

Relevance Ranking Tuning for Similarity Queries on XML Data

Paolo Ciaccia and Wilma Penzo

DEIS - CSITE-CNR

University of Bologna, Italy

{pciaccia,wpenzo}@deis.unibo.it

Abstract. Similarity query techniques integrating semantics and structure of XML data have been recently investigated. Mostly, query relaxations are not fully exploited to retrieve data that approximate query conditions on data organization. In this paper we present a method to widen the spectrum of relevant data, and we define a measure to *tune* the ranking of results, so that also information on *relevance quality* of data retrieved can be inferred from scores.

1 Introduction and Motivation

Several researchers have recently profused their efforts in the integration of semantics and structure for querying XML data [5, 7, 13]. Because of the heterogeneity of large digital libraries, and in absence of knowledge of data organization, Boolean query conditions often lead to empty results. Thus, relaxation on query requirements has been acknowledged as a “must”. As to data content, some proposals suggest the use of vague predicates to formulate queries [7, 13]. These assume thesauri, ontologies, and semantic networks to find out the relevance of terms inside documents. As to data organization, relaxations of structural conditions only allow for “larger results”, where *all* hierarchical dependencies between data are (somehow) preserved [11]. This is the case of path expressions containing wildcards. The basic idea is that results are relevant no matter how “sparse” they are. This flattens the relevance of the retrieved data. In fact, although all required information is found, it possibly appears in different contexts, and this is supposed to affect relevance. Further, a total match approach is restrictive, since it limits the set of relevant results. Actually, most of existing works do not cope with the problem of providing *structurally incomplete results*, i.e. results that only partially satisfy query requirements on text organization.

Let us consider, for instance, a user looking for stores selling CD’s authored by a singer whose lastname is “John”¹, and containing songs with “love” in the title. Among the documents shown in Fig. 1, only Doc1 fully satisfies query requirements. This is because Doc1 is the only document that presents a “lastname” tag, thus making condition on “John” checkable. But, it is evident that also documents Doc2 is relevant to the query, and should be returned.

¹ Suppose the user does not remember the firstname “Elton” and, since “John” is a very common name, she wants to specify that “John” must be a lastname.

<pre> <cdstore>Artist Shop <cd> <title>One night only </title> <singer> Elton John </singer> <song> <title> Can you feel the love tonight </title> </song> </cd> </cdstore> <cdstore>Music Store <cd> <title> Chartbusters go pop </title> <singer> <firstname> Elton </firstname> <lastname> John </lastname> </singer> <song> <title> Love of the common people </title> </song> </cd> </cdstore> </pre> <p style="text-align: center;">Doc1</p>	<pre> <musicstore name="CD Universe"> <cd> <title> Love songs </title> <singer> Elton John </singer> <year>1996</year> <song> <title> Can you feel the love tonight </title> <lyric> There's a calm surrender to the rush of day ... </lyric> </song> </cd> <cd> <title> Songs from the west coast </title> <singer> Elton John </singer> <year>2001</year> <song> <title> I want love </title> </song> </cd> </musicstore> </pre> <p style="text-align: center;">Doc2</p>	<pre> <cdshop> Music Planet <cd> <title> Disney solos for violin </title> <price>£9.95</price> <tracklist> <track> <author> Elton John </author> <title> Can you feel the love tonight </title> </track> <track> <author> Alan Menken </author> <title> The bells of Notre Dame </title> </track> <track> <author> Elton John </author> <title> Circle of life </title> </track> </tracklist> </cd> </cdshop> </pre> <p style="text-align: center;">Doc3</p>
--	--	--

Fig. 1. Sample XML documents

The above-cited approaches do not capture this approximation, except for ApProXML [12] and [5]. In these proposals, however, both Doc1 and Doc2 would be evaluated with the same score, although Doc2 contains two relevant CD's rather than only one as in Doc1. Further, also Doc3 is expected to be returned as relevant, and it is not. This is because data is organized in a slight different way: Elton John appears as a song author rather than as the CD author, as specified by the query. This structure disagreement does not allow to recognize Doc3 as relevant. This points out that evaluation of structure conditions should be made more flexible. Basically, a finer *tuning* of relevance scores should be provided: partial matches on structure, as well as approximations, actually are supposed to influence relevance, but should not be the cause of data discarding. Then, documents that present more occurrences of query requirements are expected to score higher, and this should be captured by an effective measure of relevance. Current methods usually produce absolute ranking values that, individually, do not provide information on the query rate satisfied by an answer. For instance, it is not possible to realize if, given a query, a document is assigned a low score because of an approximate yet complete match, or because of a partial yet exact match of query requirements. Since the output of a query is usually large, additional information that justifies scores would lighten the user from

the burden of discovering which solutions come up better to her expectations. Thus, in general, an *effective* relevance ranking method is expected to provide information to infer the overall *quality* of data retrieved.

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 measure that, besides ranking, also specifies *quality* of results. The outline is as follows: In Section 2 we introduce a basic query language and a tree representation for queries and documents. In Section 3 we start from the unordered tree inclusion problem [9] 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. To this end, we define the *SATES* (*Scored Approximate Tree Embedding Set*) function, to retrieve and score embeddings. We also show how some “critical” queries are effectively managed by our method. Our relevance ranking measure is then presented. Then, we compare our method with other approaches in Section 4 and, finally, in Section 5 we conclude and discuss future directions we intend to follow.

2 XML Query Language and Trees

XML (*eXtensible Markup Language*) is a markup language for defining content and organization of text documents. For the sake of simplicity, we do not require validity for XML documents, in that we do not assume the presence of DTDs. We also do not consider further features (e.g. IDREFs, namespaces) provided by the XML specification [2].

As a starting point, we consider a subset of the XQuery1.0 grammar [4], that we call $XQuery^-$. Queries are path expressions, with predicates restricted to equality on attribute and element values. An $XQuery^-$ expression has the form:

$$[\langle sep \rangle] (\langle StepExpr \rangle \langle sep \rangle)^* \langle PrimaryExpr \rangle ["\langle Expr \rangle"]$$

with the $\langle sep \rangle$ hierarchical separator having possible values “/” and “//” to denote parent-child and ancestor-descendant relationships, respectively; the expression $(\langle StepExpr \rangle \langle sep \rangle)^* \langle PrimaryExpr \rangle$ denotes a path in a document where conditions are expected to be checked in; $\langle PrimaryExpr \rangle$ indicates the query output; conditions are expressed between square brackets, as a boolean combination of predicates. The following is an example of complex query:

Example 1 Retrieve data about musical CDs on sale at stores in Manhattan, where author’s lastname is “John”, being produced in “1996”, and containing songs’ lyrics:

$$cdstore/cd[author/lastname='John' \text{ and } ../address/* = 'Manhattan' \text{ and } @year='1996' \text{ and } ../song/lyric]$$

$XQuery^-$ expressions are mapped to trees in a natural way [3]. Trees are made of typed labelled nodes and (possibly labelled) edges. Nodes may have the following types: **element**, **attribute** and **content**. A **content** leaf expresses the value its

parent `element/attribute` node is required to assume. An `element/attribute` leaf specifies a condition on the presence of that specific attribute/element in the given context. Non-leaf nodes denote the context where information is expected to be found in. Finally, each edge connecting two nodes either model simple parent-child relationship, or it may be labelled with the “*” symbol, to denote a path of positive length, thus expressing an ancestor-descendant relationship.

We briefly recall some basic definitions on trees, and we introduce some additional functions. A *tree* t is a pair (N, E) , with N finite set of typed labelled nodes, and E binary relation on N . Hierarchical relationships on a tree t , are defined as: $\forall n_1, n_2 \in N, 1) \text{parent}(n_1, n_2) \iff (n_1, n_2) \in E; 2) \text{ancestor}(n_1, n_2) \iff (\text{parent}(n_1, n_2) \vee \exists n \in N \text{ such that } \text{parent}(n_1, n) \wedge \text{ancestor}(n, n_2))$.

$\exists! r \in N$ such that: 1) $\nexists n \in N$ such that $\text{parent}(n, r)$; 2) $\forall n \in N, n \neq r, \text{ancestor}(r, n)$. r is called the *root* of the tree. $\forall n_2 \in N, \exists! n_1 \in N$ such that $\text{parent}(n_1, n_2)$. If $\text{ancestor}(n_1, n_2)$ holds, the sequence of edges that connect n_1 with n_2 is called a *path* from n_1 to n_2 .

Let \mathcal{T} be the set of all typed and labelled trees: $\forall t \in \mathcal{T}, \text{root}(t)$ and $\text{nodes}(t)$ return the root and the set of all nodes of the tree t , respectively. Given a tree $t = (N, E), \forall n \in N, \text{label}(n)$ and $\text{type}(n)$ return the label $l \in \mathcal{L}$, with \mathcal{L} set of labels, and the type $k \in \mathcal{K}$, with \mathcal{K} set of node types, for a node n , respectively; $\text{children}(n)$ returns the set of all children of node n ; $\text{leaf}(n) \iff \text{children}(n) = \emptyset$; $\text{support}(n)$ is the subtree of t rooted at n , called the *support* of n in t .

We provide a tree representation also for documents so as to reason on query and document similarity by means of tree comparison. We want to determine how much a document tree satisfies semantics and structural constraints provided by a query tree. Let \mathcal{D} be the set of well-formed XML documents.

Definition 1 (From XML Docs to trees) Given a document $d \in \mathcal{D}$, its corresponding tree $t_d \in \mathcal{T}$, is such that each element, attribute, and content data in d is mapped to a node n_e, n_a , and n_c , in t_d , respectively, and:

1. $\text{type}(n_e) = \text{element}, \text{type}(n_a) = \text{attribute}, \text{type}(n_c) = \text{content}$;
2. labels of nodes are element’s and attribute’s names, and content data values;
3. each nesting level of element/attribute/content data in d resolve to a *parent-child* relationship between corresponding nodes in t_d .

Let $\mathcal{T}_Q \subset \mathcal{T}$ and $\mathcal{T}_D \subset \mathcal{T}$ be the sets of query and data trees, respectively.

3 The Tree Embedding Problem

Satisfying a query q on a document d may lead to different results, depending on how much we are willing to relax constraints on query semantics and requirement dependencies. In our tree view, when a query has to be completely satisfied, we look for an *embedding* of a query tree t_q in a document tree t_d . This means that 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 [9]. In many cases these conditions are too restrictive, since data may not correspond completely and exactly to query requirements.

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

1. the relaxation on the concept of *total* embedding of t_q in t_d , in that we admit *partial* structural match of the query tree; further, exact match is relaxed to consider semantic similarity between nodes;
2. approximate results, that are *ranked* with respect to the *cohesion* of retrieved data, to the relaxation of semantic and structural constraints, and to the *coverage rate* of the query. Our ranking function takes into account the overall query satisfaction, in that a score provides a ranking value but also a *quality measure* of results;
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 strengthens the relevance of data presenting multiple occurrences of query patterns.

Point 1) leads to the introduction of our interpretation of *approximate tree embedding*.

Definition 2 (Approximate Tree Embedding) 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 $\tilde{e}[t_q, t_d] : nodes(t_q) \rightarrow nodes(t_d)$ such that $\forall q_i, q_j$ in the domain of \tilde{e} ($dom(\tilde{e})$):

1. $sim(label(q_i), label(\tilde{e}(q_i))) > 0$, with sim a similarity operator that returns a score in $[0,1]$ stating the semantic similarity between the two given labels
2. $parent(q_i, q_j) \Rightarrow ancestor(\tilde{e}(q_i), \tilde{e}(q_j))$

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

As to the point 2), in order to specify the ranking of results, embeddings are to be assigned a *score*, according to a *relevance ranking function* ρ .

Definition 3 (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 tuple:

$$\Sigma = (\sigma, \gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5)$$

of *scores* with values in $\mathcal{S} = [0, 1]$ such that, with respect to the approximate embedding \tilde{e} of the query expressed by t_q in the document expressed by t_d :

- γ_1 indicates how much \tilde{e} is *semantically complete* with respect to the query;
- γ_2 denotes *semantic correctness* of \tilde{e} , in that it states how well the embedding satisfies semantic requirements;
- γ_3 represents the *structural completeness* of \tilde{e} with respect to the given query; it denotes the *structural coverage* of the query;
- γ_4 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 specifies *cohesion* of \tilde{e} , by providing the grade of fragmentation of the retrieved embedding;
- σ states the *overall similarity score* of the embedding, and it is obtained as a combination of $\gamma_1, \gamma_2, \gamma_3, \gamma_4$, and γ_5 .

Formally: $\rho : \mathcal{T}_Q \times \mathcal{T}_D \times \mathcal{E} \rightarrow \mathcal{S}^6$

$$\begin{aligned}
& \forall t_q \in \mathcal{T}_Q, t_d \in \mathcal{T}_D, \text{SAT}ES(t_q, t_d) \text{ is defined as:} \\
& \text{case } \text{leaf}(\text{root}(t_q)) \wedge \text{leaf}(\text{root}(t_d)): \\
& \quad \text{if } \text{sim}(t_q, t_d) > 0 \\
& \quad \quad \text{SAT}ES(t_q, t_d) = \{[(s(t_q, t_d), \{\text{root}(t_q), \text{root}(t_d)\})]\} \\
& \quad \text{else } \text{SAT}ES(t_q, t_d) = \emptyset \\
& \text{case } \text{leaf}(\text{root}(t_q)) \wedge \neg \text{leaf}(\text{root}(t_d)): \\
& \quad \text{if } \text{sim}(t_q, t_d) > 0 \\
& \quad \quad \text{SAT}ES(t_q, t_d) = \{[(s(t_q, t_d), \{\text{root}(t_q), \text{root}(t_d)\})]\} \\
& \quad \text{else } \text{SAT}ES(t_q, t_d) = \bigcup_{c \in \text{children}(t_d)} \ominus_d (\text{SAT}ES(t_q, \text{support}(c))) \\
& \text{case } \neg \text{leaf}(\text{root}(t_q)) \wedge \text{leaf}(\text{root}(t_d)): \\
& \quad \text{if } \text{sim}(t_q, t_d) > 0 \\
& \quad \quad \text{SAT}ES(t_q, t_d) = \{[(s(t_q, t_d), \{\text{root}(t_q), \text{root}(t_d)\})]\} \\
& \quad \text{else } \text{SAT}ES(t_q, t_d) = \bigcup_{c \in \text{children}(t_q)} \ominus_q (\text{SAT}ES(\text{support}(c), t_d)) \\
& \text{case } \neg \text{leaf}(\text{root}(t_q)) \wedge \neg \text{leaf}(\text{root}(t_d)): \\
& \quad \text{if } \text{sim}(t_q, t_d) > 0 \text{ SAT}ES(t_q, t_d) = \\
& \quad \bigcup_{\substack{\mathcal{M}_{t_q}^{t_d} \\ (t_i^k, t_j^k) \in \mathcal{M} \\ (s_{l_k}^k, m_{l_k}^k) \in \text{SAT}ES(t_i^k, t_j^k) \\ l_k \in [1..|\text{SAT}ES(t_i^k, t_j^k)|]}} \bigcup_{(s_{l_1}^1, \dots, s_{l_{|\mathcal{M}|}}^{|\mathcal{M}|}), \{\text{root}(t_q), \text{root}(t_d)\} \cup m_{l_1}^1 \cup \dots \cup m_{l_{|\mathcal{M}|}}^{|\mathcal{M}|}} \\
& \quad \text{else } \text{SAT}ES(t_q, t_d) = \bigcup (\bigcup_1, \bigcup_2, \bigcup_3) \\
& \quad \quad \text{where } \bigcup_1 = \bigcup_{\substack{\mathcal{M}_{t_q}^{t_d} \\ (t_i^k, t_j^k) \in \mathcal{M} \\ (s_{l_k}^k, m_{l_k}^k) \in \text{SAT}ES(t_i^k, t_j^k) \\ l_k \in [1..|\text{SAT}ES(t_i^k, t_j^k)|]}} \ominus_q \circ \ominus_d \bigcup_{(s_{l_1}^1, \dots, s_{l_{|\mathcal{M}|}}^{|\mathcal{M}|}), m_{l_1}^1 \cup \dots \cup m_{l_{|\mathcal{M}|}}^{|\mathcal{M}|}} \\
& \quad \quad \bigcup_2 = \bigcup_{c \in \text{children}(t_d)} \ominus_d (\text{SAT}ES(t_q, c)) \\
& \quad \quad \bigcup_3 = \bigcup_{c \in \text{children}(t_q)} \ominus_q (\text{SAT}ES(c, t_d))
\end{aligned}$$

Fig. 2. The *SAT*ES Function

Now we are ready to introduce a *scored approximate tree embedding*. For the sake of simplicity, we start considering embeddings scored with respect to the overall scoring value σ . Complete relevance score computation is detailed in Section 3.1.

Definition 4 (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: $\tilde{e}_s : \mathcal{S} \times \mathcal{E}$.

The similarity function we propose for retrieval and scoring of embeddings of a query tree in a document tree, is given by the *SAT*ES (*Scored Approximate Tree Embedding Set*) *function*, shown in Fig. 2.

Definition 5 (SATES **Function)** We define the *Scored Approximate Tree Embedding Set Function* as:

$$\text{SAT}ES : \mathcal{T}_Q \times \mathcal{T}_D \rightarrow 2^{\mathcal{S} \times \mathcal{E}}$$

$\forall t_q \in \mathcal{T}_Q, \forall t_d \in \mathcal{T}_D, \text{SATES}(t_q, t_d)$ returns a set of scored approximate tree embeddings for t_q in t_d . This captures 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. In order to determine the scored embeddings, the function follows a recursive definition and examines different cases for t_q and t_d :

Both leaves. Depending on the semantic similarity of node labels, the result set is made of: 1) one scored embedding that relates the two nodes; 2) the emptyset if no similarity is found for labels. If an embedding is returned, since structural similarity between leaves can be considered “perfect”, semantic similarity assumes the key role of determining the final score for the embedding.

Query leaf and Data tree. If the semantic similarity between roots’ labels is positive, an embedding is found and its score is determined as in the previous case. Here note that, even if the structure of the two trees is different (one of them is indeed a leaf), the embedding’s score should not be influenced from this. In fact, we can evidently conclude that the *structural coverage* is complete. In case of null semantic similarity, t_d ’s children are entrusted to determine some possible embeddings for the leaf t_q . Then, the scores of the resulting embeddings are *lowered* through the \ominus_d function,² so that the skip of the root of t_d , that did not match with the query node, is taken into account in the computation of the structural similarity between t_q and t_d . Even if we can say that t_d *covers* t_q , indeed it provides a more generic context, that does not “exactly” satisfy structural requirements (and consequently also semantics). Score lowering can then be considered appropriate.

Query tree and Data leaf. This is the case that concludes recursion in the next step. Its role is made evident by the following case.

Query tree and Data tree. This is the general case that usually starts a *SATES* call. The result set is determined in a recursive fashion. Basically, two cases may occur: either 1) semantic similarity of roots’ labels exists, or 2) no similarity is found. The former is the simplest case. The idea is that, since roots are semantically related, this means that the current *data context* is promising for the examination of remaining query conditions. Thus, query root’s children get involved in the recursive computation of embeddings. Matchings are considered in the bipartite graph³ between query root’s children and data root’s children.

In case 2) of null label similarity, the final embeddings are computed from the union of three quantities, that, for simplicity, we call \cup_1 , \cup_2 , and \cup_3 . Basically, the final embeddings must consider the unsuccessful match of the roots. This implies the final scores to be properly lowered. Then, two strategies can

² Notation details are shown later in this section.

³ 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 $A = \text{children}(\text{root}(t_q))$ and $B = \text{children}(\text{root}(t_d))$.

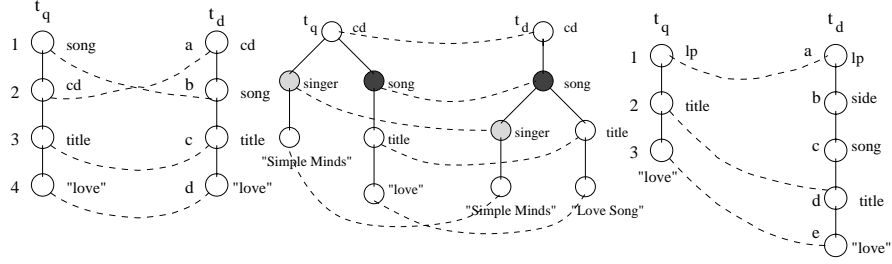


Fig. 3. Swap

Fig. 4. Unbalance

Fig. 5. Low cohesion

be followed: either looking for satisfying the remaining query conditions in the current context, or changing the context. Consider Fig. 3. Assume semantic similarity between roots' labels "song" and "cd" is 0. Before asserting that certainly no embeddings exist, we proceed in two directions: 1) We change the context of our search, trying to satisfy the query in a more specific domain: thus we move our target to the child(ren) of t_d 's root (\cup_2 computation); 2) we give up looking for a complete match of the query, and we try to discover if the current context satisfies at least the remaining conditions (\cup_3 computation). Of course, in both cases result scores are properly lowered to take into account the presence of unmatched nodes. This "crossed comparison" may point out possible "swaps" between query and data nodes. In fact, in our example, two embeddings are retrieved: $\{(1,b),(3,c),(4,d)\}$ and $\{(2,a),(3,c),(4,d)\}$. Even if they are two separate solutions, it is worth noticing that the union of the two embeddings provides a full coverage for the query tree. Thus, a more complete solution could be derived, albeit including score penalization for the swap. Further, \cup_2 computation captures also structural dissimilarities like that shown in Fig. 4 where a possible embedding is dashed. Note that, "song" and "singer" nodes are siblings in t_q and parent-child in t_d . This should penalize the final score of the embedding, and it is captured by the \ominus_d function when computing the embedding of $\text{support}(\text{"singer"})$ in $\text{support}(\text{"song"})$.⁴

Further, when comparing subtrees as an effect of recursive calls of the *SATES* function, the event of unsuccessful match for the document (current) root may be interpreted as a sign of low cohesion of the global result. As an example, let consider the trees of Fig. 5. The evaluation of $SATES(t_q, t_d)$ leads to the embedding $\{(1,a),(2,d),(3,e)\}$. In this case, although the query tree is totally embedded in the document tree, cohesion of retrieved data is rather low. This is captured by the double application of the \ominus_d function, because of the two recursive steps with unsuccessful match of (2,b) and (2,c). Note that the *SATES* function is not symmetric. Consider trees in Fig. 5 again. Finding the "inverse" approximate embedding of t_d in t_q would result in a partial satisfaction of query requirements since "side" and "song" elements do not have a correspondance in t_q . This means that the \ominus_q lowering function "weighs" differently (much more indeed) from \ominus_d . Thus, the retrieved approximate embeddings would be scored differently. This captures the intuition that priority is on satisfaction of query requirements, and then on cohesion of results.

⁴ Note that labels of trees' roots match.

SATES formal notation To reduce notational complexity in the definition of the *SATES* function provided in Fig. 2, we used some abbreviations: $sim(t_q, t_d)$ stands for $sim(label(root(t_q)), label(root(t_d)))$, and $s(t_q, t_d)$ is the score obtained by the ρ function with reference to the base case of embedding $root(t_q)$ in $root(t_d)$. Then, $\mathcal{M}_{t_q}^{t_d}$ is the *Bipartite Graph Matching* between the two sets $children(root(t_q))$ and $children(root(t_d))$. When clearly defined in the context, we use \mathcal{M} in place of $\mathcal{M}_{t_q}^{t_d}$. The \ominus_q , \ominus_d , and \otimes functions are defined as follows.

Definition 6 (Lowering Functions) The \ominus_q and \ominus_d functions change the scores of a given a set of scored approximate tree embeddings, according to a lowering factor. New scores capture the unsuccessful match of a query node and a data node, respectively. Formally:

$\ominus_q: 2^{\mathcal{S} \times \mathcal{E}} \rightarrow 2^{\mathcal{S} \times \mathcal{E}}$ such that $\forall \tilde{e}_s = (s, \tilde{e}) \in dom(\ominus_q) \ominus_q(\tilde{e}_s) = (s', \tilde{e}) \wedge s' \leq s$. \ominus_d is defined similarly.

Definition 7 (Combine Function) The \otimes function generates a new score from a set of n given scores in \mathcal{S} . Formally: $\otimes: 2^{\mathcal{S}} \rightarrow \mathcal{S}$.

3.1 Relevance Computation

An *effective* relevance ranking method is expected to provide information to infer *quality* of data retrieved. Most approaches rank results according to scores that depend on the satisfiability of matching query conditions [7, 13]. Others use cost functions to combine the costs of relaxations required to (also partially) satisfy the query [5, 12]. In this case, combination (usually sum) produces *absolute* ranking values that, individually, do not provide information on how much of the query is satisfied by an answer. Assume to compare results coming from two different queries q_1 and q_2 . According to these scoring functions [5, 12], it is possible that one document that satisfies 1 condition (out of 2) of q_1 is assigned the same score of a document 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 percentual 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, in addition to a measure of semantic satisfaction, as in the keyword case, some knowledge on completeness is supposed to provide also information on the matching rate of query structure. Also, *cohesion* of data retrieved is another important feature to be considered. Cohesion provides a measure of how “sparse” the required information is. Apart from queries where the user explicitly specifies not to take care of the depth where information may be found in, this information is an important element to be considered when ranking data. As to this property, existing approaches [7, 13] do not treat this information. As a consequence, *quality* of results can be considered an overall measure of all these features.

We provide a scoring method that takes into account semantic and structural completeness and correctness of results, as well as cohesion of relevant data

retrieved. We start modeling 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. Then, we discuss how these measures can be combined to obtain an overall score for an embedding. 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 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 \wedge (for instance, product) of label similarities of matching nodes, possibly lowered by type mismatches (attribute vs. element nodes):

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

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 hierarchical⁵ 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 number of structural penalties p (i.e. swaps and 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 $in_d^{\tilde{e}}$ 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. A combination of them indicates a global measure, that summarizes the semantic and structural characteristics of the retrieved data.

⁵ Either parent-child or ancestor-descendant relationship.

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. In general, σ is obtained as a function ϕ of the above properties: $\sigma = \phi(\gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5)$. The combination function ϕ can be derived, for instance, experimentally, or they might be specified by the user. As an example, ϕ can be a linear combination of its operands. Thus, additional flexibility can be reached by assigning *weights* to each γ_i to denote the different importance of each property.

Thus, given an embedding \tilde{e} for a query tree t_q in a document tree t_d , our ranking function ρ provides a *rich* relevance score, equipped with additional information on the grade of satisfaction of the above properties in \tilde{e} . This information helps the user to infer the overall *quality* of results. Further, the final score is enriched with some knowledge on key aspects like *completeness* and *cohesion* of data retrieved, usually neglected in scoring methods.

4 Related Work

A recent initiative of the DELOS Working Group [1] emphasizes the growing importance of evaluating research methods against large collections of XML documents. To this end, approximate matching techniques are acknowledged to be powerful tools to be exploited. Thus, many proposals provide similarity query capabilities that, besides semantics, also examines the structure of documents to rank results [5, 7, 12, 13]. However, these approaches present relevance ranking functions that, in absence of knowledge of DTDs, either neglect some potential solutions or provide too coarse-grained scores that flatten the differences among actually different results. In XXL [13] partial match on query structure is not supported. Similarity basically depends on content similarity, through the evaluation of vague predicates. Flexibility on structural match is obtained through wildcards, but the number of skipped nodes is discarded in relevance computation, thus neglecting the grade of cohesion of data retrieved. XIRQL [7] introduces the concept of *index object* to specify document units as non-overlapping subtrees. Structural conditions only act as filters to determine the context where information has to be searched in. Partial match on query structure yet is not discussed. Few proposals consider partial match on query structure [5, 12]. In ApproXQL [12] similarity scores depend on the costs of basic transformations applied on the query tree. Basic costs are added in a total cost, thus resulting in a *absolute* scoring. Thus, it is not possible to realize the rate of query satisfied by an answer. A similar approach is proposed in [5]. 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, yet producing an absolute scoring as in [12]. Despite the high flexibility provided by relaxations, also in this case information on cohesion of data is not taken into account for relevance. 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 schema matching [10], and the semantic integration of XML data [8].

5 Conclusions and Future Work

We presented an *effective* relevance ranking measure to infer *quality* of results of similarity queries on XML data. Our measure widens the spectrum of relevant data, including solutions that also partially satisfy query requirements, and that approximate text organization inside documents. Then, our relevance ranking method provides a more fine-grained scoring, in that it takes into account the matching rate of query conditions, the cohesion of data retrieved, and the presence of multiple occurrences of query requirements inside data. Thus, besides correctness, a measure of completeness of query satisfaction, as well as knowledge about cohesion of results, are additional key elements to provide information on quality of data retrieved. Several issues urge to be investigated. In order to augment query flexibility the user might be allowed to express preferences on either the semantics or the structure of a query. With regard to query processing, in order to cope with the possible hugeness of approximate results returned, we plan to exploit algorithms from works on top-k queries [6].

References

1. Eval. Initiative for XML Doc. Retrieval. <http://qmir.dcs.qmw.ac.uk/XMLEval.html>.
2. Extensible Markup Language. <http://www.w3.org/TR/2000/REC-xml-20001006>.
3. XML Information Set. <http://www.w3.org/TR/xml-infoset>.
4. XQuery 1.0: An XML Query Language. W3C Working Draft, 2001, <http://www.w3.org/TR/xquery>.
5. 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.
6. N. Bruno, L. Gravano, and A. Marian. Evaluating Top-k Queries over Web-Accessible Databases. In *Proc. of 18th Int. Conf. on Data Engineering (ICDE 2001)*, San Jose, CA, 2001.
7. N. Fuhr and K. Großjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In *Proc. ACM SIGIR Conference*, 2001.
8. E. Jeong and C. Hsu. Induction of Integrated View of XML Data with Heterogeneous DTDs. In *Proc. of 2001 ACM Int. Conf. on Information and Knowledge Management (CIKM 2001)*, Atlanta, USA, November 2001.
9. P. Kilpeläinen. *Tree Matching Problems with Application to Structured Text Databases*. PhD thesis, Dept. of Computer Science, Univ. of Helsinki, SF, 1992.
10. 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.
11. J. Robie, L. Lapp, and D. Schach. XML Query Language (XQL). In *Proc. of the Query Language Workshop (QL'98)*, Cambridge, Mass., 1998.
12. 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.
13. 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.