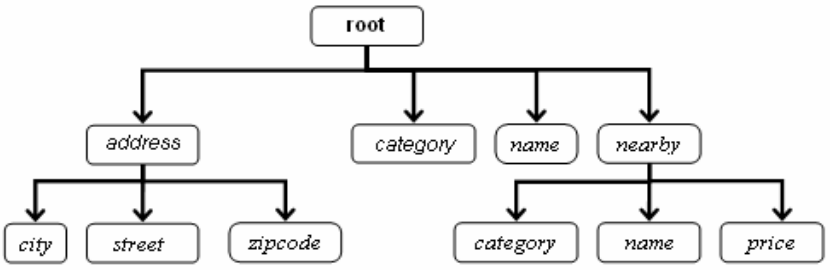

Fig. 1. Examples of sub-structures

Let us now consider DB a tree database also named structures, i.e. a forest where each tree T is composed of an identifier and a structure included in the forest. Let *supp (p)* be the support value for a structure corresponding to the number of occurrences of this structure in the database DB, i.e. the support of a structure *p* is defined as the percentage of all trees in the database which contain *p*. A tree of the database contains *p* iff *p* is a sub-structure of this tree. In order to decide whether a structure is frequent or not, a value of minimal support is specified by the user (*minSupp*) so a structure is frequent if the condition *supp (p)≥minSupp* holds. Being given a tree database DB, the problem of searching regularities in semi-structured data thus consists in finding all the maximum structures which are in DB and whose support is higher than minSupp.

Let us now consider the data sources evolutions. Let db the database increment where new information is added or removed. Let U=DB $\cup$ db, be the updated database holding all structures from DB and db. Let L^{DB} be the frequent sub-structures set in DB. The problem of keeping of up to date discovered knowledge is to seek the frequent sub-structures in U, noted L^{U}, by respecting the same support value without restarting mining algorithm from scratch.

3. The A.U.S.M.S. System

The aim of A.U.S.M.S. (Automatic Update Schema Mining System) is to propose an environment for knowledge discovery in semi-structured data from information recovery until the update of extracted knowledge. These general principles are illustrated in figure 2. The process can be broken into three main phases. First, starting from rough semi-structured data files, a pre-processing eliminates the irrelevant data and performs the transformation into the database. In a second phase, a knowledge extraction algorithm is used to find the frequent sub-structures which are stored into a database. Then, evolution of data sources is taken into account in order to update previously extracted knowledge. Finally, a visualization tool is provided to the end user.

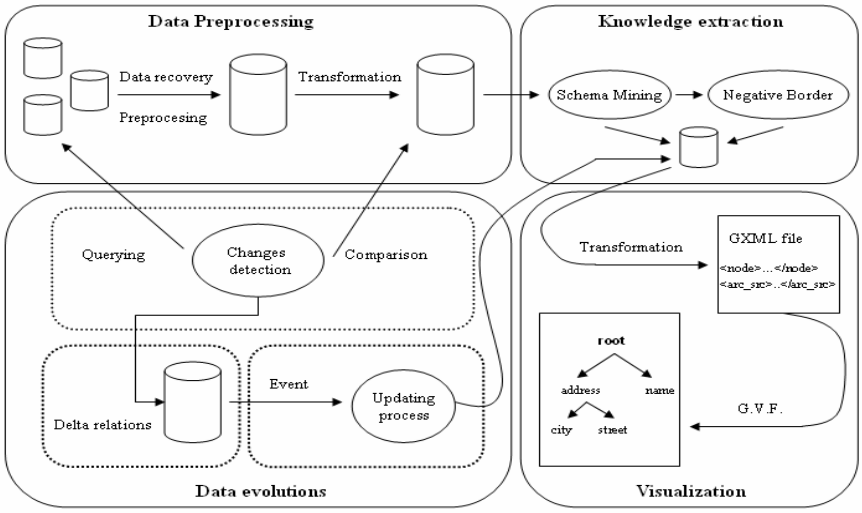

Fig. 2. General Architecture

In the following sections, we detail the extraction and the incremental process as well as the visualization tool.

3.1 Knowledge Extraction

We showed in [7] that there was a bijection between the problems of mining sub-structures such as we defined it and that of searching sequential patterns defined in [2]. To find the frequent structures we use a

level-wize algorithm largely inspired by [2]. Interested reader may refer to [8]. In order to improve the candidate generation procedure as well as the management of candidate elements, we use a bitmap representation inspired by [3]. This structure offers the advantage of considerably reducing the storage space and the ability to easily generate candidates. Moreover it is particularly adapted in the long structures search. During the candidates search phase we also generate the negative border [10]. This is made up of all the structures which are not frequent but whose sub-structures are frequent. This negative border will be used in the following phase to take into account the data sources evolutions.

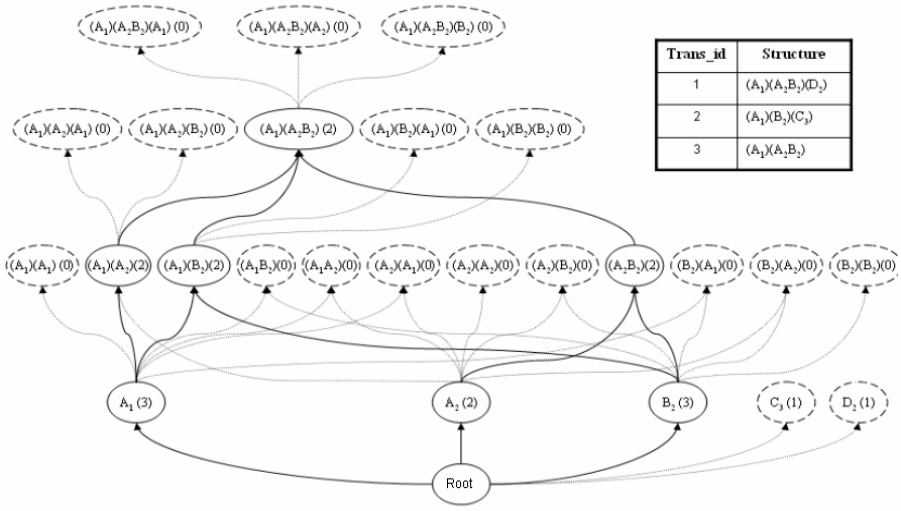

Fig. 3. Example of a negative border

Example: Let us consider figure 3 representing a lattice associated with a sample database. For a minimal support of 50 %, on level 1, only the A_1, A_2 and B_2 elements are frequent and can be used to create more complex structures. We thus store in the negative border, the C_3 and D_2 elements. On level 2, only $(A_1)\ (A_2)$, $(A_1)\ (B_2)$, $(A_2\ B_2)$ are frequent, we preserve in the negative border those of the preceding level elements which were frequent.

3.3 Taking into account data source evolutions

The negative border obtained in the previous stage enables us to take into account updates and to maintain extracted knowledge. Indeed, to

avoid applying the previous algorithm again at the time of each update, we store in the negative border the minimal information required to quickly compute the frequent sub-structures. The general principles are as follows:

Update algorithm

Input: S Set of data sources, BN the negative border, BN^{Limit}, L^{DB} the set of frequent structures, minSupp the minimal support specified by the user
Output: the updated sources S, BN updated and L^{DB} updated
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})

From a date specified by the user (delay), the data sources are compared (as we maintain previous sources s_{old} stands for the initial data sources, i.e. during the last analysis, and s_{new} represents the data being analyzed, i.e. s). This operation is carried out in the AUSMS system by an agent which acts either in a temporal way (fixed time difference since last update), or in a direct way (user activation). The agent is in charge of comparing the data sources and propagating the modifications. Thus, if the data source was modified, the updates are stored as a Δs set which manages the history of the modifications (UpdateDeltaRelation procedure). This procedure, inspired from the delta relations, used in active rules, makes it possible to reflect the side effects of the structure modifications [5].

From the information contained as Δs set, a comparison is carried out, by the procedure Validate (Δ_S, BN^{Limit}), with the elements contained in the negative border which are likely to change quickly, i.e. those which can become frequent or not, up to one element. This procedure also takes into account the addition or the suppression of new sources which of course generate a modification of the support value. If one of the conditions is then verified the modifications are brought directly into the negative border to update the set of the frequent structures (procedure Update (L^{DB})). The first stage consists in deferring the modifica-

tions in the negative border as soon as structures are added or removed. Indeed, such an operation causes the calculation of the support value to be modified for the whole base. For each structure, we thus examine the support value in the negative border and if this one is lower than the support, the branches of the tree resulting from this structure are pruned. Otherwise the other elements are re-examined and the negative border is updated according to their frequency. When the operations consist in adding or removing elements in existing structures, we analyze the negative border while starting with level 1 so as to verify how frequently the elements appear. If elements become frequent the various levels of the lattice are built recursively with those which were already frequent. If frequent elements become infrequent, the various branches of the lattices resulting from the sub-structure are pruned. At the end of this phase, the frequent elements are extracted and L^{DB} is updated as well as the negative border.

3.4 Visualization

Whereas previous modules are charged to provide and maintain frequent sub-structures, this module makes it possible to visualize these structures and offers a formalism to describe them.

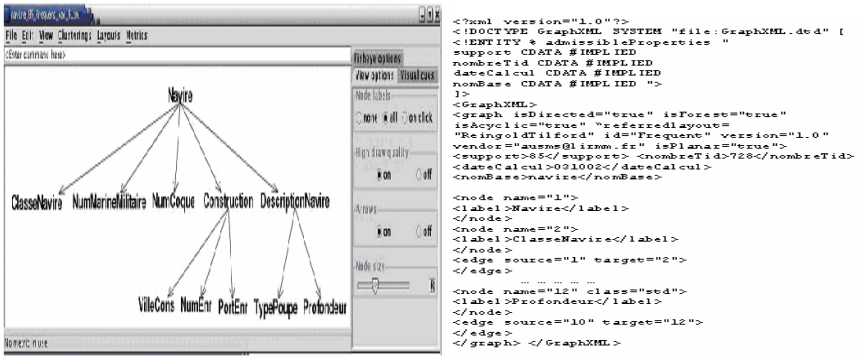

Fig. 4. Examples of extracted structures

For that, we use, initially, GraphXML [6] which is a graph description language in XML especially designed for drawing and display systems. In the second place, for visualizing extracted sub-structure as well as

their implication in the date sources, we use the Graph Visualization Framework which proposes a set of java classes to visualize and handle the structures described by the GraphXML format.
We find at the left a frequent structure resulting from the frequent sub-structures search with at least 85% of a history of ships database displayed via GVF. On the right-hand side we have the same description within the GraphXML format.

4. Related work

Our approach is very close to that proposed in [13, 14] for the search for structural association in semi-structured data. The authors propose a very effective approach and solutions based on a new representation of the search space. Moreover, by proposing optimizations based on pruning strategies, they considerably improve candidates' generation stage. In the same way, an approach suggested in [11] is rather similar to the previous approach and uses a particular tree called tag tree patterns. In [4], authors propose an algorithm called Find-Freq-Trees which also uses an approach based on a search by level as in the algorithm A-priori [1] and extends the proposal in order to discover sub-structures in long sequences. Finally in [15], author proposes two algorithms TreeMinerH and TreeMinerV for the search for frequent trees in a forest. TreeMinerH takes again the principle of the course in width of A-priori by improving candidates' generation and counting using the classes of equivalences, a structure of prefixed tree and "scope list". In TreeMinerV, a tree is described by a vertical structure. In these two algorithms, candidates' generation and counting are carried out by set operations on the "scope list"; the prefixed structure makes it possible to reduce the number of transactions to be traversed in the database. In our context, we are interested in the search for all the structures included in the base whereas they are interested only in search of tree-expression which are defined like trees going from the root to a final leaf of the tree. With this definition, they cannot find regularities of the form *[identity : {address : <street, zipcode>}]* which would be frequent but would be included in a longer transaction which is, itself, not frequent.
According to the maintenance of the extracted frequent sub-structures, there does not exist, to our knowledge, works in this field. We showed that the search for sub-structures could approach that of sequential pat-

terns. In the continuation of our work, we will thus examine the work carried out around this field. In [12], authors propose an algorithm called ISM (Incremental Sequence Mining) which allows an update of the frequent sequences. The suggested approach builds a lattice of sequence which contains all the frequent and negative border elements [10]. When new information arrives, they are added to this lattice. The problem of this approach is obviously the increasing size of the negative border which in our case is minimized, because based on bit vectors. In [9], the ISE (Incremental Sequence Extraction) algorithm was proposed for the search for frequent patterns, it generates candidates in the entire database by attaching the sequences of the incremental database to those of the original base. This approach avoids keeping the sequences contained in the negative border and the recalculation of these sequences when the initial database has been updated. However, by not preserving the negative border, it is necessary to more often traverse the base to search for candidates. In [16] the algorithm proposed uses at the same time the concepts of negative border of the original data base and the concepts of suffixes and prefixes in the contrary of ISE. To control the size of this negative border, they introduce a minimum support for these elements thus reducing its size. Moreover this algorithm realizes an extension by prefix and suffix (using the negative border). The problem of this algorithm lies in the choice of the value of the minimum support for the negative border.

5. Conclusion

In this article, we proposed a functional architecture, AUSMS, of a system of extraction and maintenance of knowledge in semi-structured objects databases. The originality of the approach lies in the implementation of effective algorithms to extract the frequent sub-structures in the base from semi-structured objects but also in the taking into account of the handled data. The tests which we carried out on bases resulting from the Web show that the adopted approach is very useful to help the end-user in the analysis of the various handled elements. It offers solutions for the search of general information in the data sources, to contribute to the interrogation of semi-structured databases and to help building views and indexes.

6. References

[1] R. Agrawal, T. Imielinski, and A. swami, “Mining Association Rules between Sets of Items in Large Databases”, Proceedings of SIGMOD’93, pp. 2076, May 1993.

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

[3] J. Ares, J. Gehrke, T. Yiu and J. Flannick, “Sequential Pattern Using Bitmap Representation”, Proceedings of PKDD’02, Edmonton, Canada, July 2002.

[4] T. Asai, K. Abe, and al., “Efficient substructure discovery from Large Semi-structured Data”, Proceedings of the (ICDM’02) Conference, Washington DC, USA, April 2002.

[5] S. Chawathe, S. Abiteboul and J. Widom, “Representing and Querying Changes History in Semistructured Data”, Proceedings of ICDE’98, Orlando, USA, February 1998.

[6] I. Herman and M.S. Marshall, “GraphXML An XML based graph interchange format”, Centre for Mathematics and Computer Sciences (CWI), Technical Report Amsterdam, 2000.

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

[8] P.A. Laur et P. Poncelet. “AUSMS : un environement pour l’extraction de sous-structures fréquentes dans une collection d’objets semi-structurées (in french)”. Actes des Journées d’Extraction et Gestion des Connaissances (EGC’03), Lyon, France, 2003.

[9] F. Masseglia, P. Poncelet and M. Teisseire, “Incremental Mining of Sequential Patterns in Large Database”, Actes des Journées BDA'00, Blois, France, Octobre 2000.

[10] 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.

[11] T. Miyahara, T. Shoudai, T. Uchida, K. Takahashi and H. Ueda, “Discovery of Frequent Tree Structured Patterns in Semistructured Web Documents”, Proceedings of PAKDD’01, pp. 47-52, Hong Kong, China, April 2001.

[12] S. Parthasarathy and M. J. Zaki, “Incremental and Interactive Sequence Mining”, Proceedings of the CIKM’99 Conference, pp. 251-258, Kansas City, USA, November 1999.

[13] K. Wang and H. Liu, ”Schema Discovery for Semi-structured Data”, Proceedings of the KDD’97 Conference, pp. 271-274., Newport Beach, USA, August 1997.

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

[15] M. Zaki, “Efficiently Mining Frequent Trees in a Forest”, Proceedings of SIGKDD’02, Edmonton, Canada, July 2002.

[16] 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), April 2002.