Poster Session: An Indexing Structure for Automatic Schema Matching

Fabien Duchateau
LIRMM - UMR 5506
Université Montpellier 2
34392 Montpellier Cedex 5 - France
duchatea@lirmm.fr

Zohra Bellahsène
LIRMM - UMR 5506
Université Montpellier 2
34392 Montpellier Cedex 5 - France
bella@lirmm.fr

Mark Roantree
Interoperable Systems Group
Dublin City University, Ireland
mark.roantree@computing.dcu.ie

Mathieu Roche
LIRMM - UMR 5506
Université Montpellier 2
34392 Montpellier Cedex 5 - France
mroche@lirmm.fr

Abstract

Querying semantically related data sources depends on the ability to map between their schemas. Unfortunately, in most cases matching between schema is still largely performed manually or semi-automatically. Consequently, the issue of finding semantic mappings became the principal bottleneck in the deployment of the mediation systems in large scale where the number of ontologies and or schemata to be put in correspondence is very large. Currently the mapping tools employ techniques for mapping two schemas at a time with human intervention for ensuring a good quality of mappings. In the large-scale scenario such techniques are not suitable. Indeed, in such a scenario one requires an automated performance oriented solution. Moreover, the automated method should also provide acceptable quality of mappings. In this paper, we present an automatic schema matching approach dealing with two aspects: performance and quality of mappings. However, we will focus on the performance aspect. For this, our method uses a B-tree index structure. Furthermore, our approach has been implemented and the experiments with real sets of schema show that it is scalable and provides very good performance.

1. Introduction

Interoperability among applications in distributed environments, including today's World-Wide Web and the emerging Semantic Web, depends critically on the ability to map between them. Unfortunately, automated data integration, and more precisely matching between schema, is still largely done by hand, in a labor-intensive and error-prone process. As a consequence, semantic integration issues have become a key bottleneck in the deployment of a wide variety of information management applications. The high cost of this bottleneck has motivated numerous research activities on methods for describing, manipulating and (semi-automatically) generating schema mappings.

The schema matching problem consists in identifying one or more terms in a schema that match terms in a target schema. The current semi-automatic matchers **[7, 1, 9, 12, 5, 10, 14]** calculate various similarities between elements and they keep the couples with a similarity above a certain threshold. The main drawback of such matching tools deals with the performances: although the matching quality provided at the end of the process is acceptable, the elapsed time to match implies a static and limited number of schema. Yet in many domain areas, a dynamic environment involving large sets of schema is required. Nowadays' matching tools must combine both acceptable quality and good performances.

In a previous work [**4**], we presented the Approxivect method to calculate a semantic similarity between two elements from different XML schema. Contrary to similar works, this approach is automatic, it does not use any dictionary or ontology and is both language and domain independent. It consists in using both terminological algorithms and structural rules. Indeed the terminological approaches enable to discover similarity of elements represented by close character strings. On the other hand, the structural rules are used to define the notion of context of a node. This context includes some of its neighbors, each of them is associated a weight representing the importance it has when evaluating the contextual node. Vectors composed of neighbor nodes are compared with the cosine measure to detect any similarity. Finally the different measures are aggregated for all couples of nodes. We have shown in [**4**] that Approxivect provides a good quality of mappings regarding the existing tool [**1**].

Unfortunately our Approxivect approach, like most of the matchers, lacks to provide good performances in terms of time. The motivation of our work is to improve this aspect of our method by using an indexing structure to accelerate the schema matching process, while still ensuring an acceptable quality by using Approxivect. The B-tree structure has been chosen to reach this goal, as it aims at searching and finding efficiently an index among a large quantity of data. Indeed, we assume that two similar labels share at least a common token, so instead of parsing the whole schema, we just search for the tokens indexed in the B-tree. A prototype, BtreeMatch, has been designed. Furthermore, we performed some experiments based on large sets of schema and the results show that our approach is scalable. Our main contributions are:

- Indexing structure for matching, which provides good performance.
- Use of tokenisation, terminological measures and context matching of label tokens; thus clustering similar label.
- Experiments with large number of real XML schemas (OAGIS, XCBL) showing good performance applicable for a large scale scenario.

The rest of the paper is structured as follows: first we briefly explain some general concepts in Section 2; in Section 3, an outline of our method is described; in Section 4, we present the results of our experiments; an overview of related work is given in Section 5 and in Section 6, we conclude and outline some future work.

2 Preliminaries

We consider schemas as rooted, labeled trees. This provides us the benefit for computing contextual semantics of nodes in the schema hierarchy.

Definition 1: A **schema** is a labeled unordered tree $S = (V_S, E_S, r_S, label)$ where:

- V_S is a set of nodes;
- r_S is the root node;
- $E_S \subseteq V_S \times V_S$ is a set of edges;
- *label* $V_S \rightarrow \Lambda$ where Λ is a countable set of labels.

Definition 2: Let V be the domain of schema nodes, the **similarity measure**, is a concept whereby two or more terms are assigned a metric value based on the likeness of their meaning / semantic content [6]. In the case of two schema nodes, this is a value $V \times V \rightarrow \Re$, noted *sim(n, n')*, defined for two nodes *n* and *n'*. Note that the semantic similarity depends on the method used to calculate it. In general, a zero value means a total dissimilarity whereas the 1 value stands for totally similar concepts.

Definition 3: A **mapping** is a non-defined relationship *rel* between nodes of two schemas V_S and V'_S:

$$V_S \times V'_S \rightarrow rel$$

The relationship between nodes can include synonyms, equality, hyperonyms, hyponyms, etc. The similarity measure between the two nodes may be compared with a certain threshold, defined by an expert, to determine if two elements should be mapped.

Example of schema matching: Consider the two following schemas used in [3]. They represent organization in universities from different country and have been widely used in the literature.

With those schemas, the ideal set of mappings given by an expert is {*(CS Dept Australia, CS Dept U.S.), (courses, undergrad courses), (courses, grad courses), (staff, people), (academic staff, faculty), (technical staff, staff), (lecturer, assistant professor), (senior lecturer, associate professor), (professor, professor)*}.

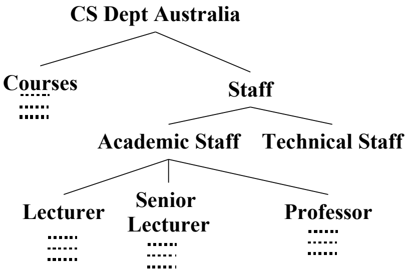

Figure 1. Schema 1: organization of an Australian university.

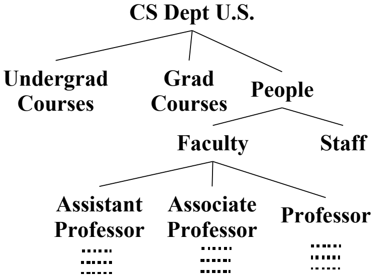

Figure 2. Schema 2: organization of a US university.

3 Overview of our approach

In this section, at first, we introduce the basis of our approach: Approxivect component which focuses on the semantic aspect and the B-tree indexing structure component, which is dealing with the performance aspect. Then, we describe our method combining these two components.

3.1 Approxivect approach

Approxivect approach aims at discovering similarities between XML elements. One of its particular feature is that it enables to discover several relationships (synonyms, hyponyms, ...) but does not rely on any dictionary and is not domain specific. See [4] for more details.

Approxivect (**Approxi**mation of **vect**ors) approach is based on two steps: first we replace labels when they have close character strings. This step uses the Levenhstein distance and 3-grams algorithms [**6, 8**]. In a second time, we calculate the cosine measure [**13**] between two vectors to determine if their context is close or not. By context we mean some important neighbor nodes like ancestors and descendants. A formula has been provided in [**4**] to calculate the importance of those neighbors for a given node.

Let us present our Approxivect algorithm. The two schemas are traversed in preorder traversal and all nodes labels are compared two by two with the Levenhstein distance and the 3-grams. Both measures are processed and according to the adopted strategy: in our previous experiments the maximum and average strategies reveals to be a good compromise. The obtained value is denoted **SM** for **String Measure**. If **SM** is above a certain threshold, which is defined by an expert, then some replacements may occur. We decided to replace the label with the larger number of characters by the label with the smaller number of characters. Indeed we consider that the smaller-sized label is more general than the bigger-sized one. This assumption can be checked easily since some labels may be written singular or plural. So we finally obtain after this first step the initial schema that has possibly been modified with character string replacements.

In the second part of our algorithm, we traverse again the schemas - in which some string replacements may have occurred due to Approxivect step 1. And the context vector of a current element is extracted in each schema. The neighbor elements composing this vector may be ancestors, descendants, siblings or further nodes of the current element. However, each of them is assigned with a weight, illustrating the importance of this neighbour with regards to the current node. The two context vectors are compared using the cosine measure, in which we include the weight of the node. Indeed, when counting the number of occurrences of a label, we multiply this number by its weight. This process enables to calculate **CM**, the cosine measure between two context vectors, and thus the semantic similarity between the two nodes related to these contexts too.

The matching quality has already been compared to another schema matcher, COMA++ [**1**]. Experiments have been done on several couples of schemas to show that Approxivect offers an acceptable quality [**4**]. For example, consider schemas 1 and 2. An expert would find 9 relevant similarities: COMA++ finds only 5 of them while Approxivect is able to discover all the relevant similarities. But Approxivect suffers from the same drawback than the other matchers : slow performances. The next section presents an indexing structure to accelerate the matching.

3.2 An indexing structure: the B-tree

In our approach, we use the B-tree as the main structure to locate matches and create mappings between XML tree structures. The advantage of searching for mappings using the B-tree approach is that B-tree have indexes that significantly accelerate this process. Indeed, if you consider the schemas 1 and 2, they have respectively 8 and 9 elements, implying 72 matching possibilities with an algorithm that tries all combinations. And those schemas are small examples, but in some domains, schemas may contain up to 6 000 elements. By indexing in a B-tree, we are able to reduce this number of matching possibilities, thus involving better performances.

As described in [**2**], B-trees have many features. A B-tree is composed of nodes, each of them having a list of indexes. A B-tree of order *M* means that each node can have up to *M* children nodes and contains a maximum of *M-1* indexes. Another feature is that the B-tree is balanced, meaning all the leaves are at the same level - thus enabling fast insertion and fast retrieval since a search algorithm in a B-tree of *n* nodes visits only $1+log_{\mathrm{M}}n$ nodes to retrieve an index. This balancing involves some extra processing when adding new indexes into the B-tree, but its impact is limited when the B-tree order is high.

The B-tree is a structure widely used in databases due to its efficient capabilities of storing information. As schema matchers need to store and retrieve quickly a lot of data when matching, an indexing structure such as B-tree could improve the performances. The B-tree has been prefered to the B+tree (which is commonly used in databases systems) since we do not need the costly delete operation. Thus with this condition, the B-tree seems more efficient than the B+tree because it stores less indexes and it is able to find an index quicker. As most databases use a B+tree structure, we did not consider

3.3 Our BtreeMatch approach

By using both Approxivect and the B-tree structure, the objective is to combine their main advantage: an acceptable matching quality and good performances. Contrary to most of the other matching tools, BtreeMatch does not use a matrix to compute the similarity of each couple of elements. Instead, a B-tree, whose indexes represent tokens, is built and enriched as we parse new schemas, and the discovered mappings are also stored in this structure. The tokens reference all labels which contains it. For example, after parsing schemas 1 and 2, the **courses** token would hold three labels: **courses** from schema 1, **grad courses** and **undergrad courses** from schema 2. Note that the labels **grad courses** and **undergrad courses** are also stored respectively under the **grad** and the **undergrad** tokens.

For each input XML schema, the same algorithm is applied: the schema is parsed element by element by preorder traversal. This enables to compute the context vector of each element. The label is split into tokens. We then fetch each of those tokens in the B-tree, resulting in two possibilities:

- no token is found, so we just add it in the B-tree with a reference to the label.
- or the token already exists in the B-tree, in which case we try to find semantic similarities between the current label and the ones referenced by the existing token. We assume that in most cases, similar labels have a common token (and if not, they may be discovered with the context similarity).

Let us illustrate this case. When **courses** is parsed in schema 1, the label is first tokenized, resulting in the following set of tokens: **courses**. We search the B-tree for this single token, but it does not exist. Thus we create a token structure whose index is **courses** and which stores the current label **courses** and it is added into the B-tree. Later on, we parse **grad courses** in schema 2. After tokenization process, we obtain this set of tokens: **grad**, **courses**. We then search the B-tree for the first token of the set, but **grad** does not exist. A token structure with this **grad** token as index is inserted in the B-tree, and it stores the **grad courses** label. Then the second token, **courses**, is searched in the B-tree. As it already exists, we browse all the labels it contains (here only **courses** label is found) to calculate the Approxivect String Measure denoted SM between them and **grad courses**. Approxivect can replace one of the label by the other if they are considered similar (depending on the Approxivect parameters). Whatever happens, **grad courses** is added in the **courses** structure. The next parsed element is **undergrad courses**, which is composed of two tokens, **undergrad** and **courses**. The first one results in an unsuccessful search, implying an **undergrad** token structure to be created. The second token is already in the B-tree, and it contains the two labels previously added: **courses** and **grad courses**. The String Measures are computed between **undergrad courses** and the two labels, involving replacements if SM reaches a certain threshold. **undergrad courses** is added in the label list of the **courses** token structure. So the index enables to quickly find the common tokens between occurences, and to limit the String Measure computation with only a few labels.

At this step, some string replacements might have occurred. Then the parser performs recursively the same action for the descendants nodes, thus enabling to add the children nodes to the context. Once all descendants have been processed, similarities might be discovered by comparing the label with tokens' references using the cosine and the terminological measures. A parameter can be set to extend the search to the whole B-tree if no mappings has been discovered.

Let us carry on our example. After processing **undergrad courses**, we should go on with its children elements. As it is a leaf, we then search the B-tree again for all the tokens which compose the label **undergrad courses**. Under the **undergrad** token, we find only one label, itself so nothing happens. Under the **courses** token, only one of the three existing labels namely **courses**, is interesting (one is itself and the other, **grad courses**, is in the same schema). The String Measure is thus applied between **courses** and **undergrad courses**. The Cosine Measure is also performed between their respective context, and the aggregation of these two measures results in the semantic measure between those labels. If this semantic measure reaches a certain threshold, then a mapping may be discovered.

4 Experiments

To evaluate the benefit provided by the index structure, we made comparison between Approxivect alone and the whole method (i.e., Approxivect + B-tree indexing structure). In order to properly evaluate our work, we developed a prototype system of BtreeMatch in Java, using the SAX parser. As some experiments about the matching quality have already been done in previous work [**4**] between Approxivect and COMA++, we do not deal with the quality here. We only focus on performances, namely the time spent to match a large number of schemas. For those experiments we used a 2 Ghz Pentium 4 laptop running Windows XP, with 2 Gb RAM. Java Virtual Machine 1.5 is the current version. The context of a node is limited to its parent and its children nodes. Although this constraint could be removed, it has been shown in the Approxivect experiments that the context should not include too many further nodes which could have a bad impact on the quality.

Table 1. Characterization of the schema sets

	XCBL Schemas	**OASIS Schemas**
Average number of nodes per schema	21	2 065
Largest / smallest schema size	426 / 3	6 134 / 26
Maximum depth	7	21

The table 1 shows the different features of the sets of schemas we used in our experiments. Two large scale scenarios are presented: the first one involves a thousand of average-sized schemas about business-to-business e-commerce, taken from the XCBL[1] standards. In the second case, we deal with OASIS[2] schemas which are also business domain related. We use only several hundreds of those schemas because they are quite large, with an average of 2000 nodes per schema.

4.1 First scenario: XCBL

Here we compare the performances of Approxivect and BtreeMatch on a large set of average schemas. The results are illustrated by the graph depicted in Figure 3. We can see that Approxivect is efficient when the number of schemas is not very large (less than 1600). BtreeMatch method provides good performances with a larger number of schemas, since two thousands schemas are matched in 200 seconds.

4.2 Second scenario: OASIS

In this scenario, we are interested by matching large schemas, with an average of 2000 nodes. The graph depicted in Figure 4 shows that Approxivect is not suited for large schemas. On the contrary, BtreeMatch is able to match an important number of large schemas in less than one minute. The graph also shows that BtreeMatch is quite linear. Indeed, for 900 schemas, BtreeMatch needs around 130 seconds to perform the matching.

5 Related Work

In the literature, many schema matching approaches [**7, 1, 9, 12, 5, 10, 14**] have been studied at length.

[1] www.xcbl.org

[2] www.oagi.org

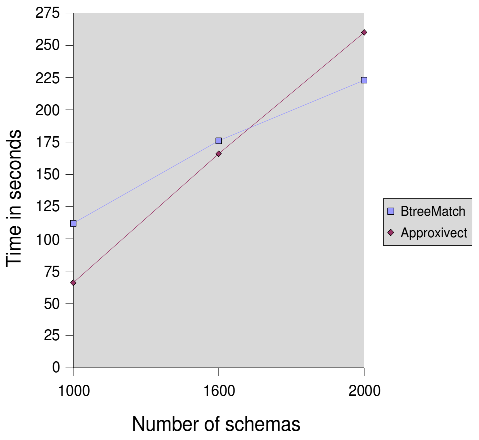

Figure 3. Matching time with XCBL schemas.

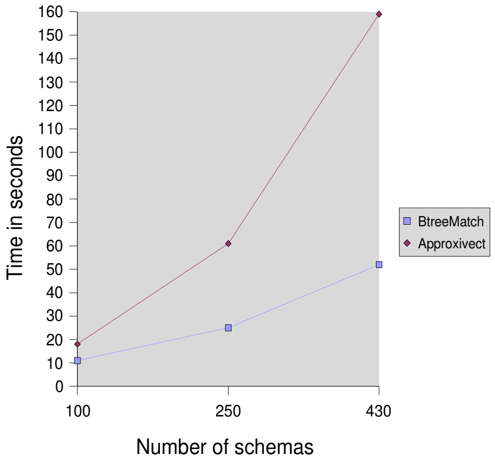

Figure 4. Matching time with OASIS schemas.

Most of them have been designed to demonstrate their benefit in different scenarios. However, the currently mapping tools employ techniques for mapping two schemas with human intervention. To the best of our knowledge, this is the only one previous work dealing with large schemas [**11**], using COMA++ tool [**1**]. In this work, first, the user divides the schema into fragments and then each fragment from source schema is mapped to target schema fragments, to find inter-fragment matching. Next, these fragment mappings are merged to compute the schema level mappings. Thus, the tool is not able to process directly large schemas. Another issue of this approach [**11**] is which criteria is the best for fragmenting the large schemas.

To conclude, the existing tools are semi-automatic and are designed for small schemas. Moreover they, did not focus on the performance aspect, while our method has the following properties:

- automatic
- designed for processing large schemas
- scalable

6 Concluding Remarks

In this paper, we presented our BtreeMatch approach to improve the time elapsed on schema matching. Our method is based on a B-tree structure which includes an index mechanism and module for discovering semantic similarity between elements of schemas. To evaluate the benefit provided by the index structure, we made comparison between Approxivect alone and BtreeMatch (i.e., Approxivect + B-tree indexing structure). The experiments have shown that BtreeMatch is faster than Approxivect in most cases, especially when the number of information that needs to be stored becomes important. An indexing structure could be needed when the schemas are either very large or numerous. Note that the B-tree can directly store the mappings into memory, whereas Approxivect needs another means of storage.

The result of our experiments are very interesting and showing that our method is scalable and provide good performance and quality of mappings. We are planning to seek for schemas involving more heterogeneity, thus we need to enhance Approxivect by adding specific parsers for each format file. Currently, the corpus of schemas available on the web are normalized, i.e. there is no synonyms, tokens have the same delimiters. For that, one of our ongoing work is to establish a benchmark involving a large corpus of non-normalized schemas for evaluating schema matching tools.

References

[1] D. Aumueller, H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with coma++. In *SIGMOD 2005*, 2005.

[2] D. Comer. The ubiquitous btree. In *Computing Surveys*, 1979.

[3] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Ontology matching: A machine learning approach. In

Handbook on Ontologies, International Handbooks on Information Systems, 2004.

[4] F. Duchateau, Z. Bellahsène, and M. Roche. A context-based measure for discovering approximate semantic matching between schema elements. Technical Report RR-06053, LIRMM, 2006.

[5] J. Euzenat et al. State of the art on ontology matching. Technical Report KWEB/2004/D2.2.3/v1.2, Knowledge Web, 2004.

[6] D. Lin. An information-theoretic definition of similarity. In *Proc. 15th International Conf. on Machine Learning*, pages 296–304. Morgan Kaufmann, San Francisco, CA, 1998.

[7] J. Madhavan, P. Bernstein, and E. Rahm. Generic schema matching with cupid. In *VLDB01*, 2001.

[8] A. Maedche and S. Staab. Measuring similarity between ontologies. In *Proc. of the European Conference on Knowledge Acquisition and Management - EKAW*, pages 251–263, 2002.

[9] S. Melnik, H. G. Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proc. of the International Conference on Data Engineering (ICDE'02)*, 2002.

[10] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, 2001.

[11] E. Rahm, H. Do, D. Aumueller, and S. Massmann. Matching large xml schemas. In *ACM SIGMOD Record 33(4):26-31*, 2004.

[12] J. Tranier, R. Barar, Z. Bellahsène, and M. Teisseire. Where's charlie: Family-based heuristics for peer-to-peer schema integration. In *Proc of IDEAS*, pages 227–235, 2004.

[13] R. Wilkinson and P. Hingston. Using the cosine measure in a neural network for document retrieval. In *Proc of ACM SIGIR Conference*, pages 202–210, 1991.

[14] M. Yatskevich. Preliminary evaluation of schema matching systems. Technical Report DIT-03-028, Informatica e Telecomunicazioni, University of Trento, 2003.