Approximate Query Answering for a Heterogeneous XML Document Base *

Federica Mandreoli, Riccardo Martoglia, and Paolo Tiberio

Università di Modena e Reggio Emilia,
Dip. di Ingegneria dell'Informazione, Modena, Italy
{tiberio.paolo, mandreoli.federica, martoglia.riccardo}@unimo.it

Abstract. In this paper, we deal with the problem of effective search and query answering in heterogeneous web document bases containing documents in XML format of which the schemas are available. We propose a new solution for the structural approximation of the submitted queries which, in a preliminary schema matching process, is able to automatically identify the similarities between the involved schemas and to use them in the query processing phase, when a query written on a source schema is automatically rewritten in order to be compatible with the other useful XML documents. The proposed approach has been implemented in a web service and can deliver middleware rewriting services in any open-architecture XML repository system offering advanced search capabilities.

1 Introduction

In recent years, the constant integration and enhancements in computational resources and telecommunications, along with the considerable drop in digitizing costs, have fostered development of systems which are able to electronically store, access and diffuse via the Web a large number of digital documents and multimedia data. In such a sea of electronic information, the user can easily get lost in her/his struggle to find the information (s)he requires. Heterogeneous collections of various types of documents, such as actual text documents or metadata on textual and/or multimedia documents, are more and more widespread on the web. Think of the several available portals and digital libraries offering search capabilities, for instance those providing scientific data and articles, or those assisting the users in finding the best bargains for their shopping needs. Such repositories often collect data coming from different sources. The documents are *heterogeneous* for what concerns the structures adopted for their representations but are *related* for the contents they deal with. In this context, XML has quickly become the de facto standard for data exchange and for heterogenous data representation over the Internet. This is also due to the recent emergence of wrappers (e.g. [1, 2]) for the translation of web documents into XML format. Along with XML, languages for describing the structures and data types and for querying XML documents are becoming more and more popular. Among the several languages proposed in recent years, the syntax and semantics of XML Schema [3] and of XQuery [4] are W3C recommendations/working drafts, for the former and the latter purposes respectively. Thus, in a large number of heterogeneous web collections, data are most likely expressed in XML and are associated to XML Schemas, while structural queries submitted to XML web search engines are written in XQuery, a language expressive

* The present work is partially supported by the "Technologies and Services for Enhanced Content Delivery" FSI 2000 Project.

enough to allow users to perform structural inquiries, going beyond the "flat" bag of words approaches of common plain text search engines.

In order to exploit the data available in such document repositories, an entire ensemble of systems and services is needed to help users to easily find and access the information they are looking for. Sites offering access to large document bases are now widely available all over the web, but they are still far from perfect in delivering the information required by the user. In particular, one of the issues which is still an open problem is the effective and efficient search among large numbers of "related" XML documents. Indeed, if, from one side, the adoption of XQuery allows users to perform structural inquiries, on the other hand, such high flexibility could also mean more complexity: Hardly a user knows the exact structure of all the documents contained in the document base. Further, XML documents about the same subject and describing the same reality, for instance compact disks in a music store, but coming from different sources, could use largely different structures and element names, even though they could be useful in order to satisfy the user's information need. Given those premises, the need for solutions to the problem of performing queries on all the useful documents of the document base, also on the ones which do not exactly comply with the structural part of the query itself but which are similar enough, becomes apparent.

Recently, several works took into account the problem of answering approximate structural queries against XML documents. Much research has been done on the instance level, trying to reduce the approximate structural query evaluation problem to well-known unordered tree inclusion (e.g. [5, 6]) or tree edit distance [7] problems directly on the data trees. However, the process of unordered tree matching is difficult and extremely time consuming; for instance, the edit distance on unordered trees was found in [8] NP hard. On the other hand, a large number of approaches prefer to address the problem of structural heterogeneity by first trying to solve the differences between the schemas on which data are based (e.g. [9–11]). However, most of the work on XML schema matching has been motivated by the problem of schema integration and the fundamental aspect of query rewriting remains a particularly problematic and difficult to be solved aspect [12]. Conversely, most of the works on query rewriting do not actually benefit from the great promises of the schema matching methods, and, while presenting interesting theoretical studies [13], they generally do not propose practical ways of efficiently performing the rewriting operation itself.

In this paper, we propose an effective and efficient approach for *approximate query answering* in heterogeneous document bases in XML format. Instead of working directly on the data, we interpret the structural component of the query by exploiting a reworking of the documents' schemas. In particular, our approach relies on the information about the structures of the XML documents which we suppose to be described in XML Schema. A *schema matching* process extracts the *semantic* and *structural* similarities between the schema elements which are then exploited in the proper query processing phase where we perform the *rewriting* of the submitted queries, in order to make them compatible with the available documents' structures. The queries produced by the rewriting phase can thus be issued to a "standard" XML engine and enhance the effectiveness of the searches. Such an approach has been implemented in a web service, named XML S^3MART (XML Semantic Structural Schema Matcher for Automatic query RewriTing), we are integrating in the context of the ongoing Italian MIUR Project "Technologies and Services for Enhanced Content Delivery" (ECD Project), whose aim is the production of new and advanced technologies enabling the full exploitation of the information available in web document collections or digital libraries, offering enhanced contents and services such as advanced search engines with a cutting-edge search effectiveness.

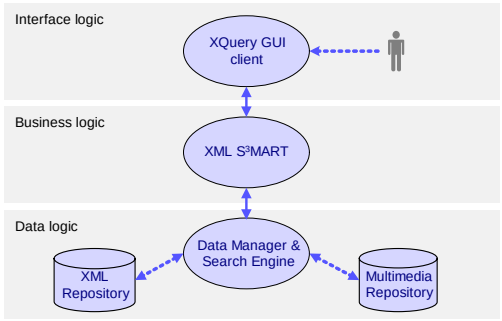

Fig. 1. The role of schema matching and query rewriting open-architecture web repository offering advanced XML search functions

The paper is organized as follows: In Sect. 2 we give a brief overview of the XML S^{3}MART service motivations and functionalities and introduce how such a module can work in an open-architecture web repository offering advanced XML search functions. Then, in Sections 3 and 4 we specifically analyze the matching and rewriting features. The results of some of the experiments we conducted are discussed in Sect. 5. Finally, Sect. 6 concludes the paper.

2 Overview of the Matching and Rewrite Service

From a technological point of view, the principle we followed in planning, designing and implementing the matching and rewriting functionalities was to offer a solution allowing easy extensions of the offered features and promoting information exchange between different systems. In fact, next generation web systems offering access to large XML document repositories should follow an *open* architecture standard and be partitioned in a series of modules which, together, deliver all the required functionalities to the users. Such modules should be autonomous but should cooperate in order to make the XML data available and accessible on the web; they should access data and be accessed by other modules, ultimately tying their functionalities together into services. For all these reasons, XML S^{3}MART has been implemented as a *web service* and makes use of SOAP which, together with the XML standard, give the architecture an high level of inter-operability.

The matching and rewriting services offered by XML S^{3}MART can be thought as the middleware (Fig. 1) of a system offering access to large XML repositories containing documents which are heterogenous, i.e. incompatible, from a structural point of view but related in their contents. Such middleware interacts with other services in order to provide advanced search engine functionalities to the user. At the interface level users can exploit a graphical user interface, such as [14], to query the available XML corpora by drawing their request on one of the XML Schemas (named *source schema*). XML S^{3}MART automatically rewrites the query expressed on the source schema into a set of XML queries, one for each of the XML schemas the other useful documents are associated to (*target schemas*). Then, the resulting XML queries can be submitted to a standard underlying data manager and search engine, such as [15], as they are consistent with the structures of the useful documents in the corpus. The results are then gathered, ranked and sent to the user interface component. Notice that the returned results can be actual XML textual documents but also multimedia data for which XML metadata are available in the document base.

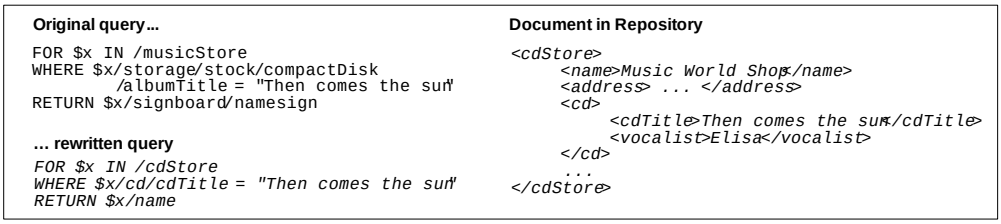

Fig. 2. A given query is rewritten in order to be compliant with useful documents in the repository

Let us now concentrate on the motivation and characteristics of our approximate query answering approach. The basic premise is that the structural parts of the documents, described by XML schemas, are used to search the documents as they are involved in the query formulation. Due to the intrinsic nature of the semi-structured data, all such documents can be useful to answer a query only if, though being different, the target schemas share meaningful similarities with the source one, both *structural* (similar structure of the underlying XML tree) and *semantical* (employed terms have similar meanings) ones. Consider for instance the query shown in the upper left part of Fig. 2, asking for the names of the music stores selling a particular album. The document shown in the right part of the figure would clearly be useful to answer such need, however, since its structure and element names are different, it would not be returned by a standard XML search engine. In order to retrieve all such useful documents available in the document base, thus fully exploiting the potentialities of the data, the query needs to be rewritten (lower left part of Fig. 2). Being such similarities independent from the queries which could be issued, they are identified by a *schema matching* operation which is preliminary to the proper query processing phase. Then, using all the information extracted by such analysis, the approximate query answering process is performed by first applying a *query rewriting* operation in a completely automatic, effective and efficient way.

As a final remark, notice that, some kind of approximation could also be required for the values expressed in the queries as they usually concern the contents of the stored documents (texts, images, video, audio, etc.), thus requiring to go beyond the exact match. We concentrate our attention only on the structural parts of the submitted queries and we do not deal with the problem of value approximation, which has been considered elsewhere (for instance in [16]).

3 Schema Matching

The schema matching operation takes as input the set of XML schemas characterizing the structural parts of the documents in the repository and, for each pair of schemas, identifies the "best" matches between the attributes and the elements of the two schemas. It is composed by three sub-processes, the first two of which, the structural expansion and the terminology disambiguation, are needed to maximize the effectiveness of the third phase, the real matching one.

Structural Schema Expansion. The W3C XML Schema [3] recommendation defines the structure and data types for XML documents. The purpose of a schema is to define a class of XML documents. In XML Schema, there is a basic difference between complex types, which allow elements as their content and may carry attributes, and simple types, which cannot have element content and attributes. There is also a major distinction between definitions which create new types (both simple and complex), and

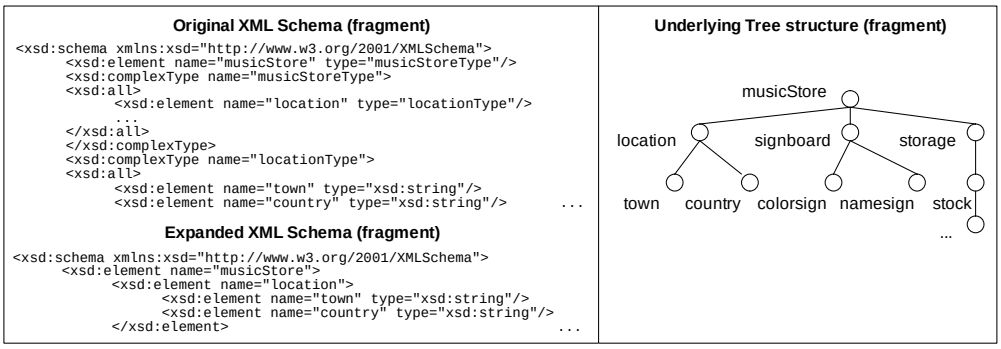

Fig. 3. Example of structural schema expansion (Schema A)

declarations which enable elements and attributes with specific names and types (both simple and complex) to appear in document instances. An XML document referencing an XML schema uses (some of) the elements introduced in the schema and the structural relationships between them to describe the structure of the document itself.

In the structural schema expansion phase, each XML schema is modified and expanded in order to make the structural relationships between the involved elements more explicit and thus to represent the class of XML documents it defines, i.e. the structural part of the XML documents referencing it. As a matter of fact, searches are performed on the XML documents stored in the repository and a query in XQuery usually contains paths expressing structural relationships between elements and attributes. For instance the path `/musicStore/storage/stock` selects all the `stock` elements that have a `storage` parent and a `musicStore` grandparent which is the root element. The resulting expanded schema file abstracts from several complexities of the XML schema syntax, such as complex type definitions, element references, global definitions, and so on, and ultimately better captures the tree structure underlying the concepts expressed in the schema.

Consider, for instance, Fig. 3 showing a fragment of an XML Schema describing the structural part of documents about music stores and their merchandiser, along with a fragment of the corresponding expanded schema file and a representation of the underlying tree structure expressing the structural relationship between the elements which can appear in the XML documents complying with the schema. As can be seen from the figure, the original XML Schema contains, along with the element definitions whose importance is definitely central (i.e. elements `musicStore`, `location`), also type definitions (i.e. complex types `musicStoreType`, `locationType`) and regular expression keywords (i.e. `all`), which may interfere or even distort the discovery of the real underlying tree structure, which is essential for an effective schema matching. In general, XML Schema constructions need to be resolved and rewritten in a more explicit way in order for the structure of the schema to be the most possibly similar to its underlying conceptual tree structure involving elements and attributes. Going back to the example of Fig. 3, the element `location`, for instance, is conceptually a child of `musicStore`: This relation is made explicit only in the expanded version of the schema, while in the original XML Schema `location` was the child of a `all` node, which was child of a complex type. Further, every complex type and keyword is discarded.

In the resulting expanded schema, every element or attribute node has a name. Middle elements have children and these can be deduced immediately from the new explicit structure. On the other hand, leaf elements (or attributes) can hold a textual

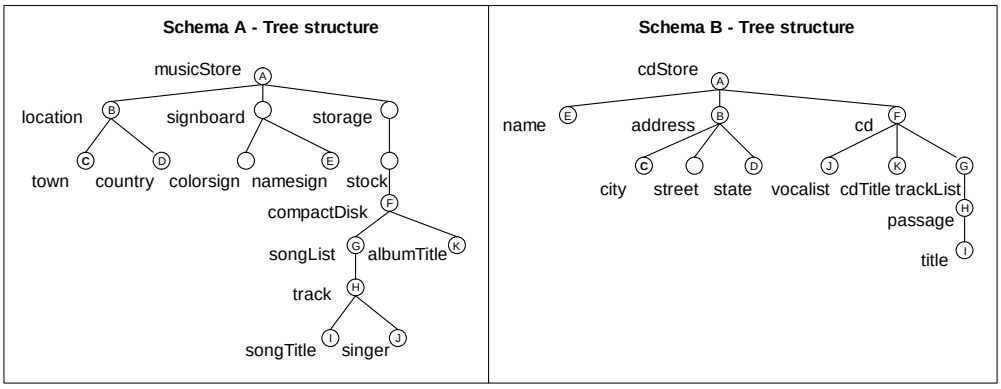

Fig. 4. Example of two related schemas and of the expected matches

value, whose primitive type is maintained and specified in the "type=..." parameter of the respective nodes of the output schema.

Terminology disambiguation. After having made explicit the structural relationships of a schema with the expansion process, a further step is required in order to refine and complete even more the information delivered by each schema, thus maximizing the successive matching computation's effectiveness. This time the focus is on the *semantics* of the terms used in the element and attribute definitions. In this step, each term is disambiguated, that is its meaning is made explicit as it will be used for the identification of the semantical similarities between the elements and attributes of the schemas, which actually rely on the distance between meanings. To this end, we exploit one of the most known lexical resources for the English language: WordNet [17]. The WordNet (WN) lexical database is conceptually organized in synonym sets or *synsets*, representing different meanings or *senses*. Each term in WN is usually associated to more than one synset, signifying it is polysemic, i.e. it has more than one meaning. At present, term disambiguation is implemented by a semi-automatic operation where the operator, by using an ad-hoc GUI, is required to "annotate" each term used in each XML schema with the best candidate among the WN terms and, then, to select one of its synsets.

Matching Computation. The matching computation phase performs the actual matching operation between the expanded annotated schemas made available by the previous steps. For each pair of schemas, we identify the "best" matchings between the attributes and the elements of the two schemas by considering both the structure of the corresponding trees and the semantics of the involved terms. Indeed, in our opinion the meanings of the terms used in the XML schemas cannot be ignored as they represent the semantics of the actual content of the XML documents. On the other hand, the structural part of XML documents cannot be considered as a plain set of terms as the position of each node in the tree provides the context of the corresponding term. For instance, let us consider the two expanded schemas represented by the trees shown in Fig. 4. Though being different in the structure and in the adopted terminology, they both describe the contents of the albums sold by music stores, for which the information about their location is also represented. Thus, among the results of a query expressed by using Schema A we would also expect documents consistent with Schema B. In particular, by looking at the two schemas of Fig. 4, a careful reader would probably identify the matches which are represented by the same letter. At this step, the terms used in the two schemas have already been disambiguated by choosing the best WN synset. As WordNet is a general purpose lexical ontology, it does not provide meanings

for specific context. Thus, the best choice is to associate terms as `albumTitle` and `songTitle` for Schema A and `cdTitle` and `title` for Schema B with the same WN term, i.e. `title`, for which the best synset can thus be chosen. In these cases, which are quite common, it is only the position of the corresponding nodes which can help us to better contextualize the selected meaning. For instance, it should be clear that the node `albumTitle` matches with the node `cdTitle`, as both refer to album title, and that `songTitle` matches with the node `title`, as both refer to song title.

The steps we devised for the matching computation are partially derived from the ones proposed in [11] and are the following:

1. the involved schemas are first converted into directed labelled graphs following the RDF specifications [18], where each entity represents an element or attribute of the schema identified by the full path (e.g. `/musicStore/location`) and each literal represents a particular name (e.g. `location`) or a primitive type (e.g. `xsd:string`) which more than one element or attribute of the schema can share. As to the labels on the arcs, we mainly employ two kinds of them: `child`, which captures the involved schema structure, and `name`. Such label set can be optionally extended for further flexibility in the matching process. From the RDF graphs of each pair of schemas a *pairwise connectivity graph (PCG)*, involving node pairs, is constructed [11] in which a labelled edge connects two pairs of nodes, one for each RDF graph, if such labelled edge connects the involved nodes in the RDF graphs.
2. Then an initial similarity score is computed for each node pair contained in the PCG. This is one of the most important steps in the matching process. In [11] the scores are obtained using a simple string matcher that compares common prefixes and suffixes of literals. Instead, in order to maximize the matching effectiveness, we chose to adopt an in-depth semantic approach. Exploiting the semantics of the terms in the XML schemas provided in the disambiguation phase, we follow a *linguistic approach* in the computation of the similarities between pairs of literals (names), which quantifies the distance between the involved meanings by comparing the WN hypernyms hierarchies of the involved synsets. We recall that hypernym relations are also known as IS-A relations (for instance "feline" is a hypernym for "cat", since you can say "a cat is a feline"). In this case, the scores for each pair of synsets (s_1, s_2) are obtained by computing the depths of the synset in the WN hypernyms hierarchy and the length of the path connecting them as follows:

$$\frac{2 * \text{depth of the least common ancestor}}{\text{depth of } s_1 + \text{depth of } s_2} .$$

3. The initial similarities, reflecting the semantics of the single node pairs, are refined by an iterative fixpoint calculation as in the similarity flooding algorithm [11], which brings the structural information of the schemas in the computation. In fact, this method is one of the most versatile and also provides realistic metrics for match accuracy [19]. The intuition behind this computation is that two nodes belonging to two distinct schemes are the more similar the more their adjacent nodes are similar. In other words, the similarity of two elements propagates to their respective adjacent nodes. The fixpoint computation is iterated until the similarities converge or a maximum number of iterations is reached.
4. Finally, we apply a stable marriage filter which produces the "best" matching between the elements and attributes of the two schemas. The stable marriage filter guarantees that, for each pair of nodes (x, y), no other pair (x', y') exists such that x is more similar to y' than to y and y' is more similar to x than to x'.

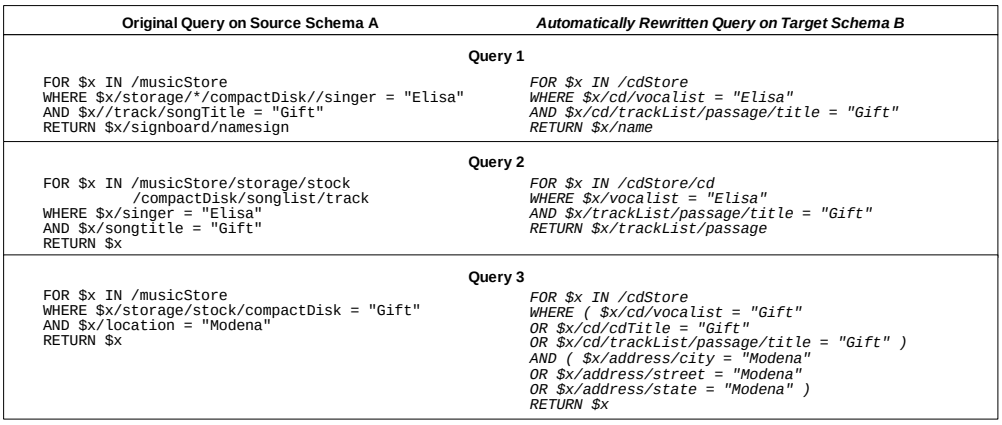

Fig. 5. Examples of query rewriting between Schema A and Schema B

4 Automatic Query Rewriting

By exploiting the best matches provided by the matching computation, we straightforwardly rewrite a given query, written w.r.t. a source schema, on the target schemas. Each rewrite is assigned a score, in order to allow the ranking of the results retrieved by the query. Query rewriting is simplified by the fact that the previous phases were devised for this purpose: The expanded structure of the schemas summarizes the actual structure of the XML data where elements and attributes are identified by their full paths and have a key role in an XQuery FLWOR expression paths. At present, we support conjunctive queries with standard variable use, predicates and wildcards (e.g. the query given in Fig. 2). Due to the lack of space, we will briefly explain the approach and show some meaningful examples. After having substituted each path in the `WHERE` and `RETURN` clauses with the corresponding full paths and then discarded the variable introduced in the `FOR` clause, we rewrite the query for each of the target schemas in the following way:

1. all the full paths in the query are rewritten by using the best matches between the nodes in the given source schema and target schema (e.g. the path `/musicStore /storage/stock/compactDisk` of Schema A is automatically rewritten in the corresponding best match, `/cdStore/cd` of Schema B);
2. a variable is reconstructed and inserted in the `FOR` clause in order to link all the rewritten paths (its value will be the longest common prefix of the involved paths);
3. *a score* is assigned to the rewritten query. It is the average of the scores assigned to each path rewriting which is based on the similarity between the involved nodes, as specified in the match.

Fig. 5 shows some examples of query rewriting. The submitted queries are written by using Schema A of Fig. 4 and the resulting rewriting on Schema B is shown on the right of the figure. Query 1 involves the rewriting of a query containing paths with wildcards: In order to successfully elaborate them, the best matches are accessed not exactly but by means of regular expressions string matching. For instance, the only path of the tree structure of Schema A satisfying the path `/musicStore/storage/*/compactDisk //singer` is `/musicStore/storage/stock/compactDisk/songList/track/singer` and the corresponding match in Schema B (label J in Fig. 4) will be the one used in the rewrite. When more than one path of the source schema satisfies a wildcard path, all the corresponding paths are rewritten and put in an `OR` clause. Query 2 demonstrates

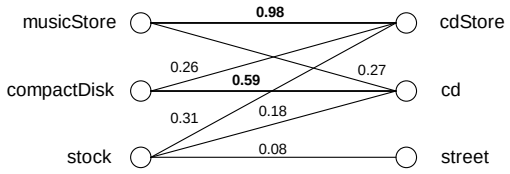

Fig. 6. A small selection of the matching results between the nodes of Schema A (on the left) and B (on the right) before filtering; similarity scores are shown on the edges.

the rewriting behavior in the variable management. The value of the `$x` variable in the submitted query is the path of the element `track` in Schema A and the corresponding element in Schema B is `passage` (label H in Fig. 4). However directly translating the variable value in the rewritten query would lead to a wrong rewrite: While the elements `singer` and `songTitle` referenced in the query are descendants of `track`, the corresponding best matches in Schema B, that is `songTitle` and `vocalist`, are not both descendants of `passage`. Notice that, in these cases, the query is correctly rewritten as we first substitute each path with the corresponding full path and then we reconstruct the variable, which in this case holds the value of path `cdStore/cd`. In example 3 an additional rewrite feature is highlighted concerning the management of predicates involving values and, in particular, textual values. At present, whenever the best match for an element containing a value is a middle element, the predicates expressed on such element are rewritten as `OR` clauses on the elements which are descendants of the matching target element and which contain a compatible value. For instance, the element `compactDisk` and its match `cd` on Schema B are not leaf elements, therefore the condition is rewritten on the descendant leaves `vocalist`, `cdTitle` and `title`.

5 Experimental Evaluation

In this section we present a selection of the most interesting results we obtained through the experimental evaluation performed on the prototype of XML S^{3}MART. Since in our method the proper rewriting phase and its effectiveness is completely dependent on the schema matching phase and becomes quite straightforward, we will mainly focus on the quality of the matches produced by the matching process. We are currently evaluating the effectiveness of our techniques in a wide range of contexts by performing tests on a large number of different XML schemas. Such schemas include the ones we devised ad hoc in order to precisely evaluate the behavior of the different features of our approach, an example of which is the music store example introduced in Fig. 4, and schemas officially adopted in worldwide DLs in order to describe bibliography metadata or audio-visual content. Due to space limitations, in this section we will discuss the results obtained for the music store example and for a real case concerning the schemas employed for storing the two most popular digital libraries of scientific references in XML format: The DBLP Computer Science Bibliography archive and the ACM SIGMOD Record.

For the music store schema matching, we devised the two schemas so to have both different terms describing the same concept (such as `musicStore` and `cdStore`, `location` and `address`) and also different conceptual organizations (notably `singer`, associated to each of the tracks of a cd in Schema A, vs. `vocalist`, pertaining to a whole cd in Schema B). Firstly, we performed a careful annotation, in which we associated each of the different terms to the most similar term and sense available in WordNet. After annotation, XML S^{3}MART iterative matching algorithm automatically identified

the best matches among the node pairs which coincides with the ones shown in Fig. 4. For instance, matches A, E, F and G between nodes with identical annotation and a similar surrounding context are clearly identified. A very similar context of surrounding nodes, together with similar but not identical annotations, are also the key to identify matches B, C, D, H and J. The matches I and K require particular attention: Schema A `songTitle` and `albumTitle` are correctly matched respectively with Schema B `title` and `cdTitle`. In these cases, all four annotations are the same (*title*) but the different contexts of surrounding nodes allow XML S^{3}MART to identify the right correspondences. Notice that before applying the stable marriage filtering each node in Schema A is matched to more than one node in Schema B; simply choosing the best matching node in Schema B for each of the nodes in Schema A would not represent a good choice. Consider for instance the small excerpt of the results before filtration shown in Fig. 6: The best match for `stock` (Schema A) is `cdStore` but such node has a better match with `musicStore`. The same applies between `stock` - `cd`, and `cd` - `compactDisk`. Therefore, the matches for `musicStore` and `compactDisk` are correctly selected (similarities in bold), while `stock`, a node which has no correspondent in Schema B, is ultimately matched with `street`. However, the score for such match is very low (< 0.1) and will be finally filtered out by a threshold filter.

As to the tests on a real case, Fig. 7 shows the two involved schemas, describing the proceedings of conferences along with the articles belonging to conference proceedings. Along with the complexities already discussed in the ad-hoc test, such as different terms describing the same concept (`proceedings` and `issue`, `inproceedings` and `article`), the proposed pair of schemas presents additional challenges, such as an higher number of nodes, structures describing the same reality with different levels of detail (as for `author`) and different distribution of the nodes (more linear for DBLP, with a higher depth for SIGMOD), making the evaluation of the matching phase particularly critical and interesting. In such real cases the annotation process is no longer trivial and many terms could not have a WN correspondence: For instance DBLP's `ee`, the link to the electronic edition of an article, is annotated as *link*, `inprocedings`, a term not available in WN, as *article*. In general, we tried to be as objective and as faithful to the schemas' terms as possible, avoiding, for instance, to artificially hint at the right matches by selecting identical annotations for different corresponding terms: For instance, terms like `procedings` (DBLP) and `issue` (SIGMOD) were annotated with the respective WN terms, `dblp` was annotated as *bibligraphy* while `sigmodRecord` as *record*.

After annotation, XML S^{3}MART matcher automatically produced the matches shown in Fig. 7. Each match is identified by the same letter inside the nodes and is associated with a similarity score (on the right). The effectiveness is very high: Practically all the matches, from the fundamental ones like B and G, involving articles and proceedings, to the most subtle, such as L involving the link to electronic editions, are correctly identified without any manual intervention. Notice that the nodes having no match (weak matches were pruned out by filtering them with a similarity threshold of 0.1) actually represent concepts not covered in the other schema, such as `authorPosition` or `location` for SIGMOD. The semantics delivered by the terminology disambiguation have a great role in deciding the matches, from matches D, E, F, G, L, and J, involving terms with similar contexts and identical annotations, to A and B, where the similarity of the annotations and contexts is very high. On the other hand, also the fixed point computation relying on the structure of the involved schemas is quite important. Indeed, nodes like `key` and `title` are present twice in DBLP but are nonetheless correctly matched thanks to the influence of the surrounding similar nodes: In particular, the `key` of an `inproceedings` is associated to the `articleCode` of an `article`, while the `key` of a `proceedings` has no match in Schema B. Matches

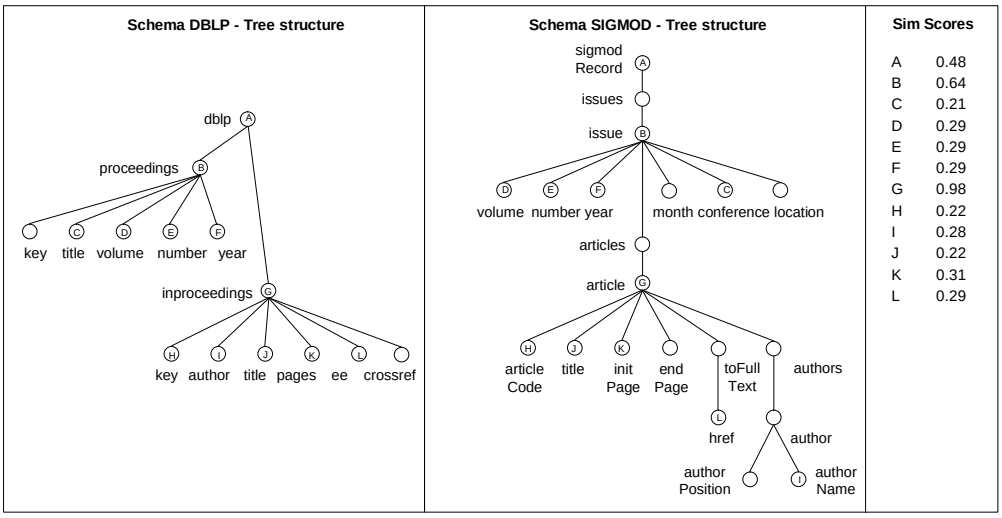

Fig. 7. Results of schema matching between Schema DBLP and Schema SIGMOD. Each match, represented by a letter, is associated to a similarity score (shown on the right).

C and I are particularly critical and can be obtained by annotating the terms with particular care: The node `conference` in Schema B actually represents the *title* of a given conference, and as such has been annotated and correctly matched. The same applies for the node `author` in Schema A, representing the *name* of an author. In this regard, we think that a good future improvement, simplifying the process of making annotations of complex terms like these, would be to allow and exploit composite annotations such has "the title of a conference", "the name of an author", or, for nodes like Schema B `articles`, "a group of authors". Further, an additional improvement, this time to the matching algorithm, might be to enable the identification of 1:n or, even more generally, of m:n correspondences between nodes: For instance match K is not completely correct since the node `pages` would have to be matched with two nodes of Schema B, `initPage` and `endPage`, and not just with `initPage`.

We performed many other tests on XML S^3MART effectiveness, generally confirming the correct identification of at least 90% of the available matches. Among them, we conducted "mixed" tests between little correlated schemas, for instance between Schema B and DBLP. In this case, the matches' scores were very low as we expected. For instance, the nodes labelled `title` in Schema B (title of a song) and DBLP (title of articles and proceedings) were matched with a very low score, more than three times smaller than the corresponding DBLP-SIGMOD match. This is because, though having the same annotation, the nodes had a completely different surrounding context. Finally notice that, in order to obtain such matching results, it has been necessary to find a good trade-off between the influence of the similarities between given pairs of nodes and that of the surrounding nodes, i.e. between the annotations and context of nodes; in particular, we tested several graph coefficient propagation formulas [11] and we found that the one delivering the most effective results is the inverse total average.

6 Conclusions

In this paper, we have considered the problem of query answering for a document base storing documents in XML format and we have proposed a method for structural query approximation which is able to automatically identify semantic and structural

similarities between the involved schemas and to exploit them in order to automatically rewrite a query written on a source schema towards other available schemas. The proposed approach has been implemented in a web service prototype, XML S^3MART, which is currently under testing and evaluation and represents a versatile solution since it accepts as input any query in the XQuery standard and, by means of rewriting, is able to enhance the effectiveness of the standard available XML search engines. The preliminary experimental results on query rewriting effectiveness are promising; as we expected, the rewriting efficiency is also particularly encouraging, thanks to the "lightness" of our method which relies on schema information alone and does not require explicit navigation of the XML data. In the near future, we plan to enhance both the schema matching, such as automatic structural word sense disambiguation, and the query rewriting by studying automatic deduction of the schema underlying the submitted queries and ad-hoc rewriting rules.

References

1. Baumgartner, R., Flesca, S., Gottlob, G.: Visual Web information extraction with Lixto. In: Proc. of the Twenty-seventh Int. Conference on Very Large Data Bases. (2001)
2. Crescenzi, V., Mecca, G., Merialdo, P.: RoadRunner: automatic data extraction from data-intensive web sites. In: Proc. of the 2002 ACM SIGMOD Int. Conference on Management of Data (SIGMOD-02). (2002)
3. Thompson, H.S., Beech, D., Maloney, M., Mendelsohn, N.: XMLSchema. W3C Recommendation (2001)
4. Boag, S., Chamberlin, D., Fernández, M.F., Florescu, D., Robie, J., Siméon, J.: XQuery 1.0: An XML Query Language. W3C Working Draft (2003)
5. Ciaccia, P., Penzo, W.: Relevance ranking tuning for similarity queries on xml data. In: Proc. of the VLDB EEXTT Workshop. (2002)
6. Schlieder, T., Naumann, F.: Approximate tree embedding for querying XML data. In: Proc. of ACM SIGIR Workshop On XML and Information Retrieval. (2000)
7. Guha, S., Jagadish, H.V., Koudas, N., Srivastava, D., Yu, T.: Approximate XML joins. In: Proc. of ACM SIGMOD. (2002) 287–298
8. Zhang, K., Statman, R., Shasha, D.: On the editing distance between unordered labeled trees. Inf. Process. Lett. **42** (1992) 133–139
9. Do, H., Rahm, E.: COMA – A system for flexible combination of schema matching approaches. In: Proc. of the 28th VLDB. (2002) 610–621
10. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic Schema Matching with Cupid. In: Proc. of the 27th VLDB. (2001) 49–58
11. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity Flooding: A Versatile Graph Matching Algorithm and ist Application to Schema Matching. In: Proc. of the 18th ICDE. (2002)
12. Rishe, N., Yuan, J., Athauda, R., Chen, S., Lu, X., Ma, X., Vaschillo, A., Shaposhnikov, A., Vasilevsky, D.: Semantic Access: Semantic Interface for Querying Databases. In: Proc. of the 26th VLDB. (2000) 591–594
13. Papakonstantinou, Y., Vassalos, V.: Query rewriting for semistructured data. In: Proc. of the ACM SIGMOD. (1999) 455–466
14. Braga, D., Campi, A.: A Graphical Environment to Query XML Data with XQuery. In: Proc. of the 4th Intl. Conference on Web Information Systems Engineering. (2003)
15. Castelli, D., Pagano, P.: A System for Building Expandable Digital Libraries. In: Proc. of the Third ACM/IEEE-CS Joint Conference on Digital Libraries. (2003)
16. Theobald, A., Weikum, G.: The index-based XXL search engine for querying XML data with relevance ranking. Lecture Notes in Computer Science **2287** (2002) 477
17. Miller, G.: WordNet: A Lexical Database for English. In: CACM 38. (1995)
18. Lassila, O., Swick, R.: Resource Description Framework (RDF) model and syntax specification. W3C Working Draft WD-rdf-syntax-19981008 (1998)
19. Do, H., Melnik, S., Rahm, E.: Comparison of schema matching evaluations. In: Proc. of the 2nd Int. Workshop on Web Databases. (2002)