Adding Flexibility to Structure Similarity Queries on XML Data

Paolo Ciaccia and Wilma Penzo

DEIS - CSITE-CNR
University of Bologna, Italy
{pciaccia,wpenzo}@deis.unibo.it

Abstract. The presence of structure inside XML documents poses the hard challenge of providing flexible query matching methods for effective retrieval of results. In this paper we present an approach that faces this issue in a twofold fashion: 1) it exploits new approximations on data structure; 2) it provides a relevance ranking method that takes into account the degree of *correctness* and *completeness* of results with respect to a given query, as well as the degree of *cohesion* of data retrieved.

1 Introduction and Motivation

The increasing availability of large digital libraries is raising XML to be the new standard for data representation. Because of the heterogeneity of document collections, querying blindly XML data requires a great deal of flexibility not to fall into the trap of "empty results", often caused by too strict constraints. In fact, the presence of structure inside XML documents is double-faced: on one hand, it helps to define the context where information has to be retrieved (for instance, "Retrieve the director of the movie entitled *Casablanca*"), on the other, it may be an obstacle to the retrieval of relevant information that slightly differs in data organization.

For instance, consider a user looking for stores in New York city selling CD's authored by Elton John, and containing songs with "love" in the title. Among the two stores listed in `Doc1` of Fig. 1, only the former fully satisfies query requirements. This is because it is the only store in `Doc1` that presents a `city` attribute, thus making condition on "New York" checkable. But, it is evident that also the second store is relevant to the query, and should be returned.

Thus, relaxation on query requirements is currently the scope of many proposals [4,7,10,19,20,21]. These works deal with similarity queries that approximate results both on semantics and on structural conditions. However, most of the above approaches do not recognize the second CD store of `Doc1` as relevant. Actually, except for ApproXQL [17], they do not cope with the problem of providing *structurally incomplete results*, i.e. results that only partially satisfy query requirements on data organization. Further, although supposing that condition on New York is checkable for the second CD store in `Doc1`, none of the above

T. Andreasen et al. (Eds.): FQAS 2002, LNAI 2522, pp. 124–139, 2002.

```
<cdstore>Artist Shop
  <address street = "105 Broad Street",
        city = "New York" >
  <cd>
    <title>One night only</title>
    <artist>Elton John</artist>
    <tracklist>
      <song>
        <title>
          Can you feel the love tonight
        </title>
      </song>
      ...
    </tracklist>
  </cd>
  ...
</cdstore>
<cdstore address ="Manhattan, New York" >
  Music Store
  <cd>
    <title> Disney solos for violin </title>
    <tracklist>
      <track>
        <author> Elton John </author>
        <title>
          Love of the common people
        </title>
      </track>
      ...
    </tracklist>
  </cd>
  ...
</cdstore>
```

Doc1

```
<musicstore name="CD Universe">
  <location>
    <town>New York</town>
  </location>
  <warehouse>
    <stock number ="157">
      <cd>
        <title>Love songs</title>
        <singer>Elton John</singer>
        <tracklist>
          <song>
            <title>
              Can you feel the love tonight
            </title>
          </song>
          ...
        </tracklist>
      </cd>
      <cd>
        <title>
          Songs from the west coast
        </title>
        <singer>Elton John</singer>
        <tracklist>
          <song>
            <title>I want love </title>
          </song>
          ...
        </tracklist>
      </cd>
    </stock>
  </warehouse>
</musicstore>
```

Doc2

Fig. 1. Sample XML documents

method would recognize it as relevant, because of a slight difference on data organization. In fact, Elton John appears as a track author rather than as the CD author, as specified by the query. This emphasizes the need of more flexibility in the evaluation of structure conditions, most of all in absence of knowledge of data organization.

As to relevance ranking, relaxations on both semantics and structure are expected to affect the score of results. However, in many proposals [4,7,10,20] the use of wildcards flatten the score of the retrieved data, which is considered relevant no matter how much "sparse" it is. Only in ApproXQL the number of elements matching a wildcard contributes to the ranking of results. But none of the above approaches takes into account neither the query *rate* satisfied by an answer, i.e. the *completeness* of the answer, as well as its *structural correctness*, nor the *cohesion* of results. In fact, also *sparseness* of data retrieved plays an important role for relevance evaluation. For instance, consider the CD store in `Doc2`. Although it satisfies the query conditions, note that the required CD's belong to the stock "157" of the store's warehouse. This changes the context where information is retrieved and, in this case, this may possibly mean that the

CD's are not currently available for sale. Then, this store contains two relevant CD's, thus probably it is expected to score higher than others.

In this paper we provide a method to find the *approximate embeddings* of a user query in XML document collections. Our proposal captures the relaxations described above, and provides a ranking measure that takes into account the degree of correctness and completeness of results with respect to a given query, as well as the degree of cohesion of data retrieved. Our proposal relies on tree representation both for XML documents [3], and for queries. The outline is as follows: In Section 2 we discuss related work. In Section 3 we start from the unordered tree inclusion problem [12] to relate query and data trees through *embeddings*; this is extended in two directions: 1) to capture also partial matching on query structure, and 2) to assign a score to the retrieved embeddings. In Section 4 we define the $SATES$ (*Scored Approximate Tree Embedding Set*) function, to retrieve and score embeddings, and we give a flavour of the flexibility and the effectiveness of the method through examples. However, we report its formal definition in appendix. Section 5 is devoted to relevance ranking computation. Properties of correctness and completeness of embeddings are presented. We also compare our method with other related approaches. In Section 6 we briefly discuss implementation and, finally, in Section 7 we conclude and discuss our planned future work.

2 Related Work

Several current proposals [4,6,7,8,10,11,16,17,20,21] deal with similarity queries on XML data. In XXL [20] similarity basically depends on document content, through the evaluation of vague predicates, and information on structure is only exploited to combine semantic similarity scores. Partial match on query structure is not supported. XIRQL [10] introduces the concept of *index object* to specify document units. Terms are weighted locally to these units, thus defining the relevance of each term in the scope given by the node. Thus, structural conditions act as filters to determine the context where information has to be searched in. Partial match on query structure yet is not discussed. The latter feature is neglected also in [7]. The ranking of scores depends on the context where data is retrieved, and semantic similarity is not exploited.

Then, other approaches [4,17] adopt tree-based retrieval models to captures XML data relationships in a natural way. In fact, classical models (e.g. vector space, probabilistic) lack the ability to handle structural information, although some extensions have been proposed for them [18,21]. In [4] relevance ranking is based on tree pattern relaxations. Nodes and edges in a query tree are assigned pairs of weights that denote scores for exact and relaxed matching, respectively. Sum is used to combine scores of each single node/edge matching. Partial match on query structure is not supported. Only ApproXQL [17] considers partial match on query structure.

Then, in all above-cited proposals cohesion of data retrieved is not considered for relevance. In [7] the length of paths connecting two matching nodes is

neglected; In [20] wildcards are intentionally used, thus discarding the (negative) contribution of intermediate nodes between matching data; In [10] this issue is not discussed. In [17] similarity scores depend on the costs of basic transformations applied on the query tree. These costs do not depend on data. Thus, semantic relationships, like synonymy, are not exploited.
Further, to our knowledge, for a given document, all proposed methods return only the *best* (in some cases, approximate) matching. The set-oriented approach used in the $SATES$ function also captures the relevance given by multiple occurrences of the query pattern in the retrieved data.

Other related approaches deal with matching elements of data schemas, usually represented as graphs [14,15]. In [14] trees are used as a starting point to perform schema matching (for instance, to compare XML documents). Structural similarity is measured as the fraction of leaves appearing in the matching. The notion of completeness is then limited to leaf coverage. Cohesion is not explicitly discussed, although it influences the final similarity of trees, through the use of descreasing factors. In [15] quality of results is measured in terms of the effort the user needs to spend to obtain a perfect match from the automatically generated one. This is accomplished through the addition/deletion of node pairs in the mapping. This suggests a measure of incompleteness of results (when nodes are to be added), but it does not provide any information on cohesion of data retrieved.

3 Relaxing the Tree Embedding Problem

In a tree view of documents and queries, several proposals [4,7,10,17,20] rely on tree/graph pattern matching techniques to decide if a document d is a possible answer to a query q.

In most of these works, *all* query nodes must have a corresponding matching node in the document tree, and *each* parent-child relationship should be guaranteed at least by an ancestor-descendant one in the data tree. This is also known as the *tree embedding problem* [12]. Nevertheless, in many cases these conditions are too restrictive, yet being not flexible enough to effectively deal with the structural heterogeneity of XML documents. In fact, in absence of knowledge of data organization, documents often do not correspond *completely* to query requirements, yet being mostly relevant although in absence of some conditions. This is the case, for instance of the second CD store of `Doc1`, that does not present any `city` element/attribute to be checked for the query condition on New York. Further, frequently, documents do not *exactly* fit the structural constraints provided in a user query. An example is `Doc1`, where Elton John appears as track author, instead of CD author as required in the sample query presented above.

Our work basically loosens the strictness of this approach, that often leads to empty results because of minor differences in data organization. The key aspects of our proposal are:

1. the relaxation on the concept of *total* embedding of a query tree t_q in a document tree t_d, in that we admit *partial* structural match of the query tree, as well as *approximations on structural relationships*; further, the match is relaxed to consider semantic similarity between nodes;
2. *ranking* of results with respect to the *cohesion* of data retrieved, to the *relaxation* of semantic and structural constraints, and to the *coverage rate* of the query;
3. a set-oriented approach to results, that depending on different relaxations on requirements, identifies possible alternatives that the user may be interested in. This also strengthens the relevance of data presenting multiple occurrences of query patterns.

Point **1)** leads to the introduction of our interpretation of *approximate tree embedding*. First, we report the basic notation we will use throughout paper and appendix. As to tree representation of XML documents, we follow the XML Information Set standard [3] (with *root* and *label* as `document element` and `local name` properties, respectively, and *parent* and *children* as the homonymous properties, denoting a parent-child relationship between argument nodes, and the set of first level children of a given node, respectively). We also define some additional functions for nodes n, n_1, and n_2 in a tree: *leaf(n)* iff $children(n) = \emptyset$, and $ancestor(n_1,n_2)$ iff $(parent(n_1,n_2) \vee \exists\ n \in N$ s.t. $(parent(n_1,n) \wedge ancestor(n,n_2))$. We denote with $\mathcal{T}$ the set of such trees. Given a tree $t \in \mathcal{T}$, $nodes(t)$ returns the set of all nodes of t. Then, given a node n in a tree $t \in \mathcal{T}$, we define *supp(n)* the subtree of t rooted at n, and we call it the *support* of n in t. Let $\mathcal{T}_D \subset \mathcal{T}$ be the set of all data trees, and $\mathcal{T}_Q \subset \mathcal{T}$ be the set of all query trees.

We intentionally neglect the presence of namespaces and links (IDREFs) inside XML documents and we assume data is well-formed, but we do not require validity with respect to a Document Type Definition (DTD).

Definition 1 (Approximate Tree Embedding (ATE)). Given a query tree $t_q \in \mathcal{T}_Q$ and a document tree $t_d \in \mathcal{T}_D$, an *approximate tree embedding* of t_q in t_d is a *partial* injective function $\tilde{e}[t_q, t_d] : nodes(t_q) \nrightarrow nodes(t_d)$ such that $\forall q_i, q_j$ in the domain of $\tilde{e}$ ($dom(\tilde{e})$):

a). $sim(label(q_i),label(\tilde{e}(q_i))) > 0$, where *sim* is a similarity operator that returns a score normalized in [0,1] stating the semantic similarity between the two given labels

b). $parent(q_i,q_j) \Rightarrow (ancestor(\tilde{e}(q_i),\tilde{e}(q_j)) \vee sibling(\tilde{e}(q_i),\tilde{e}(q_j)))$

c). $sibling(q_i,q_j) \Rightarrow (sibling(\tilde{e}(q_i),\tilde{e}(q_j)) \vee ancestor(\tilde{e}(q_i),\tilde{e}(q_j)))$

Let $\mathcal{E}$ be the set of approximate tree embeddings.

Points *b)* and *c)* of Def. 1 capture both the presence of intermediate nodes between $\tilde{e}(q_i)$ and $\tilde{e}(q_j)$ (ancestorship), and additional approximations on structural conditions (sibling relationship), that we call *structure unbalances*, as shown in the following example:

Example 1. Recall the sample query presented in the Introduction: *Retrieve stores in New York city selling CD's authored by Elton John, and containing*

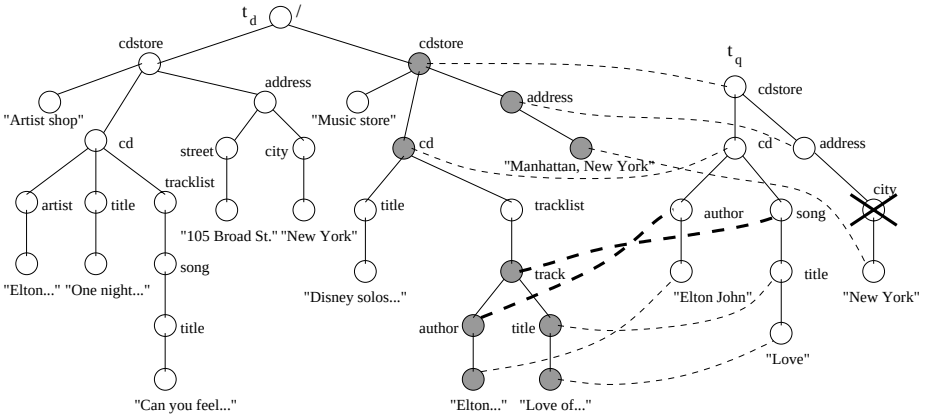

Fig. 2. Partial coverage and unbalance of query tree in `Doc1`

songs with "love" in the title. Consider Fig. 2, where a possible result is shaded in the `Doc1` tree. Note that t_q is only *partially* covered in this solution, since the `city` node does not have any correspondence in t_d. Further, the emphasized dashed lines show a *structure unbalance*.

As to point **2)**, in order to specify the ranking of results, embeddings are to be assigned a *score*, according to a *relevance ranking function* ρ.
Because of the flexibility allowed in the retrieval of embeddings, different types of relaxations are to be taken into account for relevance score computation. For instance, partiality of the $\tilde{e}$ function affects the *completeness* of results. Approximations according to the *a)* point of Def. 1 influence the *semantic correctness* of the retrieved data. Relaxations like those allowed in *b)* and *c)* denote *structural discrepancies* as well as *low cohesion* of data since intermediate nodes are allowed in the document tree. This latest relaxation is evident in Fig. 3 where the shaded solution in the `Doc2` tree presents a relevant CD in the stock "157" of the store's warehouse.
This denotes fragmentation of data retrieved, that contributes to lower the final result score. Our ranking function takes into account all these features, as detailed below.

Definition 2 (Relevance Ranking Function). We denote with ρ a *ranking function* that, given a triple $(t_q, t_d, \tilde{e}[t_q, t_d])$, returns a score $\sigma \in S = [0, 1]$ such that, with respect to the approximate embedding $\tilde{e}$ of the query expressed by t_q in the document expressed by t_d, σ is obtained as a combination of the following components:

- γ_1, that indicates how much $\tilde{e}$ is *semantically complete* with respect to the given query
- γ_2, which denotes *semantic correctness* of $\tilde{e}$, in that it states how well the embedding satisfies semantic requirements

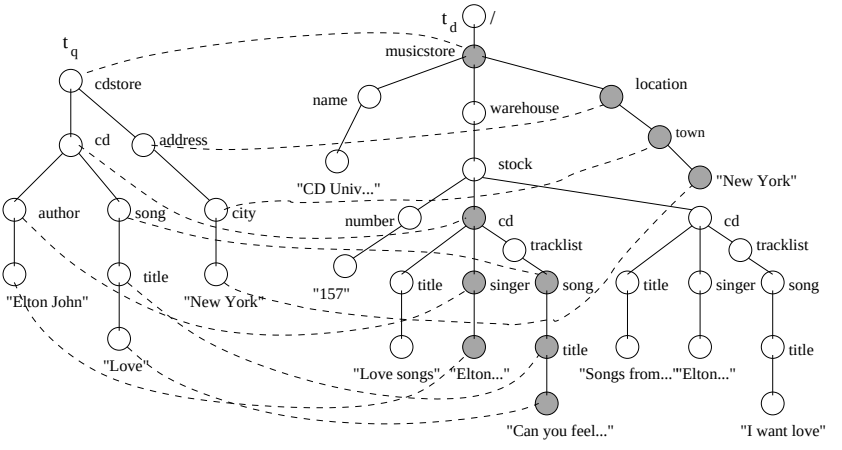

Fig. 3. Partial coverage of query tree and low cohesion in `Doc2`

- γ_3, that represents the *structural completeness* of $\tilde{e}$ with respect to the given query; it denotes the structural *coverage* of the query
- γ_4, that expresses *structural correctness* of $\tilde{e}$, in that it is a measure of how well constraints on structure are respected in the embedding
- γ_5, which specifies *cohesion* of $\tilde{e}$, by providing the grade of fragmentation of the retrieved embedding

Thus, $\sigma = \phi(\gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5)$, with ϕ a *combine* function, states the *overall similarity score* of the embedding $\tilde{e}$. Formally:

$$\rho : \mathcal{T}_Q \times \mathcal{T}_D \times \mathcal{E} \rightarrow S$$

As to the components of a score σ, in Section 5 we will detail each γ_i, when discussing relevance computation. Now we are ready to introduce a *scored approximate tree embedding*.

Definition 3. (Scored Approximate Tree Embedding) A *scored approximate tree embedding* $\tilde{e}_s$ is an approximate tree embedding extended with a *score* in $\mathcal{S}$. Formally: $\tilde{e}_s : \mathcal{S} \times \mathcal{E}$.

With regard to point **3)**, consider Fig. 4. The CD store already retrieved through the embedding of Fig. 3 presents a second relevant CD. Thus, we expect this music store to be a high relevant answer to the user query, since requirements on CD occur twice. On the other hand, `Doc1` contains two relevant stores, as shown in figures 2 and 5. This also increases the relevance of this document.

In order to capture these situations, our tree embedding method considers the presence of multiple occurrences of query requirements, by providing a set-oriented approach to results. This information is then exploited to strengthen the score of *dense* results.

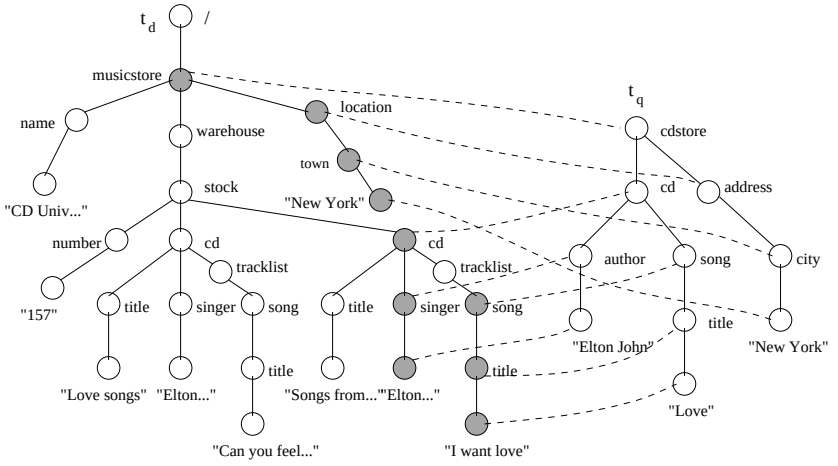

Fig. 4. Further occurrence of query tree in `Doc2`

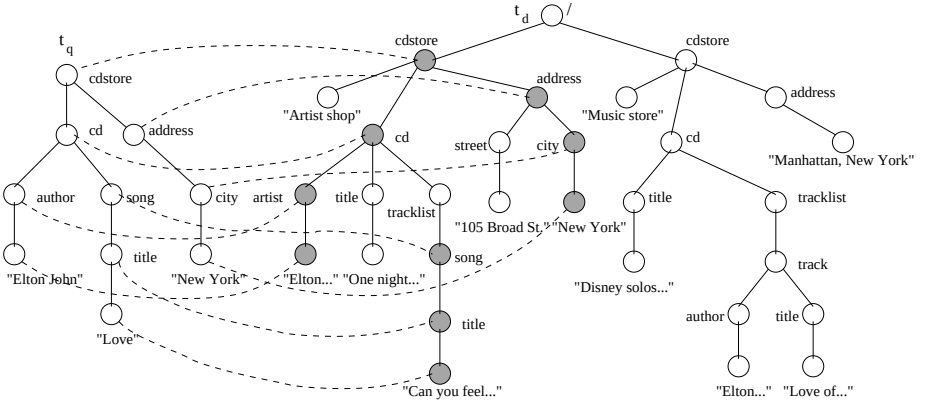

Fig. 5. Full coverage of query tree in `Doc1`

4 Approximate Embedding of Query Trees

The similarity function we propose for retrieval and scoring of embeddings of a query tree in a document tree, is given by the $SATES$ (*Scored Approximate Tree Embedding Set*) *function*.

Definition 4 ($SATES$ Function). We define the *Scored Approximate Tree Embedding Set Function* as:

$$SATES : \mathcal{T}_Q \times \mathcal{T}_D \rightarrow 2^{\mathcal{S} \times \mathcal{E}}$$

$\forall t_q \in \mathcal{T}_Q, \forall t_d \in \mathcal{T}_D$, $SATES(t_q, t_d)$ returns a set of scored approximate tree embeddings for t_q in t_d. The $SATES$ function returns a result set, thus capturing

the possibility of having more than one embedding between a query tree and a document tree.

Intuitively, the $SATES$ function states "how well" a data tree t_d fits a query tree t_q, also taking care of multiple fittings. To determine the scored embeddings, the function is defined in a recursive fashion. In order to show the embedding process, we follow an example-driven approach, and we provide the formal definition of the $SATES$ function in appendix.

We recall the example of Fig. 2, and we try to find an embedding for t_q in t_d. For simplicity, we start considering only the second branch of the t_d tree, i.e. the one relating to `"Music store"`. We refer to it as the *current* t_d, and we denote it with t_d^*. Then, we will show later how the final embedding of t_q in t_d will include also the embedding for the `"Artist shop"` store, thus resulting in a *set* of embeddings. Starting top-down, the method tries to find a match between roots' labels. If a match is possible, then the $SATES$ function recursively applies to children of t_q and t_d^*. This is made according to a bipartite graph matching[1] between the children of t_q and the children of t_d^*, and we denote it with $\mathcal{M}_{t_q}^{t_d}$. In our example, since the labels of t_q's root and t_d^*'s root match, we consider the most promising matching $\mathcal{M}_{t_q}^{t_d}$ between the pairs ($\texttt{address}_q$,$\texttt{address}_d$), and ($\texttt{cd}_q$,$\texttt{cd}_d$), where indices q and d denote the belonging to query and data trees, respectively. In fact, other possible matchings $\mathcal{M}_{t_q}^{t_d}$, involving for instance the `"Music store"` node, do not produce relevant results. However, alternative matchings help in finding structural discrepancies, as we will show later in this Section. Then, the embedding method proceeds recursively on the above-mentioned pairs.

First, let us consider the pair ($\texttt{address}_q$,$\texttt{address}_d$): We have that $t_q^* = supp(\texttt{address}_q)$ and $t_d^* = supp(\texttt{address}_d)$, with t_q^* the *current* t_q. Since roots' labels match, a $\mathcal{M}_{t_q}^{t_d}$ is established between nodes $\texttt{city}_q$ and $\texttt{"Manhattan,..."}_d$. Then, recursion is applied again. However, in this case *no similarity is found* for roots' labels (at present, $t_q^* = supp(\texttt{city}_q)$, and $t_d^* = supp(\texttt{"Manhattan,..."}_d)$). Thus, the method considers two possible strategies:

1. **(optimistic)**: it tries to find a *complete* embedding for t_q^* at a deeper level in t_d^*. This means that the search context is changed to a more specific context;
2. **(pessimistic)**: it intentionally *gives up* looking for a match of t_q^*'s root (thus accepting a *partial match*), and tries to satisfy the remaining query conditions ($t_q^* = supp(\texttt{"New York"}_q)$), by proceeding either:
   a) in the current data context, i.e. t_d^* stays unchanged, or
   b) changing the context, i.e. looking for a match at a deeper level in t_d^*

In the current example, it is easy to notice that the only feasible strategy is 2a. In fact, since t_d^* is indeed a *leaf*, it is not possible to change any search context,

[1] Consider a graph $G = (V, E)$. G is *bipartite* if there is a partition $V = A \cup B$ of the nodes of G such that every edge of G has one endpoint in A and one endpoint in B. A *matching* is a subset of the edges no two of which share the same endpoint. In our case $A = children(root(t_q))$ and $B = children(root(t_d^*))$.

thus making alternatives 1 and 2b not applicable. Then recursion concludes for this branch (i.e. for the pair ($\texttt{address}_q$,$\texttt{address}_d$)), since the current trees are actually two leaves, and a match is found for them.

Now, let us get back to the pair ($\texttt{cd}_q$,$\texttt{cd}_d$): We have that their labels match, and two matchings $\mathcal{M}_{t_q}^{t_d}$ are possible among their children:

1. Let consider first $\mathcal{M}_{t_q}^{t_d} = \{(\texttt{author}_q,\texttt{title}_d),(\texttt{song}_q,\texttt{tracklist}_d)\}$. It is easy to show that the former pair does not produce any result; on the other hand, in the latter one we follow strategy 1, and we recursively compute $SATES$ on the pair ($\texttt{song}_q$,$\texttt{track}_d$). This time, labels are similar and the embedding process goes on, matching ($\texttt{title}_q$,$\texttt{title}_d$), since the pair ($\texttt{title}_q$,$\texttt{author}_d$) does not return any result. Finally, the pair ("Love"$_q$,"Love of ..."$_d$) is added to the embedding.
2. As to the second possible matching $\mathcal{M}_{t_q}^{t_d} = \{(\texttt{song}_q,\texttt{title}_d), (\texttt{author}_q,\texttt{tracklist}_d)\}$, once again we neglect the former pair. In fact, although an embedding is retrieved, i.e. $\{(\texttt{title}_q,\texttt{title}_d)\}$, the corresponding leaves do not match, thus making the embedding not significant.[2] With regard to the second pair, the double application of strategy 1 allows for discovering the embedding of $(\texttt{author}_q,\texttt{author}_d)$, including the pair ("Elton John"$_q$,"Elton John"$_d$). On the other hand, similarly to the previous case, the pair $(\texttt{author}_q,\texttt{title}_d)$ leads to empty results.

Thus, in order to capture structural unbalances, the $SATES$ function takes into account all possible matchings $\mathcal{M}_{t_q}^{t_d}$. Finally, for the sake of completeness, recall that the starting embedding was $SATES(t_q, t_d)$. The set-oriented approach of our method allows for returning two possible solutions, through the double application of strategy 1: The former time to try the embedding of the pair $(\texttt{cdstore}_q,\texttt{cdstore}'_d)$, and the latter one for the pair $(\texttt{cdstore}_q,\texttt{cdstore}''_d)$, where the number of apices denotes the first and second CD store in `Doc1`, respectively. A further situation where the set-oriented approach contributes in the retrieval of multiple occurrences of results is shown in figures 3 and 4, where the final embedding set is obtained by the union of the embeddings originated by the pairs $(\texttt{cd}_q,\texttt{cd}'_d)$ and $(\texttt{cd}_q,\texttt{cd}''_d)$, respectively.

5 Relevance Computation

Most approaches [4,7,10,17,20] score results with ranking values that, individually, do not provide information on the *query rate* satisfied by an answer. Assume, for instance, to compare results coming from two different queries q_1 and q_2. According to most scoring methods [4,17], it is possible that one document that satisfies 1 condition (out of 2) of q_1 is assigned the same score of a document

[2] This policy privileges embeddings with query leaves matching. This is to limit the huge amount of results that satisfy structural constraints, but do not match content conditions. From a strict structural point of view, the above pair should belong to the embedding.

that satisfies 9 conditions (out of 10) of the more complex query q_2. Although results are incomparable, one would expect documents (exactly) satisfying a high percentage of conditions to score higher than documents (exactly) satisfying a lower rate. Thus, besides information on *correctness* of results, a measure of *completeness* is desirable. As to XML documents, this information is somehow made more complex by the presence of structure inside documents. This implies that some knowledge on completeness is supposed to provide also information on the matching rate of query structure. Then, apart from queries where the user explicitly specifies not to take care of the depth where information may be found in, also *cohesion* of data retrieved is an important element to be considered when ranking results.

Here, we provide a scoring method that captures the above-mentioned features. We start modelling a set of properties for each embedding $\tilde{e}$. Property values are normalized in the interval $[0, 1]$, where values close to 1 denote high satisfaction. Properties are:

Semantic Completeness. It is a measure of how much the embedding is semantically complete with respect to the given query. It is computed it as the ratio between the number of query nodes in the embedding, $n_q^{\tilde{e}}$, and the total number of query nodes, n_q:

$$\gamma_1 = \frac{n_q^{\tilde{e}}}{n_q}$$

Semantic Correctness. It states how well the embedding satisfies semantic requirements. It represents the overall semantic similarity captured by the nodes in the embedding. This is computed as a combination of label similarities of matching nodes, possibly lowered by type mismatchings (attribute vs. element nodes):

$$\gamma_2 = \bigwedge_{q_i \in dom(\mathcal{E})} sim(label(q_i), label(\tilde{e}(q_i)))$$

where $\wedge$ is a scoring function [9] the computes the conjunction of label similarities. For instance, $\wedge$ could be a fuzzy *t-norm* [13], such as the *min* function or the product operator.

Structural Completeness. It represents the *structural coverage* of the query tree. It is computed as the ratio between: 1) the number of node pairs in the image of the embedding, $hp_q^{\tilde{e}}$, that satisfy the same hierarchical[3] relationship of the query node pairs which are related to, and 2) the total number of hierarchy-related pairs in the query tree (hp_q):

$$\gamma_3 = \frac{hp_q^{\tilde{e}}}{hp_q}$$

Structural Correctness. It is a measure of how many nodes respect structural constraints. It is computed as the complement of the ratio between the

[3] Either parent-child or ancestor-descendant relationship.

number of structural penalties p (i.e. unbalances) and the total number of hierarchy-related pairs in the data tree that also appear in the embedding:

$$\gamma_4 = 1 - \frac{p}{hp_d^{\tilde{e}}}$$

Cohesion of results. It represents the grade of fragmentation of the resulting embedding. It is computed as the complement of the ratio between the number of intermediate data nodes $in_d^{\tilde{e}}$ among the nodes in the embedding, and the total number of data nodes in the embedding, also including the intermediate ones, $n_d^{\tilde{e}}$:

$$\gamma_5 = 1 - \frac{in_d^{\tilde{e}}}{n_d^{\tilde{e}}}$$

These properties can be naturally partitioned in two sets: properties related to semantics, and properties concerning structure. The combination of the first two scores provides the overall information on *semantic satisfaction*. As to the remaining properties, they all provide different perspectives for the evaluation of structure similarity. A combination of them indicates a global measure, that summarizes the *structural satisfaction* of the retrieved data. However, it is beyond the scope of this paper to evaluate which is the best function to be used for combining these scores in the computation of the overall score σ of an embedding, as defined in Def. 2. Clearly, additional flexibility can be reached by assigning *weigths* to each γ_i to denote the different importance of each property.

5.1 Comparison with Related Approaches

Table 1 shows a comparison of our ranking method with other similar approaches, according to different relaxations on structure.

Table 1. Relaxations supported on structure

Approach	partial match	unbalance	intermed. data nodes	multiple occurr.
XXL [20]	no	no	yes	no
XIRQL [10]	no	no	yes	no
Damiani et al. [7]	no	no	yes	no
Amer-Yahia et al. [4]	no	yes[4]	yes	no
ApproXQL [17]	yes	no	yes	no
SATES	yes	yes	yes	yes

All the discussed methods allow for intermediate data nodes in the results. However, all but ApproXQL do not consider the number of exceeding data nodes in the computation of scores. In fact, all methods except ApproXQL would return

[4] *Subtree Promotion* in [4] captures the unbalances described in Def. 1, point *b)*. The symmetric case, point *c)*, is not discussed.

the same score for the approximate tree embeddings of figures 4 and 5, even though the former embedding presents a higher fragmentation.[5]

However, consider the query embedding of Fig. 6, where the current query is simplified with respect to the sample we used throughout the paper, since it does not require any constraint on CD store location.

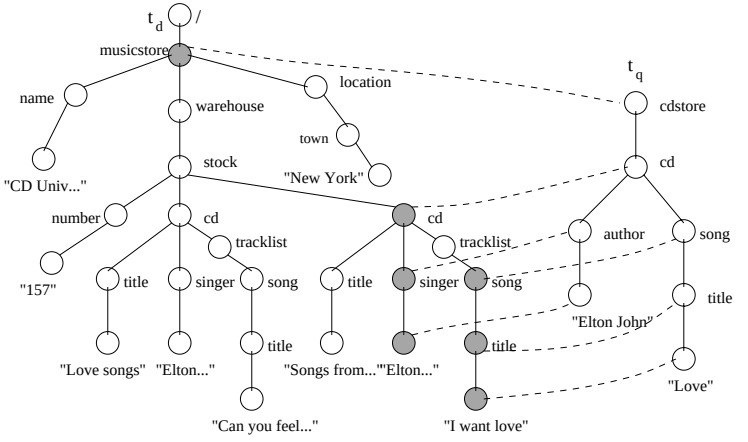

Fig. 6. Embedding of the simplified sample query

Table 2 shows how ApproXQL and *SATES* evaluate the embeddings of figures 4 and 6. For ApproXQL, the same score would be assigned in both cases, since the

Table 2. Different scoring in presence of different cohesion of data

Approach	Score for embedding of Fig. 4	Score for embedding of Fig. 6
ApproXQL	high	high
SATES	high	medium

number of relaxations to meet data organization can be quantified in the same number of intermediate nodes for both queries (3 nodes, indeed). Nevertheless, the set of query requirements is different, and the number of exceeding nodes is proportionally more heavy for the embedding of Fig. 6. *SATES* takes into account this feature, and assigns two different scores to results, scoring higher the embedding of Fig. 4.

[5] For simplicity, we assume the matchings on labels to score the same for both embeddings, so that only evaluation on structure affects the ranking.

6 Implementation

We implemented a prototype and experienced our similarity measure on a real collection of XML documents. The dataset used is provided by the Astronomical Data Center [1]. The prototype has been developed in Java2 v.1.3.1 and uses the API for XML parsing, and Xerces 1.4.4 to parse documents. As to semantic similarity, the system refers to the WordNet semantic network, [2] exploiting relationships like synonymy, hyperonymy, and holonymy among terms, and accesses it through the Java WordNet Library (JWNL). As a similarity measure we used an adaptation of the Sussna's formula [5]. We plan to experience and refine our method, in order to find the better rules to be used for our *combine* and *ranking* functions, with the aim of representing results at the best of their meaning.

7 Conclusions and Future Work

Most of the existing ranking approaches are inadequate when querying heterogeneous collections of XML documents. In fact, as to similarity on documents' structure, they basically require *all* query conditions are somehow preserved in the data retrieved. Also, they do not fully exploit relaxations on structure to capture slightly different organization of data.

We presented an approach that widens the spectrum of relevant data, including solutions that also partially satisfy query requirements, and that approximate text organization inside documents, also capturing unbalances of structure and multiple occurrences of query conditions. Then, we rank results according to a set of properties of the retrieved data. These properties provide, besides correctness, a measure of completeness of query satisfaction, as well as knowledge about cohesion of results. In order to augment query flexibility we plan to allow the user to express preferences on either the semantics or the structure of a query. As to queries with conditions referring to ordered data, we intend to study constraints on our generic unordered tree embedding method. We are aware of the problem of blindly querying XML data: Documents might be organized largely differently from the user's point of view. Thus, we plan to study new complex hierarchical relaxations on data, and to make this process transparent to the user. With regard to implementation, in order to cope with the possible hugeness of approximate results returned, we plan to use threshold conditions, to keep only the most relevant results.

References

1. ADC XML Resources Home Page. `http://xml.gsfc.nasa.gov/`.
2. WordNet Home Page. `http://www.cogsci.princeton.edu/ wn/`.
3. XML Information Set. `http://www.w3.org/TR/xml-infoset`.
4. S. Amer-Yahia, S. Cho, and D. Srivastava. Tree Pattern Relaxation. In *Proc. of the 8th Int. Conf. on Extending Database Technology (EDBT 2002)*, March 2002.

5. A. Budanitsky. Lexical Semantic Relatedness and its Application in Natural Language Processing. Technical Report CSRG-390, University of Toronto, 1999.
6. S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. XML-GL: A Graphical Language for Querying and Restructuring XML Documents. *Computer Networks*, 31:1171–1187, 1999.
7. E. Damiani and L. Tanca. Blind Queries to XML Data. In *In Proc. of Int. Conf. on Database and Expert Systems Applications (DEXA)*, pages 345–356, 2000.
8. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A Query Language for XML. In *Proc. Int. World Wide Web Conference)*, Canada, 1999.
9. R. Fagin. Combining Fuzzy Information from Multiple Systems. In *Proceedings of the 15th ACM Symposium on Principles of Database Systems (PODS'96)*, pages 216–226, Montreal, Canada, June 1996.
10. N. Fuhr and K. Großjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In *Proc. ACM SIGIR Conference*, 2001.
11. R. Goldman, J. McHugh, and J. Widom. From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. In *In Proc. of 2nd Int. Workshop on the Web and Databases (WebDB'99)*, Philadelphia, PA, June 1999.
12. P. Kilpeläinen. *Tree Matching Problems with Application to Structured Text Databases*. PhD thesis, Dept. of Computer Science, Univ. of Helsinki, SF, 1992.
13. G.J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic*. Prentice Hall PTR, 1995.
14. J. Madhavan, P. A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *Proc. of the 27th VLDB Conf.*, pages 49–58, Rome, Italy, 2001.
15. S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In *Proc. of the 18th Int. Conf. on Data Engineering (ICDE 2002)*, San Jose, CA, March 2002.
16. J. Robie, L. Lapp, and D. Schach. XML Query Language (XQL). In *Proc. of the Query Language Workshop (QL'98)*, Cambridge, Mass., 1998.
17. T. Schlieder. Similarity Search in XML Data Using Cost-Based Query Transformations. In *Proc. of 4th Int. Work. on the Web and Databases (WebDB01)*, 2001.
18. T. Schlieder and H. Meuss. Result Ranking for Structured Queries against XML Documents. In *DELOS Workshop: Information Seeking, Searching and Querying in Digital Libraries*, 2000.
19. T. Schlieder and F. Naumann. Approximate Tree Embedding for Querying XML Data. In *In Proc. ACM SIGIR Workshop on XML and Information Retrieval*, Athens, Greece, July 2000.
20. A. Theobald and G. Weikum. Adding Relevance to XML. In *Proc. 3rd Int. Workshop on the Web and Databases (WebDB 2000)*, pages 35–40, May 2000.
21. J. Wolff, H. Florke, and A. Cremers. Searching and Browsing Collections of Structural Information. In *Proc. of the IEEE Advances in Digital Libraries*, pages 141–150, USA, May 2000.

A The *SATES* Function

For simplicity, let $sim(t_q, t_d)$ be $sim(label(root(t_q)), label(root(t_d)))$, and let $s(t_q, t_d) = \rho(t_q, t_d, \{root(t_q), root(t_d)\})$. We use $\mathcal{M}$ in place of $\mathcal{M}_{t_q}^{t_d}$. The $\ominus_q$ and $\ominus_d$ functions change the scores of a set of approximate tree embeddings, according to a lowering factor. New scores capture the unsuccessful match of a query node and a data node, respectively. The $\otimes$ function generates a new score from a set of n given scores in $\mathcal{S}$.

$\forall t_q \in \mathcal{T}_Q, t_d \in \mathcal{T}_D, SATES(t_q, t_d)$ is defined as:

case $leaf(root(t_q)) \wedge leaf(root(t_d))$:
  if $sim(t_q, t_d) > 0$
    $SATES(t_q, t_d) = \{[(s(t_q, t_d), \{(root(t_q), root(t_d))\}]\}$
  else $SATES(t_q, t_d) = \emptyset$

case $leaf(root(t_q)) \wedge \neg leaf(root(t_d))$:
  if $sim(t_q, t_d) > 0$
    $SATES(t_q, t_d) = \{[(s(t_q, t_d), \{(root(t_q), root(t_d)\})]\}$
  else $SATES(t_q, t_d) = \bigcup_{c \in children(t_d)} \ominus_d (SATES(t_q, supp(c)))$

case $\neg leaf(root(t_q)) \wedge leaf(root(t_d))$:
  if $sim(t_q, t_d) > 0$
    $SATES(t_q, t_d) = \{[(s(t_q, t_d), \{(root(t_q), root(t_d)\})]\}$
  else $SATES(t_q, t_d) = \bigcup_{c \in children(t_q)} \ominus_q (SATES(supp(c), t_d))$

case $\neg leaf(root(t_q)) \wedge \neg leaf(root(t_d))$:
  if $sim(t_q, t_d) > 0$ $SATES(t_q, t_d) =$

$$\bigcup_{\mathcal{M} \in \mathcal{M}_{t_q}^{t_d}} \bigcup_{\substack{(t_i^k, t_j^k) \in \mathcal{M} \\ (s_{l_k}^k, m_{l_k}^k) \in SATES(t_i^k, t_j^k) \\ l_k \in [1..|SATES(t_i^k, t_j^k)|]}} [\otimes(s(t_q, t_d), s_{l_1}^1, \ldots, s_{l_{|\mathcal{M}|}}^{|\mathcal{M}|}), \{(root(t_q), root(t_d))\} \cup m_{l_1}^1 \cup \ldots \cup m_{l_{|\mathcal{M}|}}^{|\mathcal{M}|}]$$

  else $SATES(t_q, t_d) = \bigcup(\bigcup_1, \bigcup_2, \bigcup_3)$

$$\text{where } \bigcup_1 = \bigcup_{\mathcal{M} \in \mathcal{M}_{t_q}^{t_d}} \ominus_q \circ \ominus_d \bigcup_{\substack{(t_i^k, t_j^k) \in \mathcal{M} \\ (s_{l_k}^k, m_{l_k}^k) \in SATES(t_i^k, t_j^k) \\ l_k \in [1..|SATES(t_i^k, t_j^k)|]}} [\otimes(s_{l_1}^1, \ldots, s_{l_{|\mathcal{M}|}}^{|\mathcal{M}|}), m_{l_1}^1 \cup \ldots \cup m_{l_{|\mathcal{M}|}}^{|\mathcal{M}|}]$$

$$\bigcup_2 = \bigcup_{c \in children(t_d)} \ominus_d (SATES(t_q, c))$$

$$\bigcup_3 = \bigcup_{c \in children(t_q)} \ominus_q (SATES(c, t_d))$$