Available online at www.sciencedirect.com
ScienceDirect
Data & Knowledge Engineering 64 (2008) 266–293
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

Fragment-based approximate retrieval in highly heterogeneous XML collections

I. Sanz [a], M. Mesiti [b,*], G. Guerrini [c], R. Berlanga [a]

[a] Department of Computer Science and Engineering, Universitat Jaume I, Avg. de Vicent Sos Baynat, s/n E-12071 Castelló, Spain
[b] Dipartimento di Informatica e Comunicazione, Università degli Studi di Milano, Via Comelico, 39/41 I-20135 Milano, Italy
[c] Dipartimento di Informatica e Scienze dell'Informazione, Università degli Studi di Genova, Via Dodecaneso, 35 I-16146 Genova, Italy

Received 30 January 2007; accepted 24 May 2007
Available online 6 August 2007

Abstract

Due to the heterogeneous nature of XML data for internet applications exact matching of queries is often inadequate. The need arises to quickly identify subtrees of XML documents in a collection that are similar to a given pattern. Similarity involves both tags, that are not required to coincide, and structure, in which not all the relationships among nodes in the tree structure are strictly preserved.

In this paper we present an efficient approach to the identification of similar subtrees, relying on ad-hoc indexing structures. The approach allows to quickly detect, in a heterogeneous document collection, the minimal portions that exhibit some similarity with the pattern. These candidate portions are then ranked according to their actual similarity. The approach supports different notions of similarity, thus it can be customized to different application domains. In the paper, three different similarity measures are proposed and compared. The approach is experimentally validated and the experimental results are extensively discussed.

Keywords: XML; Approximate structural retrieval; Enhanced indexing techniques

1. Introduction

Interoperability among systems is commonly achieved through the interchange of XML documents that can represent a great variety of information resources: semi-structured data, database schemas, concept taxonomies, ontologies, etc. Most XML document collections are highly heterogeneous from several viewpoints. The first level of heterogeneity is *tag heterogeneity*: vocabulary discrepancies may occur in the element tags. Two elements representing the same information can be labelled by tags that are stems (e.g., `author` and `authors`), that are one substring of the other (e.g., `authors` and `co-authors`), or that

* Corresponding author.
E-mail addresses: Ismael.Sanz@uji.es (I. Sanz), mesiti@dico.unimi.it (M. Mesiti), guerrini@disi.unige.it (G. Guerrini), berlanga@uji.es (R. Berlanga).

0169-023X/$ - see front matter
doi:10.1016/j.datak.2007.05.008

are similar according to a given *thesaurus* (e.g., `author` and `writer`). The second level of heterogeneity is *structural heterogeneity* that results in documents with different hierarchical structures. Structural heterogeneity can be produced by the different schemas (i.e. DTDs or XML Schemas) behind the XML documents. Moreover, as schemas can also include optional, alternative, and complex components, structural heterogeneity can appear even for a single schema collection.

Highly heterogeneous XML collections can be found in a great variety of application domains, most of them related to the discovery and integration of semi-structured information coming from disparate and autonomous sources. Semantic Web Services is one of these application domains, which has been used for experimental evaluation. In this application, services are described according to concepts from one or several ontologies. In this way, service descriptions can be represented as trees expressing the hierarchical relationships that can appear in service methods and input/output parameters. Notice that these parameters usually have complex types (e.g., `address`) which can be defined in different ways across services. For example, in the ASSAM Web service collection, address and personal data present several structural relationships. In this context, user requests cannot account for all these structural variations as users do not a priori know how data are organized. Instead, the user expresses her request by arranging the elements of interest as she considers it is most likely to find them (pattern). In other words, she expresses an approximation of what she wishes to find, although the final results are not required to exactly comply to the given pattern.

A more complex and specific scenario is that of biomedical information exchange. Sharing clinical data among clinicians is crucial for the development of their researches and daily tasks, see e.g. [6]. Contrary to hospital information systems, where a great effort on data standardization has been produced (e.g. HL7 and IHE), clinical data managed within hospital departments has not been standardized at all because these data are quite irregular and dynamic. Clinical data cover many aspects of a patient: visits, examinations, diagnosis, symptoms, and a long etcetera. Also clinical procedures use to change over time as new findings over medical evidences are discovered (e.g., new treatments, new lab tests, etc.). Irregularity and dynamicity are two features of semi-structured data that are well supported by XML technology, therefore current biomedical projects like myGRID,[1] PEDRo[2] and caBIG[3] propose different XML-based models for clinical data exchange. Heterogeneity increases dramatically when several departments share their clinical records, since they can consider different detail levels and some clinical variables are more relevant than others. Also the structure of patient records can differ notably from one department to another depending on the particular use of the data. In this context, clinicians need a flexible retrieval tool for discovering interesting information from all the available data sources (e.g., local patient records, external patient records as well as public databases). For example, given a patient record, a clinician might wish to search not only similar patient records published within her department but also other document portions describing similar cases or reporting interesting findings with respect the patient at hand.

To sum up, in this paper we stress the tag and structural heterogeneity of XML document collections. This can lead to search a very large amount of highly heterogeneous documents. In this context, we propose an approach for identifying the portions of documents that are similar to a given *pattern*. In our context a pattern is a labelled tree whose labels are those that should be retrieved, preferably with the relationship imposed by the hierarchical structure, in the collection of documents (that for simplicity is named the *target* collection). We develop a two-phase approach where, in the first phase, tags occurring in the pattern are employed for identifying the portions of the target in which the nodes of the pattern appear. Exploiting the ancestor/descendant relationships existing among nodes in the target, subtrees (named *fragments*) are extracted having common/similar tags to those in the pattern, but eventually presenting different structures. The structural similarity between the pattern and the fragments is evaluated as a second step, for two purposes. First, for merging fragments in a *region* when the region exhibits a higher structural similarity with the pattern than the fragments it is originated from. Then, for ranking the identified fragments/regions and producing the result. In this second phase, different similarity measures can be considered, thus accounting for different

[1] http://www.mygrid.org.uk/.
[2] http://pedrodownload.man.ac.uk/main.html.
[3] https://cabig.nci.nih.gov/.

degrees of document heterogeneity, depending on the application domain and on the heterogeneity degree of the target collection.

The proposed approach is thus highly flexible. The problem is however how to perform the first step efficiently, that is, how to efficiently identify *fragments*, that is portions of the target containing labels similar to those of the pattern, without relying on strict structural constraints. Our approach employs ad-hoc data structures: a *similarity-based inverted index* (named *SII*) of the target and a *pattern index* extracted from *SII* on the basis of the pattern labels. Through *SII*, nodes in the target with labels similar to those of the pattern are identified and organized in the levels in which they appear in the target. Fragments are generated by considering the ancestor–descendant relationship among such vertices. Then, identified fragments are combined in *regions*, allowing for the occurrence of nodes with labels not appearing in the pattern, as described above. Finally, some heuristics are employed to avoid considering all the possible ways of merging fragments into regions and for the efficient computation of similarity, thus making our approach more efficient without loosing precision.

In the paper, we formally define the notions of fragments and regions and propose the algorithms allowing their identification, relying on our pattern index structure. The use of different structural similarity functions taking different structural constraints (e.g., ancestor–descendant and sibling order) into account is discussed. The practical applicability of the approach is finally demonstrated, both in terms of quality of the obtained results and in terms of space and time efficiency, through a comprehensive experimental evaluation. The contribution of the paper can thus be summarized as follows: (i) specification of an approach for the efficient identification of regions by specifically tailored indexing structures; (ii) characterization of different similarity measures between a pattern and regions in a collection of heterogeneous semi-structured data; (iii) realization of a prototype and experimental validation of the approach. The paper is an extended version of [7]. With respect to [7], the presentation of the approach has been deeply revised, all the underlying concepts are clearly and formally defined, and the corresponding algorithms, only sketched in [7], are detailed and their correctness and complexity are discussed. Moreover, the experimental evaluation considerably extends the preliminary results presented in [7].

The remainder of the paper is organized as follows. Section 2 formally introduces the notions of pattern, target, fragments, and regions the approach relies on. Section 3 is devoted to the discussion of different approaches to measure the similarity between the pattern and a region identified in the target. Section 4 discusses how to efficiently identify fragments and regions. Section 5 presents experimental results while Section 6 compares our approach with related work. Finally, Section 7 concludes the paper.

2. Pattern, target, fragment, and region trees

In our approach, both patterns and targets are represented as trees. Thus, fragments and regions are (sub)trees as well. In this section, we introduce these notions. First of all, however, we introduce some basic notions and some useful notations on trees together with some functions for element tag similarity.

2.1. Trees

The notation used throughout this paper to represent trees is fairly standard. A *tree* is a structure $T = (V, E)$, where V is a finite set of *vertices*, E is a binary relation on V that satisfies the following conditions: (i) the root (denoted root(T)) has no parent; (ii) every node of the tree except the root has exactly one parent; (iii) all nodes are reachable via edges from the root, that is, $(\text{root}(T), v) \in E^*$ for all nodes in V (E^* is the Kleene closure of E). If $(u, v) \in E$, we say that (u, v) is an *edge* and that u is the *parent* of v (denoted $\mathscr{P}(v)$). A *labeled tree* is a tree with which a node labeling function label is associated. Given a tree $T = (V, E)$, Table 1 reports functions and symbols used throughout the paper. In the following, when using the notations, the tree T will not be explicitly reported whenever it is clear from the context. Otherwise, it will be marked as subscript of the function.

Node order is determined by a pre-order traversal of the document tree [8]. In a pre-order traversal, a tree node v is visited and assigned its pre-order rank $pre(v)$ before its children are recursively traversed from left to right. A post-order traversal is the dual of the pre-order traversal: a node v is visited and assigned its post-order rank $post(v)$ after all its children have been traversed from right to left. The *level* of a node v in the tree is defined, as usual, by stating that the level of the root is 1, and that the level of any other node is

Table 1
Notations

Symbol	Meaning
root(T)	Root of T
$\mathscr{V}(T)$	Set of vertices of T (i.e. V)
$\|V\|$, $\|T\|$	Cardinality of $\mathscr{V}(T)$
label(v),label(V)	Label associated with a node v and the nodes in V
$\mathscr{P}(v)$	Parent of vertex v
$pre(v)$	Pre-order traversal rank of v
$post(v)$	Post-order traversal rank of v
$level(v)$	Level in which v appears in T
$level(T)$	Depth of T
$pos(v)$	Left-to-right position at which v appears among its siblings
desc(v)	Set of descendant of v (desc$(v) = \{u \mid (v,u)) \in E^*\}$)
$nca(v,u))$	Nearest common ancestor of v and u
$d(v)$	Distance of node v from the root
$d^{\max}$	Maximal distance from the root ($d^{\max} = \max_{v \in \mathscr{V}(T)} d(v)$)

the successor of the level of its parent. The *position* of a node v among its siblings is defined as the left-to-right position at which v appears among the nodes whose father is the father of v. Each node v is coupled with a quadruple $(pre(v), post(v), level(v), pos(v))$ as shown in Fig. 1a ($pre(v)$ is used as node identifier). For the sake of clarity, in the following graphics we omit the quadruples when they are not relevant for the discussion.

Pre-and post-ranking can also be used to efficiently characterize the descendants u of v. A node u is a descendant of v, $v \in desc(u)$, iff $pre(v) < pre(u) \wedge post(u) < post(v)$. There are some other useful properties that are trivially computed using these numbers. If, given two nodes, one is neither an ascendant nor a descendant of the other, then they are either left- or right-relatives. A node v is a *left-relative* of a node u if $pre(v) < pre(u)$ and $post(v) < post(u)$. The definition of *right-relative* is analogous. Given a tree $T = (V, E)$ and two nodes $u,v \in V$, the *nearest common ancestor* of u and v, $nca(u,v)$, is the common ancestor of u and v such that any other common ancestor of u and v is an ancestor of $nca(u,v)$. Note that $nca(u,v) = nca(v,u)$, and $nca(u,v) = v$ if u is a descendant of v. The distance $d(v)$ of a node v from the root, which coincides with its pre-order rank, amounts to the number of nodes traversed in moving from the root to the node in the pre-order traversal. The maximal distance $d^{\max}$ corresponds to the number of nodes in the tree, i.e. $|T|$. $d^{\max}$ also corresponds to the post-order rank of the root.

Example 1. Referring to the tree in Fig. 1a, node $(2,2,2,1)$ is a descendant of node (1,3,1,1), whereas it is a left-relative of node $(3,1,2,2)$. The distance of node $(3,1,2,2)$ from the root is 3, and $d^{\max}$ is 3 as well.

2.2. Label similarity

Labels can be similar or dissimilar depending on the adopted criteria of comparison specified by means of functions. In this paper, the following similarity functions have been considered, even if other ones can be easily integrated:

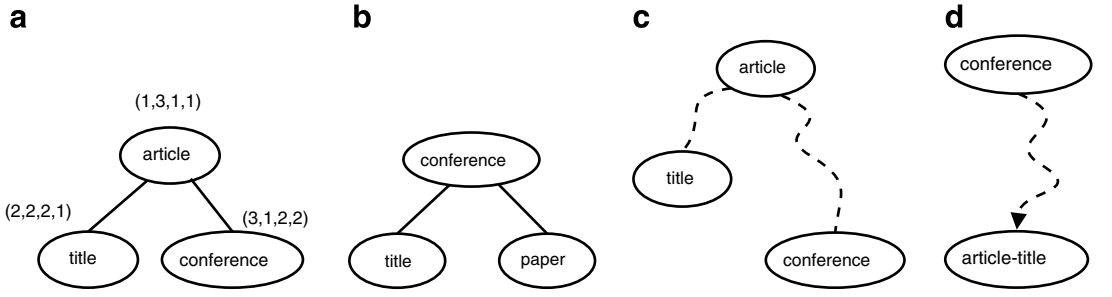


Fig. 1. (a) Pre/Post-order rank, a matching fragment with a different order (b), missing levels (c) and missing elements (d).

- *Case insensitive similarity function* S^t_{ci}. Two tags are similar if their differences depend only on the case (e.g., `author` and `Author` are similar).
- *Stemming function* S^t_{st}. Two tags are similar if one is a stem of the other (e.g., `author` and `authors` are similar).
- *Edit distance function* S^t_{ed}. Two tags are similar if their edit distance is less than a pre-fixed threshold (e.g., `author` and `auth` are similar).
- *Substring function* S^t_{ss}. Two tags are similar if the first one is contained in the second one (e.g., `author` and `conference-author` are similar).
- *Ontology-based function* S^t_{o}. Two tags are similar if they are synonym relying on a given thesaurus (e.g., `author` and `writer` are similar).

Relying on these similarity functions, the notions of similarity between two tags or between a tag and a set of tags are defined as follows:

Definition 1 (*Similarity and Similarly Belonging*). Let $\mathscr{S}$ be a set of label similarity functions. Let l_1, l_2 be two labels, l_1 is *similar* to l_2 according to $\mathscr{S}$ (denoted as $l_1 \simeq_{\mathscr{S}} l_2$) if and only if $l_1 = l_2$ or l_1 is similar to l_2 according to a function in $\mathscr{S}$. Let then l be a label and L be a set of labels, l *similarly belongs to* L according to $\mathscr{S}$ (denoted as $l \propto_{\mathscr{S}} L$) if and only if $\exists n \in L$ s.t. $l \simeq_{\mathscr{S}} n$.

In the following, when the subscript $\mathscr{S}$ is omitted we will implicitly refer to the whole set of similarity functions introduced above.

2.3. Pattern and target trees

A pattern is a labeled tree. The pattern is a tree representation of the user interest and can correspond to a collection of navigational expressions on the target tree (e.g., XPath or XQuery expressions in XML documents) or simply to a set of labels for which a ''preference'' is specified on the hierarchical or sibling order in which such labels should occur in the target. Labels in the pattern should be semantically distinct each other in order to avoid that two pattern labels can match the same element in the collection.

Example 2. Consider the pattern in Fig. 1a. Intuitively, the pattern expresses an interest in document portions related to articles, their titles, and the conferences where they are presented. Possible matches for this pattern are reported in Fig. 1b–d. The matching tree in Fig. 1b contains similar labels but at different positions, whereas the one in Fig. 1c contains similar labels but at different levels. Finally, the matching tree in Fig. 1d misses an element and the two elements appear at different levels.

The target is a set of heterogeneous documents in a source. The target is conveniently represented as a tree with a dummy root labeled `db` and whose subelements are the documents of the source. This representation relies on the common model adopted by native XML databases (e.g., eXist, Xindice [9]) and simplifies the adopted notations. An example of target is shown in Fig. 2. The dummy root has pre-order rank 0 and is at level 0 in the tree.

Definition 2 (*Target*). Let $\{T_1, \ldots, T_n\}$ be a collection of trees, where $T_i = (V_i, E_i)$, $1 \leqslant i \leqslant n$. A *target* is a tree $T = (V, E)$ such that:

- $V = \cup_{i=1}^{n} V_i \cup \{r\}$, and $r \notin \cup_{i=1}^{n} V_i$,
- $E = \cup_{i=1}^{n} E_i \cup \{(r, \mathrm{root}(T_i)), 1 \leqslant i \leqslant n\}$,
- $\mathrm{label}(r) = \mathtt{db}$.

2.4. Fragment and region trees

The basic building blocks of our approach are *fragments*. Given a pattern P and a target T, a fragment is a subtree of T, belonging to a single document of the target, in which only nodes with labels similar to those in P are considered. Two vertices u, v belong to a fragment for a pattern iff their labels as well as the label of

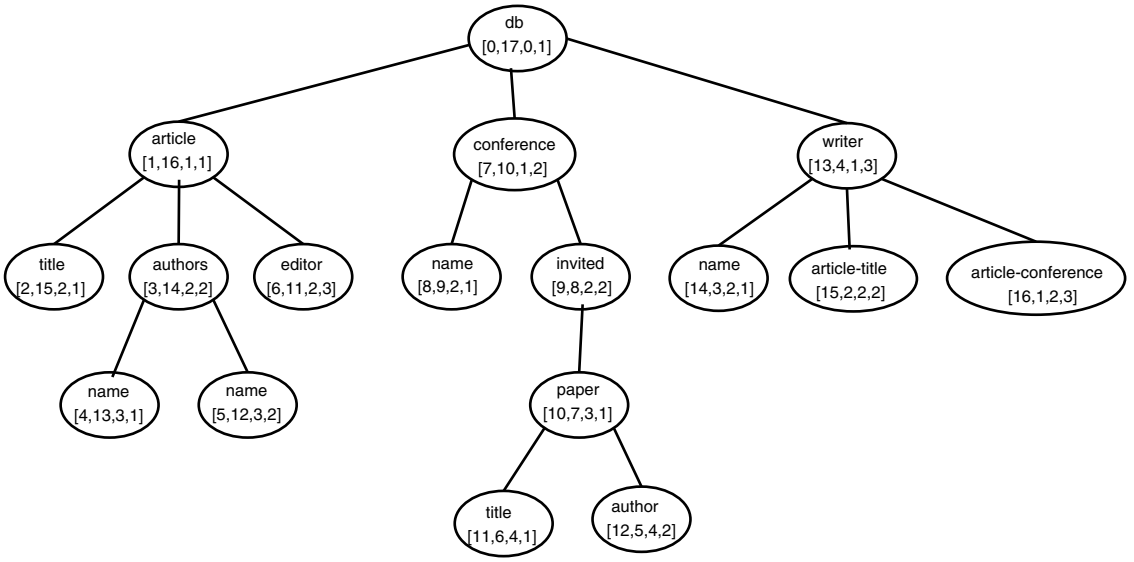


Fig. 2. A target.

their nearest common ancestor similarly belong to the labels in the pattern. Edges in the fragment either correspond to a direct edge in the target (father–children relationship) or to a path in the target (ancestor–descendant relationship). Several edges in the target, indeed, can be collapsed in a single edge in the fragment by "skipping" nodes that are not included in the fragment since their labels do not similarly belong to those in the pattern.

Definition 3 (*Fragment*). A *fragment* F of a target $T = (V_T, E_T)$ for a pattern P is a subtree (V_T, E_T) of T for which the following properties hold:

- V_F is the maximal subset of V_T such that $\text{root}(T) \notin V_F$ and $\forall u,v \in V_F$, $\text{label}(u)$, $\text{label}(v)$, and $\text{label}(nca(u,v)) \propto \text{label}(\mathscr{V}(P))$;
- for each $v \in V_F$, $nca(\text{root}(F),v) = \text{root}(F)$;
- $E_F = \{(u,v) | u,v \in V_F \wedge (u,v) \in E_T^* \wedge (\nexists w \in V_F, w \neq u, v \text{ s.t. } ((u,w) \in E_T^* \wedge (w,v) \in E_T^*))\}$.

Example 3. Consider the pattern in Fig. 1a and the target in Fig. 2. By considering all the label similarity functions introduced in Section 2.2, the corresponding four fragments are shown in Fig. 3a. Labels in the first fragment are exactly the same appearing in the pattern. By contrast, in the others require to exploit the substring and ontology-based functions. For instance, the second tree contains `paper` as label, and `paper` $\simeq_{S_o^t}$ `article`. Similarly, the third and fourth trees contain node labels that are similar, exploiting the substring function S_{ss}^t to labels `title` and `conference` in the pattern, respectively. The second tree provides an example of fragment in which a node of the original tree (i.e. node $(9,8,2,2)$ labeled by `invited`) is not included.

Starting from fragments, *regions* are introduced as a combination of fragments rooted at the nearest common ancestor in the target. Two fragments can be merged in a region only if they belong to the same document. In other words, the common root of the two fragments is not the `db` node of the target.

Example 4. Consider the tree T rooted at node n (13,4,1,3) in Fig. 2. It has two subtrees (the ones containing elements `article-title` and `article-conference`) that are fragments with respect to the pattern in Fig. 1a. Though n is not (part of) a fragment, the subtree consisting of n and its fragment subtrees could have a higher similarity with the pattern tree in Fig. 1a than its subtrees separately. Therefore, combining fragments into regions may lead to subtrees with higher similarities.

Fig. 3. (a) Fragments and (b) generated region.

A region can be a single fragment or it can be obtained by merging different fragments in a single subtree whose root is the nearest common ancestor of the fragments. Thus, while all fragment labels similarly belong to those in the pattern, a region can contain labels not similarly belonging to pattern labels.

Definition 4 (Regions). Let $F_P(T)$ be the set of fragments identified between a pattern P and a target T. The corresponding set of regions $R_P(T)$ is inductively defined as follows:

- $F_P(T) \subseteq R_P(T)$;
- For each $F = (V_T, E_T) \in F_P(T)$ and for each $R = (V_R, E_R) \in R_P(T)$ s.t. label(nca(root(F),root(R))) $\neq$ `db`, $S = (V_S, E_S) \in R_P(T)$; where:
  - root(S) = nca(root(F),root(R)),
  - $V_S = V_F \cup V_R \cup \{\text{root}(S)\}$,
  - $E_S = E_F \cup E_R \cup \{(\text{root}(S),\text{root}(F)),(\text{root}(S),\text{root}(R))\}$.

Example 5. The fragments in Fig. 3a are also regions as well as the region reported in Fig. 3b obtained by merging the third and fourth fragments in Fig. 3a. Note that the region root label (i.e. `writer`) does not similarly belong to any label in the pattern.

Relying on this definition, regions are all possible combinations of fragments in a document of the target. This number can be exponential in the number of fragments. In Section 4.3 the *locality principle* will be discussed to reduce the number of regions to consider.

The notions of level of a node and distance between two nodes, when applied to regions, refer to the corresponding notions in the original target tree. More specifically, they refer to the target subtree whose root is the region root and whose leaves are the region leaves. We will refer to this tree as the target subtree *covered* by the region. This subtree, as already discussed, may contain additional internal nodes that are not included in the region since their labels do not appear in the pattern. Specifically, internal nodes are included in the covered subtree if they are either in the path from the region root to some region node or if they are internal sibling of two nodes in the covered tree (i.e. right-sibling of one of them and left-sibling of the other).

Definition 5 (*Covered Subtree*). Let T be a target and R be a region on it. The subtree of T *covered* by R, denoted as $C(R)$, is the subtree of T such as:

- $V_{C(R)} \subseteq V_T$ is inductively defined as follows:
  - $V_R \subseteq V_{C(R)}$;
  - $\forall v \in V_T$ such that $\exists u,w \in V_{C(R)}$ and (u,v), $(v,w) \in V_T$, $v \in V_{C(R)}$;
  - $\forall v \in V_T$ such that $\exists u,w,f \in V_{C(R)}$ and $(f,u),(f,v),(f,w) \in V_T$, u is a left-sibling of v, and w is a right-sibling of v, $v \in V_{C(R)}$;
- $E_{C(R)} = \{(u,v) | (u,v) \in E_T \text{ s.t. } u,v \in V_{C(R)}\}$.

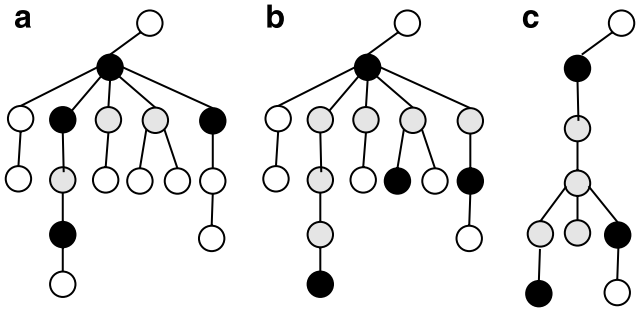


Fig. 4. Covered subtrees in a target.

Example 6. Consider region R_2 of Fig. 5. The depth of the region $level(R_2) = 4$ though the region includes only three nodes. Similarly, the level of the `paper` labeled node is 3. The distance of the `paper` labeled node is 4, which is also the maximal distance in the region.

Fig. 4 presents different parts of a target where black nodes identify region nodes and gray nodes together with black nodes form the covered subtree.

3. Similarity of a region w.r.t. a pattern

In this section we present the foundation of our two-phase approach to identify target regions similar to a pattern. We first identify the possible matches between the vertices in the pattern and those in the region having similar labels, without exploiting the hierarchical structure of the tree. Then, the hierarchical structure is taken into account to select, among the possible matches, those that are structurally more similar. Specifically, after having introduced the definition of mapping, we propose three different similarity measures that combine structure and tag matching.

3.1. Mapping between a pattern and a region

A mapping between a pattern and a region is a relationship among their elements that takes the tags used in the documents into account. Our definition differs from the definition of mapping proposed by other authors [10,11]. Since our focus is on heterogeneous semi-structured data, we do not take the hierarchical organization

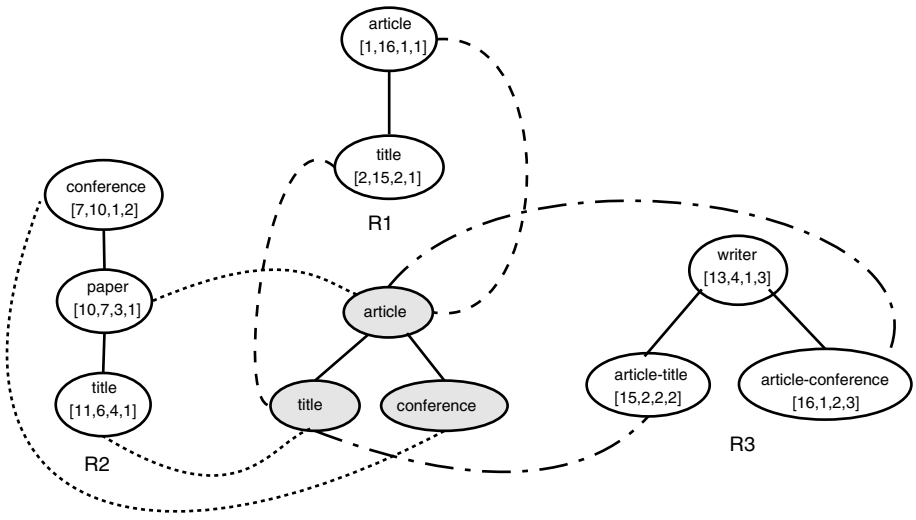


Fig. 5. Mapping between the pattern and different regions.

of the pattern and the region into account in the definition of the mapping. We only require that the element labels are similar.

Definition 6 (*Mapping M*). Let P be a pattern and R be a region. A mapping M is a partial injective function between the vertices of P and those of R such that $\forall x_p \in \mathscr{V}(P), M(x_p) \neq \perp \Rightarrow \text{label}(x_p) \simeq \text{label}(M(x_p))$.

Example 7. Fig. 5 reports the pattern P of Fig. 2 in the center and three target regions, R_1,R_2, R_3. Dashed lines represent a mapping among the vertices of the pattern and those of each region.

Several mappings can be established between a pattern and a region. The best one will be selected by means of a similarity measure that evaluates the degree of similarity between the two structures relying on the degree of similarity of their matching vertices.

Example 8. Fig. 6 reports how different mappings can be established between the region R in Fig. 3b and the pattern P in Fig. 1a.

Given a similarity measure, like the ones discussed in the next section, assessing the similarity between vertices, a mapping can be evaluated as follows.

Definition 7 (*Evaluation of a Mapping M*). Let M be a mapping between a pattern P and a region R, and let $\mathscr{S}im$ be a vertex similarity function. The evaluation of M is:

$$\mathscr{E}val(M) = \frac{\sum_{x_p \in \mathscr{V}(P) \text{s.t.} M(x_p) \neq \perp} \mathscr{S}im(x_p, M(x_p))}{|\mathscr{V}(P)|}$$

Similarity between a pattern and a region is then defined as the maximal evaluation among the mappings that can be determined between the pattern and the region.

Definition 8 (*Similarity between a Pattern and a Region*). Let $\mathscr{M}$ be the set of mappings between a pattern P and a region R. The similarity between R and P is defined as:

$$\mathscr{S}im(P,R) = \max_{M \in \mathscr{M}} \mathscr{E}val(M)$$

3.2. Similarity between matching vertices

We now present three approaches for computing the similarity between a pair of matching vertices. The first one assesses similarity only on the bases of label matches, whereas the other two take the structure into account.

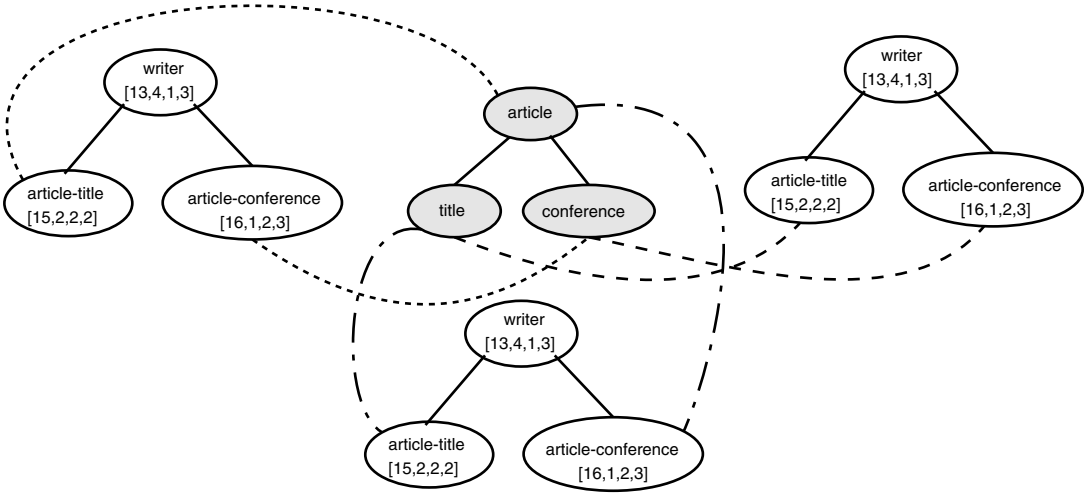


Fig. 6. Identification of different mappings from the same region and pattern.

3.2.1. Match-based similarity

In the first approach, similarity only depends on node labels. Similarity is 1 if labels are identical, whereas a pre-fixed penalty δ is applied if labels are similar. If they are not similar, similarity is 0.

Definition 9 (*Match-based Similarity*). Let P be a pattern, R be a region in a target T, x_p a node of P, and $x_r = M(x_p)$. Their similarity is computed as:

$$\mathscr{S}im_M(x_p, x_r) = \begin{cases} 1 & \text{if } \text{label}(x_p) = \text{label}(x_r) \\ 1-\delta & \text{if } \text{label}(x_p) \simeq \text{label}(x_r) \\ 0 & \text{otherwise} \end{cases}$$

Example 9. Let x_p be the vertex tagged `article` in the pattern P in Fig. 5 and x_r^1, x_r^2, x_r^3 the corresponding vertices in the regions R_1, R_2, R_3. Table 2a reports, in the first column, the match-based similarity between x_p and the corresponding vertices in the three regions.Table 2b reports, in the first column, the match-based evaluations of the mappings between P and each of the regions (see Fig. 5), computed according to Definition 7. For regions R_1 and R_2, there is a single mapping (M_1 and M_2, respectively). For region R_3, by contrast, the three mappings M_3^l, M_3^r, and M_3^b, appearing in the left, right, and bottom of Fig. 6, are evaluated. The table also highlights in bold the similarity of P with each of the regions, computed according to Definition 8. The evaluation details are in Appendix A.

3.2.2. Level-based similarity

In the second approach, the match-based similarity is combined with the evaluation of the level at which x_p and $M(x_p)$ appear in the pattern and in the region. Whenever they appear in the same level, their similarity is equal to the similarity computed by the first approach. Otherwise, their similarity linearly decreases as the number of levels of difference increases. We recall that levels in the region refer to the levels in the target subtree *covered* by the region.

Definition 10 (*Level-based similarity*). Let P be a pattern, R be a region in a target T, x_p a node of P, and $x_r = M(x_p)$. Their similarity is computed as:

$$\mathscr{S}im_L(x_p, x_r) = \mathscr{S}im_M(x_p, x_r) - \frac{|level_P(x_p) - level_R(x_r)|}{max(level(P), level(R))}$$

The similarity is 0 if the obtained value is below 0.

Example 10. The second columns of Tables 2a and 2b report the level-based similarities. The evaluation details are in Appendix A.

Table 2a
Similarity of matching vertices

	$\mathscr{S}im_M$	$\mathscr{S}im_L$	$\mathscr{S}im_D$
x_r^1	1	1	1
x_r^2	$1-\delta$	$\frac{1}{2}-\delta$	$\frac{1}{2}-\delta$
x_r^3	$1-\delta$	$\frac{1}{2}-\delta$	$\frac{1}{3}-\delta$

Table 2b
Evaluation of mappings and similarity of a pattern with regions

	$\mathscr{S}im_M$	$\mathscr{S}im_L$	$\mathscr{S}im_D$
$\mathbf{M_1}$	$\frac{2}{3}$	$\frac{2}{3}$	$\frac{2}{3}$
$\mathbf{M_2}$	$1-\frac{\delta}{3}$	$\frac{7}{12}-\frac{\delta}{3}$	$\frac{1}{2}-\frac{\delta}{3}$
M_3^b	$\frac{2}{3}\cdot(1-\delta)$	$\frac{1}{2}-\frac{2}{3}\cdot\delta$	$\frac{1}{2}-\frac{2}{3}\cdot\delta$
M_3^l	$\frac{2}{3}\cdot(1-\delta)$	$\frac{1}{2}-\frac{2}{3}\cdot\delta$	$\frac{5}{9}-\frac{2}{3}\cdot\delta$
$\mathbf{M_3^r}$	$\frac{2}{3}\cdot(1-\delta)$	$\frac{2}{3}\cdot(1-\delta)$	$\frac{2}{3}\cdot(1-\delta)$

$$d_R(v_1) = 1$$

$$d_R(v_i) = d_R(v_{i-1}) + \begin{cases} level(v_i) - level(v_{i-1}) & v_i \in desc(v_{i-1}) \\ pos(v_i) - pos(v_{i-1}) & v_i \text{ sibling of } v_{i-1} \\ pos(av_i) - pos(av_{i-1}) + & \\ \qquad level(v_i) - level(av_i) & \text{otherwise} \end{cases}$$

Fig. 7. Distance of a vertex in a region R.

3.2.3. Distance-based similarity

Since two nodes can be in the same level, but not in the same position, a third approach is introduced. The similarity is computed by taking the distance of nodes x_p and $M(x_p)$ with respect to their roots into account. Thus, in this case, the similarity is the highest only when the two nodes are in the same position in the pattern and in the region. We recall that distances in the region refer to the distances in the target subtree *covered* by the region computed through the recursive function d_R in Fig. 7. Given a region R, $v_1, \ldots, v_n$ are the vertices of R ordered according to their pre-order rank in the target T. In the figure we report the computation of the distance for the root of R (v_1) and for a generic vertex of R that is not the root. Note that, $d_R^{\max} = d_R(v_n)$. Given two adjacent vertices v_{i-1} and v_i that are not siblings, av_{i-1} and av_i denote their common sibling ancestors. We remark that the last expression for computing the distance of a generic v_i in R includes the two previous cases in the definition of d_R. However, for the sake of clarity we have pointed out these two particular cases.

Definition 11 (*Distance-based Similarity*). Let P be a pattern, R be a region in a target T, x_p a node of P, and $x_r = M(x_p)$. The similarity is computed as:

$$\mathscr{S}im_D(x_p, x_r) = \mathscr{S}im_M(x_p, x_r) - \frac{|d_P(x_p) - d_R(x_r)|}{\max(d_P^{\max}, d_R^{\max})}$$

The similarity is 0 if the obtained value is below 0.

Example 11. The third columns of Tables 2a and 2b report the distance-based similarities. The evaluation details are in Appendix A.

4. Construction of fragments and regions

The focus of this section is on the data structures and algorithms for the efficient identification of fragments and regions in the target. As specified in Definition 3, each fragment is a set of nodes bound by the ancestor–descendant relationship in the target. A general purpose indexing structure along with an indexing structure depending on the pattern P are employed for improving the performances of our approach. Fragments are merged into regions, as specified in Definition 4, only when the similarity between P and the generated region is greater than the similarity between P and each single fragment. Target subtrees covered by a region should be evaluated only accessing nodes in the regions and information contained in the auxiliary indexing structures.

In the remainder of the section, we first present the indexing structures. Then, we discuss the algorithms for the construction of a list of fragments and for the creation of regions starting from such a list.

4.1. Similarity-based inverted index and pattern index

Directly evaluating the pattern on the target is inefficient and introduce scalability issues in the approach, due to the tag and structural heterogeneity of the collection. For these reasons, a *similarity-based inverted index (SII)* is proposed. This index is independent from the retrieval pattern and is composed by a traditional

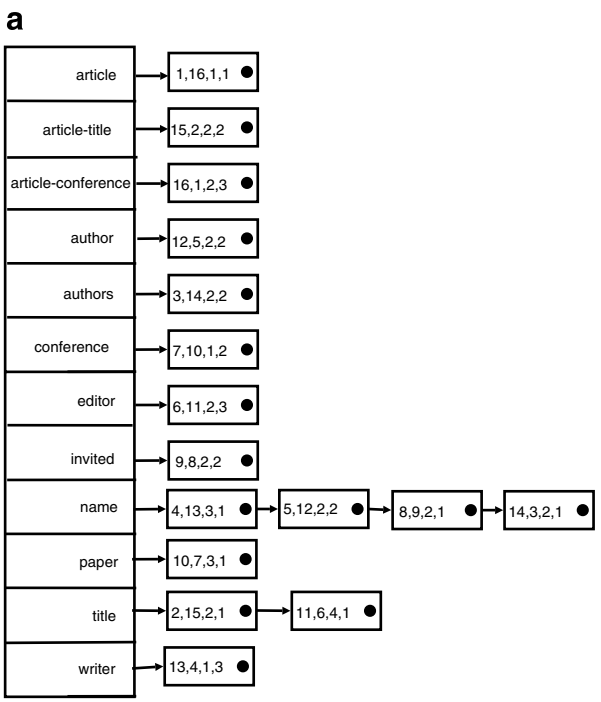
inverted index coupled with a table, *name-similarity table*, that specifies relationships among tags in the collection relying on the semantic functions discussed in Section 2.

This index allows us to easily identify in the collection nodes with similar tags according to different criteria. Since the number of labels that normally occur in a target is sensibly smaller than the number of elements, the size of the name-similarity table is often contained and it can also fit in main memory.

The SII index is built as follows. Starting from the labels of a target T, a traditional inverted index is created, that is, each distinct label l occurring in the target is associated with the list of vertices labeled by l, ordered according to the pre-order rank. The 5-tuple $(pre(v), post(v), level(v), pos(v), \mathscr{P}(v))$ is maintained for each vertex $v \in \mathscr{V}(T)$. Then, tags in the collection are grouped according to each of the tag-based similarity functions presented in Section 2 and each group is progressively numbered. Each tag is finally associated with the list of the group identifiers it belongs to. Fig. 8b reports this association by means of a table. For example, `author` and `authors` belong to the same class according to function S^t_{ed}, whereas `author`, `authors`, and `writer` belong to the same class according to function S^t_o. In this way, when nodes tagged l should be retrieved in the target, the rows in the name-similarity table corresponding to l are extracted, and then, according to one or more similarity measures, all the similar tags are easily identified. In the case l does not belong to the name-similarity table, one of the tag-based similarity functions can be applied to identify a similar tag in the table.

Given a pattern P, for every node v in P, all the occurrences of nodes u in the target tree such that $\text{label}(v) \simeq_{\mathscr{S}} \text{label}(u)$ are retrieved from the SII index according to the functions in $\mathscr{S}$, and organized level by level in a *pattern index*. The pattern index therefore depends on the pattern that should be evaluated on the target. The number of levels in the index depends on the levels in T at which vertices occur with labels similar to those in the pattern. For each level, vertices are ordered according to the pre-order rank.

Depending on the label similarity functions in $\mathscr{S}$, different pattern indexes can be generated. Fig. 9a shows the pattern index obtained by identifying nodes identical to those of the pattern, Fig. 9b reports the pattern index obtained by applying the ontology-based function S^t_o, and, finally, Fig. 9c the pattern index in which all the criteria have been exploited. All the pattern indexes are obtained starting from the pattern in Fig. 1a evaluated on the SII index in Fig. 8 corresponding to the target in Fig. 2.



b

tag	S^t_{st}	S^t_{ci}	S^t_{ed}	S^t_{ss}	S^t_o
article				1	1
article-title				1	
article-title				2	
article-conference				1	
article-conference				3	
author	1		1		2
authors	1		1		2
conference				3	
editor					
invited					
name					
paper					1
title				2	
writer					2

Fig. 8. SII index components: (a) inverted index and (b) name-similarity table.

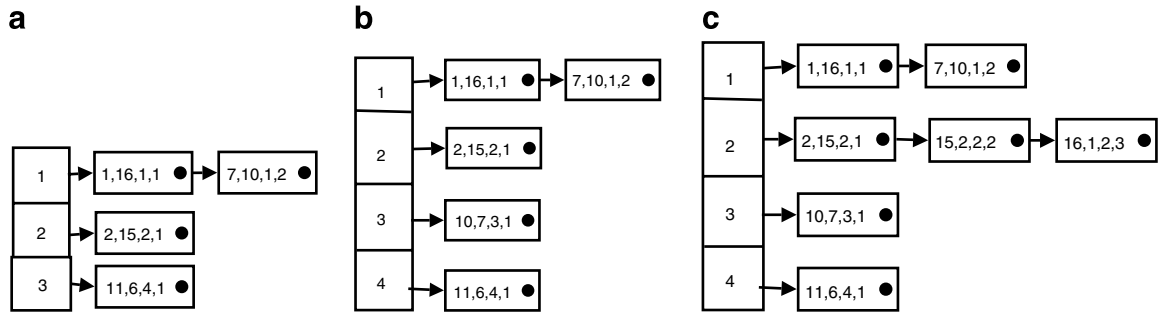


Fig. 9. Pattern index (a) with label equality, (b) applying ontology-based function, and (c) applying all the criteria.

The construction of the pattern index requires to identify the entries in *SII* with similar tags to those of *P*. For each node of *P*, this operation requires (in the worst case) to check the distinct labels that tag each entry of *SII*. Once the entries are identified, the corresponding nodes are inserted in the pattern index. Since the nodes in each entry are ordered according to the pre-order rank in the target, the construction of the pattern index is performed in linear time. Therefore, the number of operations for each tag in *P* is in the order of $\mathcal{O}(K \cdot M)$ where *K* is the number of entries and *M* the maximal number of nodes in each entry. The number of operations for the construction of the entire pattern index for a pattern *P* is thus $\mathcal{O}(|P| \cdot K \cdot M)$. We remark that the use of the name-similarity table does not affect the complexity of the approach in the worst case analysis but only in the average case.

Algorithm 1: Create Fragments

Require: *PI*,*F*,*v*,*l*

{PI: pattern index
F: current fragment
v: vertex in current fragment
l: a level in the pattern index}
$SF = \emptyset$
if $l \leqslant size(PI) \wedge head(PI(l))$ is not null **then**
  while $head(PI(l))$ is not null and precedes root(F) **do**
    Create a new fragment $F' = (V'_F, \emptyset)$ such that $V'_F = \{head(PI(l))\}$
    $SF = SF \cup \{F'\} \cup$ createFragments(PI,F',$head(PI(l))$,$l+1$)
    remove $head(PI(l))$
  end while
  while $head(PI(l))$ is not null and is a descendant of (root(F)) **do**
    Insert $head(PI(l))$ in the vertex of F
    if $head(PI(l))$ precedes v **then**
      Insert (root(F), $head(PI(l))$) in the edges of F
    else
      Insert (v, $head(PI(l))$) in the edges of F
    end if
    $SF = SF \cup$ createFragments(PI, F, $head(PI(l))$, $l+1$)
    remove $head(PI(l))$
  end while
  $SF = SF \cup$ createFragments(PI, F, v, $l+1$)
end if

Ensure: return *SF*

4.2. Algorithms for the construction of fragments

Once the pattern index is generated, the fragments are generated through the recursive Algorithm 1: *CreateFragments* and Algorithm 2: *CreateListOfFragments*. The main advantage of these algorithms is the iden-

tification of the fragments through a single visit of the pattern index so that the complexity of fragments construction is kept linear.

To simplify the presentation of the algorithm we assume that, once a node in the pattern index is visited and inserted in a fragment, it is removed from the pattern index. In the algorithms, given a pattern index PI, $size(PI)$ denotes the number of levels, $PI(l)$ denotes the list of nodes at level l ($1 \leqslant l \leqslant size(PI)$), and $head(PI(l))$ denotes the first node in the list $PI(l)$.

Algorithm *CreateFragments* is invoked relying on the level and the pre-order rank of the roots of the fragments to be generated. Given a fragment F, *CreateFragments* identifies all the nodes of F appearing in the pattern index and generates a tree on such nodes according to Definition 3. Moreover, the recursive calls of this function also return the fragments whose roots appear in a lower level and precede the root of F.

Relying on the assumption that, when a fragment is generated it is removed from PI, the PI on which *CreateFragments* is invoked for a fragment F does not contain all the nodes belonging to fragments whose roots are at a level k such that $k < level(\text{root}(F))$. After the invocation of *CreateFragments* on F, PI will not contain all the nodes of F along with the nodes of the fragments whose roots precede root(F) at a level h such that $(h > l)$.

Algorithm *CreateFragments* takes as input the pattern index PI, the current fragment F, a node v in F (initially the root of F, then an internal node for which we are looking for direct descendants), and the level l in PI where we are looking for descendants of F. Its behavior relies on the following proposition.

Proposition 1. *Let CreateFragments be invoked for a fragment F on a level l of a pattern index PI and a leaf node v of F located at a level m in PI($m < l$). The following properties hold*:

(1) If $head(PI(l))$ is a left relative of root(F), $head(PI(l))$ is the root of a new fragment.
(2) If $head(PI(l))$ is a descendant of root(F), but not of v, $head(PI(l))$ belongs to F and is a descendant of root(F).
(3) If $head(PI(l))$ is a descendant of v in F, $head(PI(l))$ belongs to F and is a descendant of v.
(4) If $head(PI(l))$ is a right relative of root(F), no elements of F can be identified at this level.

The cases described by Proposition 1 can occur in each invocation of Algorithm *CreateFragments* and handled as follows. We refer to Fig. 10 for a better understanding of the behavior of the algorithm. If $head(PI(l)) = t$ is left relative of root(F) (case (1) of Proposition 1), t is the root of another fragment. Indeed, all the fragments F^o such that root(F^o) precedes root(F) at a level k ($k < l$) could contain t have been removed from PI. Therefore, a new fragment F' is created and Algorithm *CreateFragments* is invoked for this fragment and its root at level $l + 1$. If $head(PI(l)) = u$ belongs to the descendants of root(F), but it is not a descendant of v (case (2) of Proposition 1), u is left relative of all internal nodes of F. Therefore, u is inserted as child of root(F) and Algorithm *CreateFragments* is invoked on F and u to identify possible descendants of u in F at

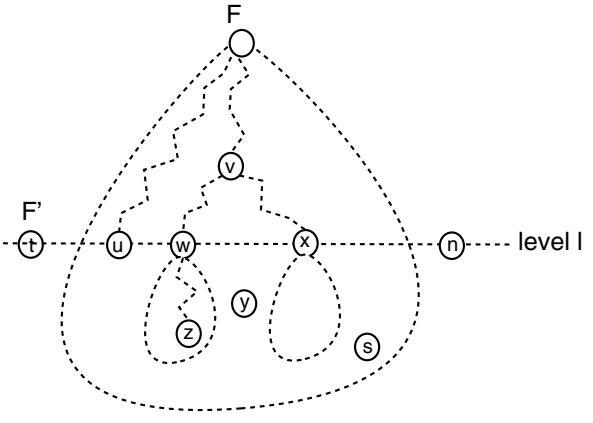


Fig. 10. Situations that can arise in the creation of a fragment.

level $l+1$. If $head(PI(l)) = w$ (the behavior is the same for node x) belongs to the descendants of v (case (3) of Proposition 1), w is associated as a child of v and Algorithm *CreateFragments* is invoked on F and w to identify possible descendants of w in F at level l+1 (as shown in Fig. 10 node z will be identified). If $head(PI(l)) = n$ is right relative of F (case (4) of Proposition 1), nor n nor the nodes that could follow n at level l can belong to F. Algorithm *CreateFragments* is thus invoked recursively on F, the same v and level $l+1$ in order to identify further descendants of v that do not stay at level l. Through this call, nodes y and s will be identified if v is their immediate ancestor in F.

Starting from the root of a fragment F, Algorithm *CreateFragments* is recursively invoked on the levels of the pattern index between the level of root(F) and $size(PI)$. Therefore, we are guaranteed that if a level contains a node of F, it is detected and inserted in the correct position in the hierarchical structure of F.

Algorithm *CreateListOfFragments* generates fragments starting from the first level of PI and moving downwards to its last level. In the first level, all the nodes are roots of fragments. Therefore, Algorithm *CreateFragments* is invoked on each of the fragments of the first level. Once all the fragments of the first level have been identified, the algorithm proceeds with the nodes of PI still belonging to the remaining levels (if they are not empty). Algorithm *CreateListOfFragments* ends when it reaches the last level of PI and all the nodes have been removed from PI.

Algorithm 2: Create List Of Fragments

Require: PI
  {$PI =$ pattern index}
  $SF = \emptyset$
  for $l = 1$ **to** $size(PI)$
    while $head(PI(l))$ is not null
      Create a new fragment $F = (V_F, \emptyset)$ such that $V_F = \{head(PI(l))\}$
      $SF = SF \cup \{F\} \cup$ createFragments(PI,F,$head(PI(l))$,$l+1$)
      remove $head(PI(l))$
    end while
  end for
Ensure return SF

Proposition 2. *Let PI be the pattern index for a pattern P. Algorithm* 2: *CreateListOfFragments correctly identifies all fragments in PI.*

All the atomic operations in the algorithm (checking whether a node precedes another one, adding a node to a fragment, removing a node from PI) require a constant number of operations. The algorithm visits each vertex in the pattern index only once by removing in each level the vertices already included in a fragment. Therefore, its complexity is in $\mathcal{O}(N)$, where N is the number of nodes in PI.

Fig. 11 contains fragments $F_1, \ldots, F_4$ obtained from the PI of Fig. 9c.

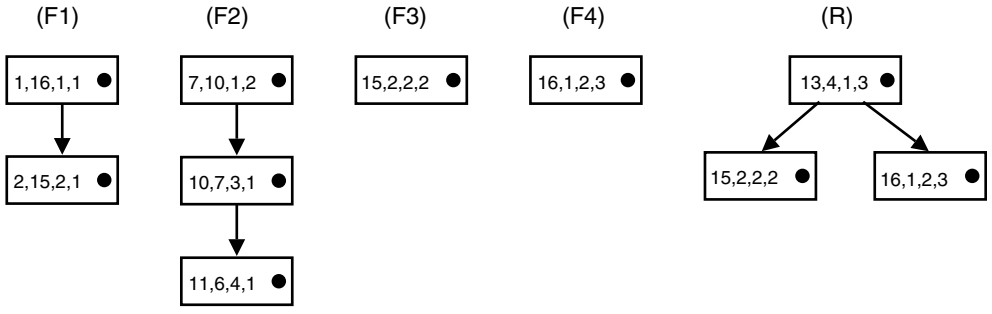


Fig. 11. Construction of fragments and regions.

4.3. Algorithm for the construction of regions

Two fragments should be merged in a single region when, relying on the adopted similarity function, the similarity of the pattern with the region is higher than the similarity with the individual fragments.

Whenever a document in the target is quite big and the number of fragments is high, the regions that should be checked can grow exponentially. To avoid such a situation we exploit the following *locality principle*: merging fragments together or merging fragments to regions makes sense only when the fragments/regions are close. Indeed, as the size of a region tends to be equal to the size of the document, the similarity decreases.

In order to meet such locality principle regions are obtained by merging adjacent fragments. Operatively, two adjacent fragments can be merged when their common ancestor v is not the root of the target. If it is not the root of the target, their common ancestor v becomes the root of the region and the roots of the fragments become the direct children of v. We remark that the common ancestor of two fragments can be easily obtained by traversing the $\mathscr{P}$-*arent* link included in the nodes of the pattern index.

Combining the locality principle and the approach for merging together two adjacent fragments, Algorithm 3: *CreateListOfRegions* is obtained. The algorithm works on the list of fragments SF obtained from Algorithm *CreateListOfFragments* that is ordered according to the pre-order rank of the roots of the fragments it contains. Once a possible region R_i is obtained, by merging two adjacent fragments $SF(i-1)$ and $SF(i)$, the similarity $\mathscr{S}im(P,R_i)$ is compared with the maximal value between $\mathscr{S}im(P,SF(i-1))$ and $\mathscr{S}im(P,SF(i))$. If $\mathscr{S}im(P,R_i)$ is the highest, $SF(i-1)$ is removed from the list and $SF(i)$ substituted with R_i. Otherwise, $SF(i-1)$ is kept alone and we try to merge $SF(i)$ with its right adjacent fragment. The process ends when all the fragments in the list have been checked.

Algorithm 3: Create List Of Regions

Require SF, P
{SF: list of fragments
P: Pattern}
for $i=2$ **to** $size(SF)$ **do**
  Try to generate region R_i by merging $SF(i-1)$ and $SF(i)$
  if R_i has been generated **then**
    if $\mathscr{S}im(P,R_i) \geqslant max\{\mathscr{S}im(P,SF(i-1)),\mathscr{S}im(P,SF(i))\}$ **then**
      Remove $SF(i-1)$
      Substitute $SF(i)$ with R_i
    end if
  end if
end for
Ensure return SF

Example 12. Considering the running example we try to generate regions starting from the fragments in Fig. 11. Since the common ancestor between F_1 and F_2 is the root of the target, the two fragments cannot be merged. Same behavior for fragments F_2 and F_3. Since the common ancestor between F_3 and F_4 is a node in the same document, region R in Fig. 11 is generated. Since the similarity of P with R is higher than its similarity with F_3 and F_4, R is kept and F_3,F_4 removed. At the end of the process we have regions $\{F_1,F_2,R\}$.

A key point of our approach is the efficient computation of the similarity between a pattern P and a fragment F or a region R. An array indexed on the labels of P is employed to keep the highest evaluation of similarity between a node in P and all the nodes in F/R with a similar label. The evaluation in the array are finally added to obtain the similarity between P and F/R. The number of operations is $\mathscr{O}(|P|)$. Since the similarity between P and F can be computed during the construction of a fragment, the complexity for evaluating the similarity between P and F in this phase is constant. Things are different for the similarity between P and R, where R is the combination of two fragments. In this case, the evaluations of two arrays should be compared and thus the number of operations is $\mathscr{O}(|P|)$.

For the construction of a region a pair of fragments F_1 and F_2 must be considered. The arrays, associated with F_1 and F_2 containing the highest similarity between P and them, must be visited for obtaining the evaluation of a region. Since the arrays have $|P|$ entries and each fragment is considered only once, the number of comparisons is proportional to the size of the list of fragments. The number of operation is $\mathcal{O}(|P| \cdot size(SF))$. Since in the worst case, each fragment contains a single node of the pattern index PI, $size(SF) = size(PI)$. Therefore, the creation of regions from SF requires $\mathcal{O}(N \cdot |P|)$ operations, where N is the number of nodes in PI.

We wish to remark that the construction of regions is quite fast because the target should not be explicitly accessed. All the required information are contained in the inverted indexes. Moreover, thanks to our *locality principle* the number of regions to check is proportional to the number of fragments. Finally, the regions obtained through our process do not present all the vertices occurring in the target but only those necessary for the computation of similarity. Vertices appearing in the region but not in the pattern are evaluated through the pre/post-order rank of each node.

Example 13. Consider the region R_2 in Fig. 5 and the corresponding representation F_2 in Fig. 11. Vertex `invited` is not explicitly present in R. However, its lack can be taken into account by considering the levels of vertex `conference` and vertex `paper`.

The following proposition summarizes the complexity of the algorithm for the creations of regions starting from a pattern.

Proposition 3. *Let P be a pattern, K the number of distinct label in the SII index, and M the maximal size of an entry in SII. Moreover, let N be the number of nodes in the pattern index PI. The number of operations for the creation of regions starting from a pattern is $\mathcal{O}(max\{(|P| \cdot K \cdot M), (N \cdot |P|)\})$.*

5. Prototype and experimental results

We have implemented a prototype using a Oracle Berkeley DB in the back-end, which is being used for the exploration of heterogenous sources in biomedical information integration projects. An example of this functionality is shown in two screenshots: Fig. 12 shows the combined results for a query and Fig. 13 shows the top results found in different collections. The figure highlights the richness and heterogeneity of the retrieved structures. This allows the data engineer to study the available collections as a whole, progressively refining the targets and the measures. Note that the system allows tailoring the similarity measure on-the-fly; this case uses a domain-specific label-matching function (see Section 2.2).

We have used this system to experimentally evaluate, both on real and synthetic data, the following aspects of our approach:

(1) Reasonable performance with large collections.
(2) Handling of severe structural distortions.
(3) Retrieval effectiveness in highly heterogeneous collections.

In the remainder of the section we report our results along these aspects.

5.1. Performance

The performance of the system has been tested using two large synthetic datasets, with associated test patterns. Dataset 1 is designed to contain just a few matching results, embedded in a large number of *don't-care* nodes (around 7500 relevant elements out of 10^7 elements). In contrast, dataset 2 has a high proportion of relevant elements (3×10^5 out of 5×10^5). In order to obtain results for a range of dataset sizes, smaller collections have been obtained by sampling each dataset. The characteristics of all datasets are summarized in Fig. 14a and b. Results in Fig. 14c and d show that the retrieval performance is linearly dependent on the size of the result set.

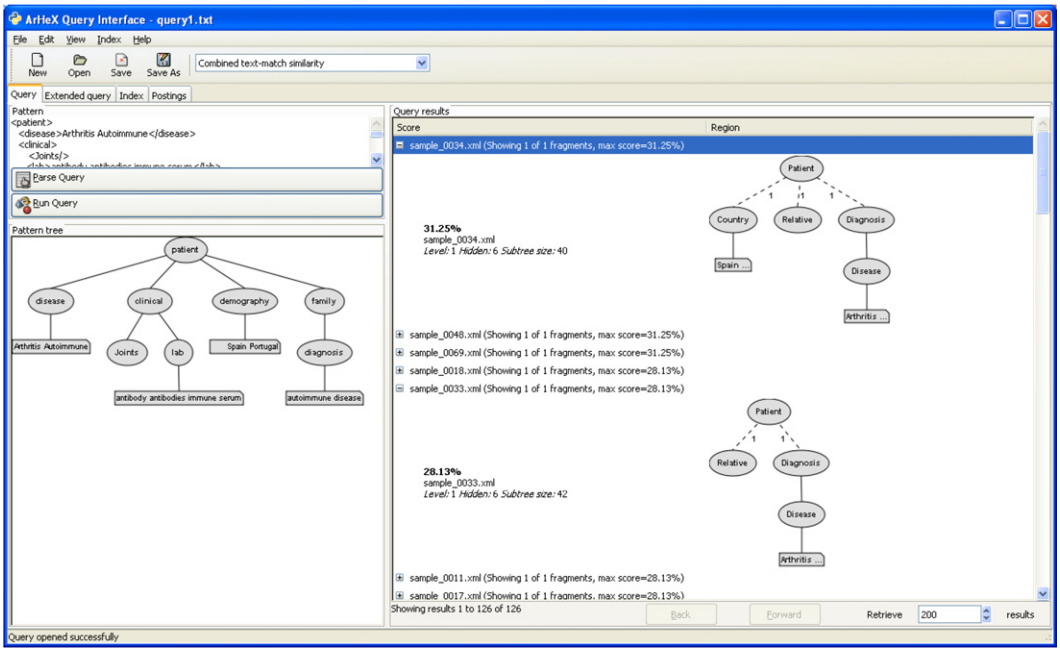


Fig. 12. Screenshot of the GUI prototype showing the top results in some collections, using a variation of the match-based similarity and a flexible label-matching function.

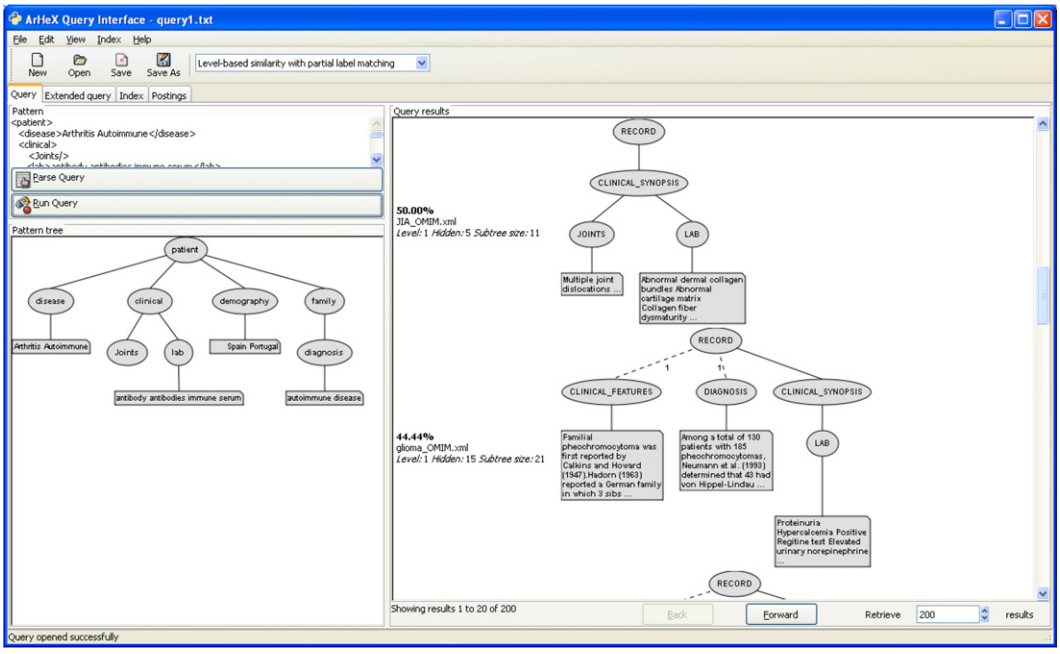


Fig. 13. Screenshot of the GUI prototype showing some results of a query over a set of heterogenous biomedical-related collections, using level-based similarity and a flexible label-matching function.

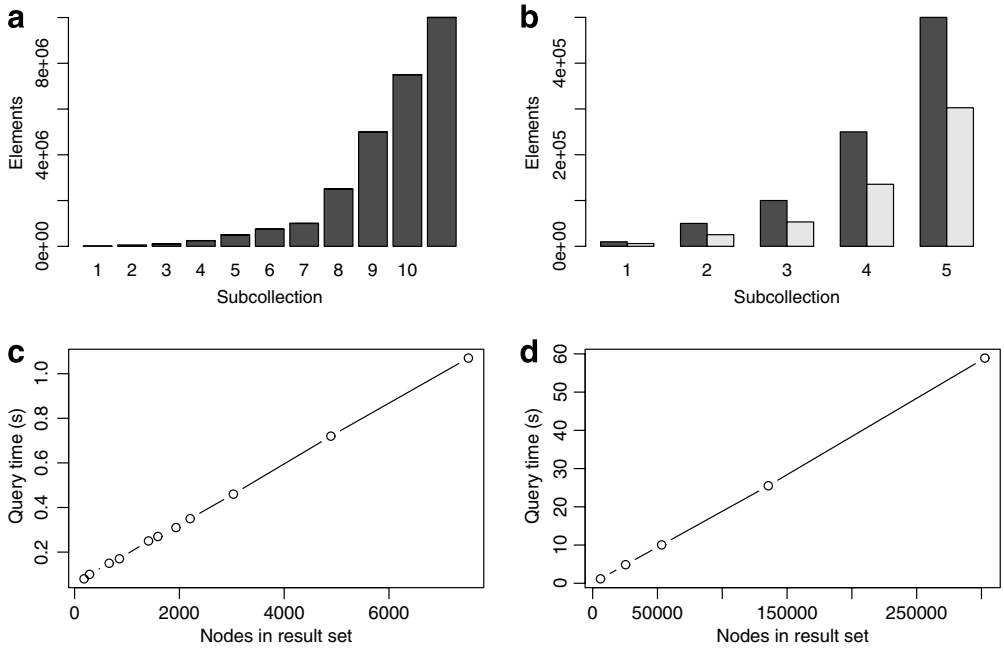


Fig. 14. (a) Total number of elements in each subcollection extracted from dataset 1. (b) Total number of elements (■) and number of relevant elements (□) in each subcollection extracted from synthetic dataset 2. (c) Execution time in dataset 1. (d) Execution time in dataset 2.

5.2. Effect of structural distortions

The second aspect we have evaluated is the effect of structural variations in fragments. In order to test this, we have generated another synthetic dataset, in which we have embedded potential matches of a test pattern containing 15 elements with the following kinds of controlled distortions: (1) addition of n random nodes; (2) deletion of n random nodes; (3) switching the order of nodes in the same level, with a given probability; (4) switching parent and child nodes, with a given probability.

Results in Fig. 15 show that the system is able to find all relevant fragments despite the introduced distortions. Predictably, only the removal of relevant nodes in the target has an effect in the average relevance of results. Adding nodes, switching nodes in the same level, and interchanging parent and child nodes have no effects on the retrieval rate.

5.3. Retrieval in highly heterogeneous collections

In our third set of experiments we analyze the retrieval effectiveness in the context of highly heterogeneous collections. Our goal is to evaluate the effectiveness of our approach according to different parameters, such as the size of the data collection, its degree of vocabulary and structural heterogeneity, and its degree of conformance to the query.

We experimented on a set of document collections patterned after the highly heterogeneous ASSAM[4] dataset. While the information content of the generated collections is the same of that of the real ASSAM, the new collections are bigger and present different levels of heterogeneity and conformance to the queries. The generation is based on empirical estimations of the relative probabilities of ASSAM's main topics and entities. Since the queries present a tree structure as well, pattern generators have been employed for the automatic

[4] http://moguntia.ucd.ie/repository/datasets/.

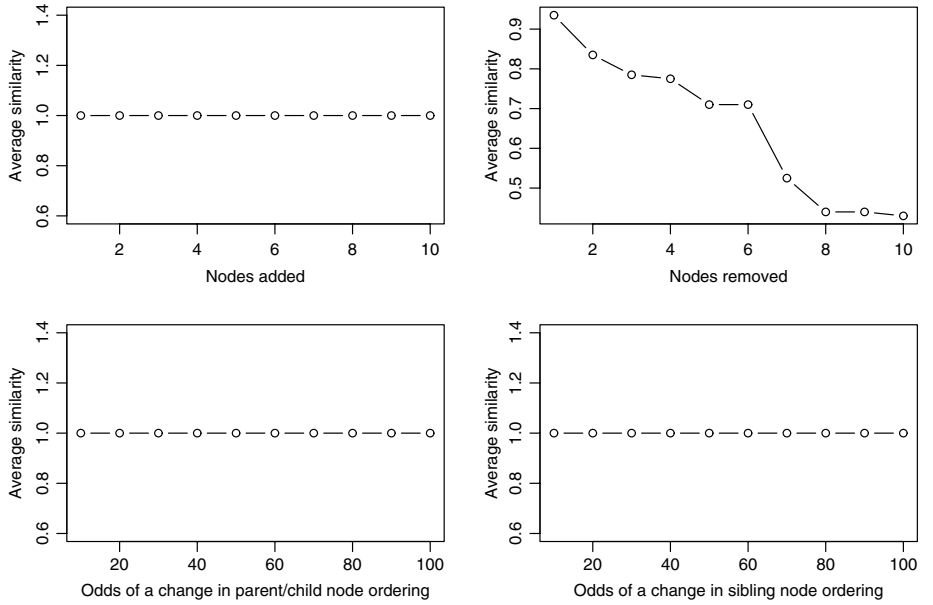


Fig. 15. Change in similarity with the addition and removal of nodes in regions.

construction of synthetic queries. The use of a probabilistic approach for the generation of queries simulates the users' uncertainty about the structure of the target collection in query formulation.

To obtain quality measures such as precision and recall we need relevance assessments. These are manually evaluated on the original ASSAM collection. In the generated collections, query identifiers are linked to the pattern used for the generation of the collection, thus specifying which generated subtrees constitute a correct answer for the query. This allows us to build collections of arbitrary size while being able to assess the relevance of each query answer.

In what follows, we first describe the generation of documents and queries and then present the experimental results on the original collection and on the generated ones.

Generation of documents and queries. To generate suitable test collections and queries we have built a new system whose generation model significantly expands ToXgene [12] facilities, that only provides limited support for "random structures", for the specification of heterogeneous contents.

Our collection generator provides a set of *XML pattern constructors*, which encapsulate different templates for generating XML structures. A *generator pattern* is itself an XML document, whose nodes represent both pattern constructors and the tags to be generated. In general, the ancestor/descendant relationships between these tags will be preserved in the generated documents. Each node in a generator pattern may have associated a probability indicating the likelihood of finding the subtree it represents in the generated documents (if omitted, the probability is taken to be *1.0*). Thus, if node u is a child of node v, then $P(u|v) = P(u) \cdot P(v|v')$, where v' is the parent of v. If u is the pattern root node, then $P(u)$ is the probability of the whole pattern. This simple model supports a wide variety of possible heterogeneous structures; Table 3 shows the main pattern constructors available.

Table 4 shows the summary of the generation patterns used in our experiments. They mainly reflect the features of the ASSAM dataset. As this dataset is mainly a semi-structured representation of a set of web services, their structures are quite heterogeneous and tag names also present many variations. In this collection, tag names are usually phrases that combine in some way the key concepts of the web service. This feature has been simulated using the `a:combi` pattern constructor, estimating n-gram probabilities from ASSAM data. This has been simulated by introducing subpatterns with the constructor `a:subPattern`. In Table 4 the last column indicates the used subpatterns in each generator pattern. Two sample generators models are shown in Fig. 16.

Table 3
Constructors for pattern generators

Constructor	Pattern examples	Generated XML
a:genPattern	⟨a:genPattern a:id='1'⟩ ⟨address/⟩ ⟨/a:genPattern⟩	⟨address/⟩
sequence	⟨tl⟩⟨a:sequence⟩ ⟨a/⟩⟨b/⟩ ⟨/a:sequence⟩⟨/tl⟩	⟨tl⟩ ⟨a/⟩⟨b/⟩ ⟨/tl⟩
xor	⟨tl a:xor='yes'⟩ ⟨a/⟩⟨b/⟩ ⟨/tl⟩	(*a*) ⟨tl⟩⟨a/⟩⟨/tl⟩ (*b*) ⟨tl⟩⟨b/⟩⟨/tl⟩
combi	⟨a:combi⟩ ⟨a/⟩⟨b/⟩ ⟨/a:combi⟩	a) ⟨a_b/⟩ b) ⟨a/⟩ c) ⟨b/⟩
if-ancestor	⟨a:if-ancestor a:name='tl'⟩ ⟨tag2/⟩ ⟨/a:if-ancestor⟩	⟨tl⟩... ⟨tag2/⟩...⟨/tl⟩
types	⟨person a:types='user,client'/⟩	a) ⟨user⟩ b) ⟨person⟩ c) ⟨client⟩
dmin, dmax	a) ⟨tl⟩⟨t2 a:dmin='0' a:dmax='0'/⟩⟨/tl⟩ b) ⟨tl⟩⟨t2 a:dmin='2' a:dmax='2'/⟩⟨/tl⟩	a) ⟨tl_t2⟩ b) ⟨tl_t2⟩ ⟨tl⟩⟨kjkkij⟩⟨t2/⟩ ⟨/kjkkij⟩⟨/tl⟩ ⟨/tl_t2⟩
a:subPattern	⟨tl⟩⟨a:subPattern a:id='1'/⟩⟨/tl⟩	⟨tl⟩⟨address/⟩⟨/tl⟩

Table 4
Database and query patterns used in experiments

Pattern	ID	Tags	Depth	Uses
Database patterns				
Weather info	BD-1	31	4	Address
Postal address	BD-2	14	4	–
Stock quotes	BD-3	16	4	–
Bank info	BD-4	9	8	Address, Stock
Credit card	BD-5	8	3	Bank Info
Personal data	BD-6	19	8	Address, Bank Info
Query patterns				
Weather	Q-1	12	3	–
ZIP codes	Q-2	3	1	–
PostBox	Q-3	3	1	–
Quotes	Q-4	5	2	–
Bank code	Q-5	3	1	–
ATM location	Q-6	3	2	Address
Contact person	Q-7	5	2	–
Employee salary	Q-8	3	2	–
Credit card	Q-9	3	2	–
User account	Q-10	4	2	–

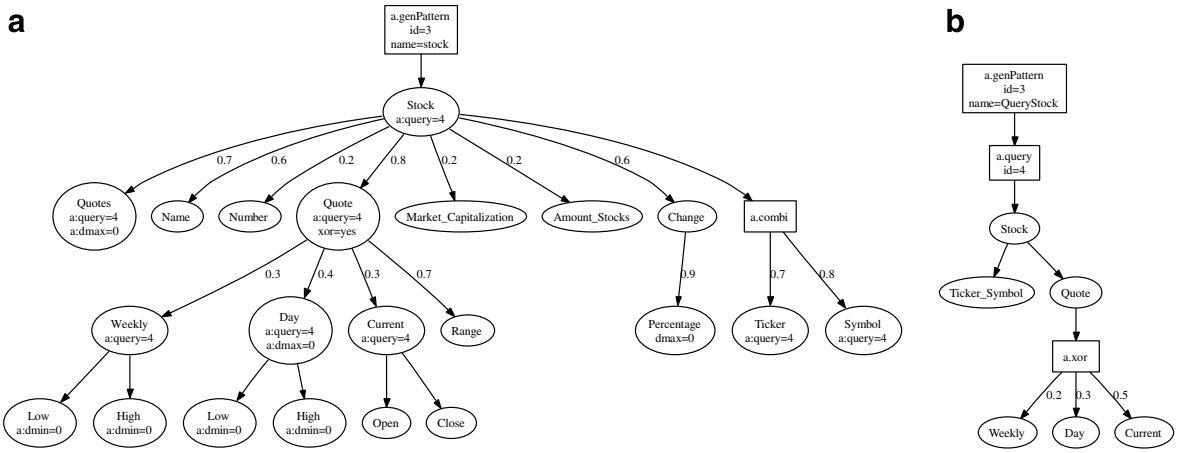


Fig. 16. (a) Generator for database pattern 3. (b) Generator for query 4, associated with pattern 3.

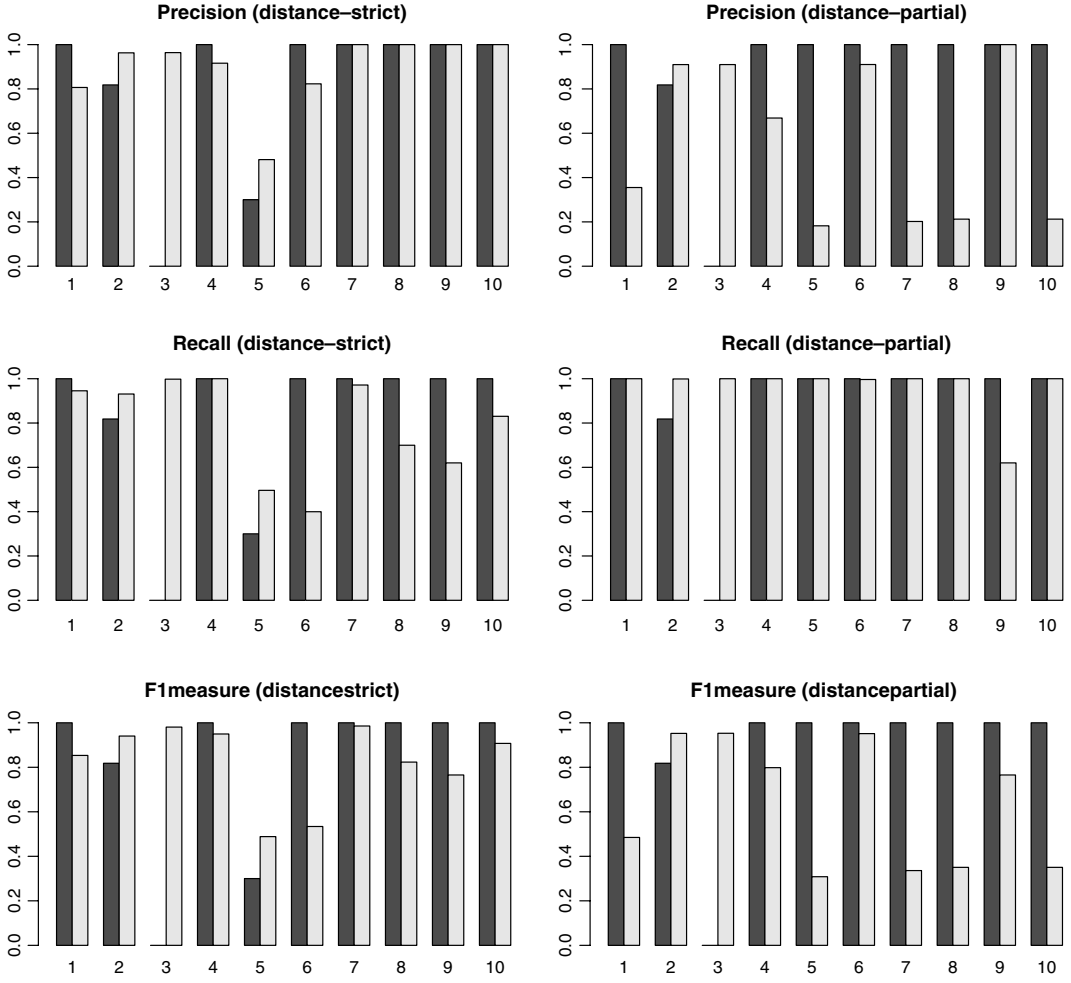


Fig. 17. Precision, recall and F_1 for measure *distance*, using *strict* and *partial* label matching. For each of the 10 queries, the darker bars (■) are computed using only the highest-ranked 10 results, the lighter bars (□) consider all the returned matches.

Results on the generated documents and queries. The generated queries were evaluated against a generated collection of 50000 documents, with a total of one million nodes and around five thousand unique labels (around 500 times larger than the original ASSAM). Given the characteristics of the collection, we selected the following parameters for the experiments:

- We compared the performance of *strict* label matching with a *partial* matching function adapted to the characteristics of the collection as described above. The resulting partial matching function combines most of the similarity functions described in Section 2.2.
- We used the structural distance similarity measure described in Section 3.2, with $\delta = 0.1$; the results obtained with the level measure were similar for this particular collection.

Fig. 17 shows the precision, recall and F_1-measure, as well as the precision@10, recall@10 and F_1@10 (i.e. the values express the 10 highest-ranked results).

The results under these experimental conditions can be summarized as follows:

- Setting the cutoff at 10 produces much better results than considering all the results. There is an exception for query 3; a closer analysis of this particular case shows that query 3 contains very ambiguous tags, producing a large set of irrelevant results despite the use of tag similarity functions.
- As expected, the results obtained using strict label matching produce a better precision than those obtained using partial label matching, while partial matching produces better recall. Overall, the F_1 measure is generally better for strict matching on all the results, but the combination of 10-highest ranked and partial matching is the best combination.

Results on the original ASSAM collection. Analogous queries were evaluated against the original ASSAM collection, and the relevance of the results checked by hand. A summary of the results is shown in Fig. 18. The first column indicates the query number, and the second column indicates the similarity threshold used for the answer set (we used an explicit cutoff percentage *MinSim* instead of selecting the k highest results due to the relatively small size of the collection). The results are closely related to those obtained in our more general experimental setting, including the low F_1 for query Q3.

6. Related work

The need of shifting from exact queries with boolean answers to proximity queries with ranked approximate results is a relevant requirement of XML query languages for searching the Web and several approaches in this direction have been developed. These approaches share the goal of integrating structural conditions typical of database queries with weighted and ranked approximate answers typical of Information Retrieval (IR). Work from the IR area [13] is mainly concerned with the retrieval of XML documents, by representing them with known IR models. These approaches mainly focus on the textual part of XML documents and structure is only used to guide user queries.

A crucial issue in ranked approximate querying is how to score results. A great variation of XML similarity measures have been proposed for different applications (see [14,15] for a survey). In this section, we will briefly discuss the similarity measures the most meaningful approaches to XML approximate querying rely on to rank results. Further specific structural similarity measures have been proposed in the areas of Schema Match-

Query	*MinSim*	*Prec.*	*Recall*	F_1
Q1 (Weather)	0.2	1	0.75	0.86
Q3 (Addresses)	0.15	0.29	0.7	0.41
Q4 (Bank data)	0.2	1	0.8	0.89

Fig. 18. Results for the ASSAM datasets.

ing [16] and more recently Ontology Alignment [17]. In the former, tree/graph-matching techniques allow determining which schema portions from the target schema can be mapped to the source ones. In this context, data types and domain constraints are used to decide if two nodes match. However, schemas usually exhibit simple structures that seldom exceed three depth levels (relation-attribute-type). In the latter, the problem consists in finding out which concepts of a target ontology can be associated to concepts of the source ontology.

In this section, we focus on approximate query answering for XML documents taking both vocabulary and structural heterogeneity into account. First, we discuss the kind of heterogeneity taken into account and the similarity measures employed to score results. Then, algorithmic approaches to query relaxation and to compute the top-k results are reviewed.

Heterogeneity degree and similarity evaluation. An early approach for XML approximate retrieval is ELIXIR [18] that allows approximate matching in data content (tree leaves), and ranks the results according to the matching degree, disregarding structure in the evaluation. No structural heterogeneity is considered, and vocabulary heterogeneity is allowed only for data content elements, not for tags.

More sophisticated approaches, like XIRQL [19] and XXL[20], accept approximate matching at nodes and then discuss how to combine weights depending on the structure. Vocabulary heterogeneity is supported for content and element tags, but no structural heterogeneity is allowed: conditions on document structure are interpreted as filters, thus they need to be exactly satisfied. XXL supports a similarity operator and, to use this operator, the user should be aware of the occurrence of similar keywords or element tags.

In the approach proposed by Damiani and Tanca [21] both XML documents and queries are modeled as graphs labeled with fuzzy weights capturing the information relative relevance. They propose to employ both structure related weighting (weight on an edge) and tag related weighting (weight on a node). Some criteria for weighting are proposed such as the weight decreases as moving far away from the root, the weight depends on the dimension of the subtree. Shortcut edges are considered, thus allowing the insertion of nodes, which weight is function of weights of edges. The match score is a normalized sum of weight of edges.

In the tree relaxation approach [2] exact and relaxed weights are associated with query nodes and edges. The score of a match is computed as the sum of the corresponding weights, and the relaxed weight is function of the transformations applied to the tree. The considered transformations are: *relax node*, replacing the node content with a more general concept; *delete node*, making a leaf node optional by removing the node and the edge linking it to its parent; *relax edge*, transforming a parent/child relationship to an ancestor/descendant relationship; *promote node*, moving up in the tree structure a node (and the corresponding subtree).

The approXQL [5] approach can also handle partial structural matchings. All the paths in the query are however required to occur in the document. The allowed edit operations on the document tree are delete node, insert intermediate node, relabel node. The score of match is function of the number of transformations, each one of which is assigned with a user-specified cost.

Amer-Yahia et al. in [3] account for both vocabulary and structural heterogeneity and propose scoring methods that are inspired by *tf*idf* and rank query answers on both structure and content. Specifically, twig scoring accounts for all structural and content conditions in the query. Path scoring, by contrast, is an approximation of twig scoring that loosens the correlations between query nodes when computing scores. The key idea in path scoring is to decompose the twig query into paths, independently compute the score of each path, and then combine these scores.

Table 5
Heterogeneity degree in XML approximate querying

	Vocabulary heterogeneity	Structural heterogeneity	Ranking
ELIXIR	On content	–	Structure independent
XIRQL	On content	–	Structure dependent
XXL	On content and tag	–	Structure dependent
[21]	On content and tag	Node insertion	Structure dependent
[2]	On content and tag	Delete node, relax edge, promote node	Structure dependent
approXQL	On content and tag	Node insertion, node deletion	Structure dependent
[3]	On content and tag	Edge generalization, leaf deletion, subtree promotion	Structure dependent

The main differences among the considered approaches are summarized in Table 5. Note how approaches to XML approximate queries have progressively shifted to higher degrees of heterogeneity, and to cope also with structural heterogeneity. Our approach allows for more significant structural variations and can be considered as a step forward in the treatment of structural heterogeneity in the context of XML. All the considered approaches, indeed, enforce at least the ancestor–descendant relationship in pattern retrieval, whereas our approach also allows to invert and relax this relationship. Our approach, moreover, is highly flexible because it allows choosing the most appropriate structural similarity measures according to the application semantics.

Tree embedding and top-k processing. A different perspective from which approaches to XML approximate querying started, is that of looking for approximate structural matches, allowing the structure of the document and query trees to partially match. Kilpeläinen [1] discusses ten different variations of the tree inclusion problem, that is, the problem of locating instances of a query tree Q in a target tree T. He orders these different problems, ranging from unordered tree inclusion to ordered subtrees, highlighting the inclusion relationships among them. Moreover, he gives a general schema of solution, and instantiates it for each problem, discussing the resulting complexities. His goal, however, is not to measure the distance, nor to rank results. All the instances of the pattern in the data tree are returned.

Another approach aimed at identifying the matches but that does not rank them, is by Kanza and Sagiv [22]. They advocate the need of more flexible query evaluation mechanisms in the context of semi-structured data, where both queries and data are modeled as graphs. They propose mapping query paths to data paths, so long as the data path includes all the labels of the query path; the inclusion needs not to be contiguous or in the same order.

Sub-optimal approaches for ranked tree-matching have been proposed for dealing with the NP complexity of the tree inclusion problems widely discussed and analyzed in [1]. In these approaches, instead of generating all (exponentially many) candidate subtrees, the algorithms return a ranked list of ''good enough'' matches. For example, in [23] a dynamic programming algorithm for ranking query results according to a cost function is proposed. In [2] a data pruning algorithm is presented where intermediate query results are filtered dynamically during evaluation process. `ATreeGrep` [24], instead, uses as basis an exact matching algorithm, but allowing a fixed number of ''differences'' in the result. [4] proposes an adaptive query processing algorithm to evaluate approximate matches where approximation is defined by relaxing XPath axes. In adaptive query processing, different plans are permitted for different partial matches, taking the top-k nature of the problem into account.

Starting from the user query, the approaches [2–5], for structural and content scoring of XML documents, generates relaxations of the query structure that preserve the ancestor–descendant relationships existing among nodes in the query. These relaxations do not consider the effective presence of such a structure in the collection of documents. The number can also be really high when the number of nodes in the query is high. The novelty of our approach relies on the employment of ad-hoc indexing structures. By exploiting such auxiliary information, only the variations on the pattern that occur in the target are considered. The number of variations to be evaluated is thus considerably reduced, still allowing for greater structural heterogeneity, as discussed in the previous section. Finally, in [2–5] constraints in the user queries are relaxed to augment the obtained results, whereas in our approach, we start from a completely relaxed query. We follow this approach because of the high degree of heterogeneity of the collection of documents we work on and we apply indexing structures for improving the performance.

7. Conclusions and future work

In this paper we have developed an approach for the identification of subtrees similar to a given pattern in a collection of highly heterogeneous semi-structured documents. In this context, the hierarchical structure of the pattern cannot be employed for the identification of the target subtrees but only for their ranking. Peculiarities of our approach are the support for tag similarity relying on a thesaurus, the use of indexing structures to improve the performance of the retrieval, and a prototype of the system.

As future work we plan to compare different similarity measures in order to identify those more adequate depending on the application context and the heterogeneity of the considered data. Such measures can be col-

lected in a framework of functions that a user can select, compose, and apply depending on her needs. Moreover we plan to consider more sophisticated patterns in which further constraints on vertices and edges can be stated. For instance, an element or a portion of the pattern could be requested to mandatorily appear in the target regions, or the difference between the levels in which two elements appear could be constrained by fixing a threshold. The constraints should then be considered in the similarity evaluation. Finally, we wish to consider subtree identification in a collection of heterogeneous XML Schemas. In this context, the proposed approach should be tailored to the typical schema constraints (e.g., optionality and repeatability of elements, groups, types).

Appendix A. Details of similarity evaluation

Detailed match-based similarity (see Example 9). Let x_p be the vertex tagged `article` in the pattern P in Fig. 5 and x_r^1, x_r^2, x_r^3 the corresponding vertices in the regions R_1, R_2, R_3. In R_1, `article` appears with the same label then the match-based similarity is 1, whereas in the other two regions a similar tag appears, then the match-based similarity is $1-\delta$.

Consider now Table 2b. Since R_1 has two perfect node matches over three nodes in the pattern, the resulting similarity is $\frac{2}{3}$. Region R_2 has two perfect matches and a similarity match, over three nodes in the pattern, thus the resulting similarity is $\frac{3-\delta}{3}$. All the three mappings with region R_3 have two similarity matches and no exact matches over three nodes in the pattern, thus the resulting similarity is $\frac{2}{3}\cdot(1-\delta)$ in all cases.

Detailed level-based similarity (see Example 10). In Table 2a, the level-based similarity is 1 for region R_1 because the vertex tagged `article` in that region appears at the same level it appears in the pattern. By contrast, the level-based similarity between x_p and x_r^2 is $\frac{1}{2}-\delta$ because their labels are similar there are two levels of difference (the level of x_r^2 is 3 and the level of x_p is 1) and the maximal number of levels in P and R_2 is 4. The level-based similarity between x_p and x_r^3 is $\frac{1}{2}-\delta$ because their labels are similar there is one level of difference (the level of x_r^3 is 2 and the level of x_p is 1) and the maximal number of levels in P and R_3 is 2.

Consider now Table 2b. Since R_1 has two perfect (both tags and levels coincide) node matches over three nodes in the pattern, the resulting similarity is $\frac{2}{3}$. Consider now region R_2. The similarities are: $\frac{1}{2}-\delta$ for node labeled `article`, $\frac{1}{2}$ for node labeled `title`, and $\frac{3}{4}$ for node labeled `conference`, thus resulting in $\frac{7}{12}-\frac{\delta}{3}$. For region R_3, the bottom mapping in Fig. 6 results in $\frac{1}{2}-\delta$ for node labeled `article` and $1-\delta$ for node labeled `title`, thus in $\frac{1}{2}-\frac{2}{3}\cdot\delta$ overall similarity. The left mapping in Fig. 6 yields to the same similarity. By contrast, the right mapping in Fig. 6 yields to $1-\delta$ both for node labeled `article` for node labeled `title`, thus in $\frac{2}{3}\cdot(1-\delta)$ overall similarity, which is the highest one.

Detailed distance-based similarity (see Example 11). The distance-based similarity between x_p and x_r^1 is 1 because the vertex is in the same position (root). In region R_2, vertex `article` has distance 3 (where the maximal distance is 4) whereas the distance in the pattern is 1. Its distance-based similarity is thus $1-\delta-\frac{3-1}{4}$. Node `article` has distance 3 in R_3 (where the maximal distance is 3) whereas the distance in the pattern is 1. Its distance-based similarity is thus $1-\delta-\frac{3-1}{3}$.

Consider now Table 2b. Since R_1 has two perfect (both tags and positions coincide) node matches over three nodes in the pattern, the resulting similarity is $\frac{2}{3}$. Consider now region R_2. The similarities are: $\frac{1}{2}-\delta$ for node labeled `article`, $\frac{1}{2}$ for node labeled `title` and for node labeled `conference`, thus resulting in $\frac{1}{2}-\frac{\delta}{3}$. For region R_3, the bottom mapping in Fig. 6 results in $\frac{1}{2}-\delta$ for node labeled `article` and $1-\delta$ for node labeled `title`, thus in $\frac{1}{2}-\frac{2}{3}\cdot\delta$ overall similarity. The left mapping in Fig. 6 yields to $1-\delta$ for node labeled `conference` and to $\frac{2}{3}-\delta$ for node labeled `article`, thus in $\frac{5}{9}-\frac{2}{3}\cdot\delta$ overall similarity. By contrast, the right mapping in Fig. 6 yields to $1-\delta$ both for node labeled `article` for node labeled `title`, thus in $\frac{2}{3}\cdot(1-\delta)$ overall similarity, which is the highest one.

Appendix B. Proof sketches

Proof Sketch of Proposition 1. We prove the properties of this proposition through Fig. 10. In the figure, we present a fragment F with a node v at level m and we consider the nodes that can occur at level l ($m < l$). These

nodes are those that can occur at level l in the pattern index for the creation of a fragment F. The following situations can arise.

(1) If $head(PI(l)) = t$ is a left relative of F, t is the root of another fragment. Indeed, all the fragments whose roots precede root(F) have been removed. Therefore, t cannot belong to other fragments and it is the root of a new fragment.
(2) If $head(PI(l)) = u$ is a descendant of root(F), but not of v, u is left relative of all internal nodes of F. If this is not true, there would be a vertex p in F (in a level between $level$(root(F)) and l) that precedes v for which u would have been added as child of p. Therefore, u can only be the direct child of root(F).
(3) If $head(PI(l)) = w$ is a descendant of v, for the same reasons of previous item, w can only be a direct child of v.
(4) If $head(PI(l)) = n$ is a right relative of F, no elements of F can be identified at this level. Indeed, n and the nodes that follows it have a higher rank than that of root(F). □

Proof sketch of Proposition 2. The proposition directly follows from Proposition 1 and the considerations of how Algorithm 2: *CreateListOfFragments* works. Indeed, if a node v at a given level of the current PI is not removed by the recursive calls of Algorithm 1: *CreateFragments* on nodes at level $m < level(v)$, it means that it is the root of a fragment and will be removed along with all its descendants when Algorithm 2: *CreateListOfFragments* is invoked on its level. □

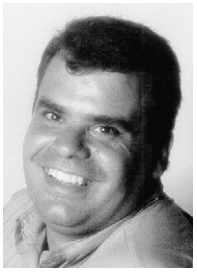
Proof Sketch of Proposition 3. The number of operations for the entire process of region construction is the maximum between the number of operations required for the construction of the pattern index and those required for the extraction of regions from the pattern index. Therefore, exploiting the complexity specified for these operations, the process of region construction is in $\mathcal{O}(\max\{|P| \cdot K \cdot M), N \cdot |P|\})$. □

References

[1] P. Kilpeläinen, Tree Matching Problems with Applications to Structured Text Databases. PhD thesis, Deptartment of Computer Science, University of Helsinki, 1992.
[2] S. Amer-Yahia, S. Cho, D. Srivastava, Tree pattern relaxation, in: Proceedings of the 8th International Conference on Extending Database Technology. LNCS 2287, Springer, 2002, pp. 496–513.
[3] S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, D. Toman, Structure and content scoring for XML, in: Proceedings of the 31st International Conference on Very Large Data Bases, 2005, pp. 361–372.
[4] A. Marian, S. Amer-Yahia, N. Koudas, D. Srivastava, Adaptive processing of top-k queries in XML, in: Proceedings of the 21st International Conference on Data Engineering, 2005, pp. 162–173.
[5] T. Schlieder, Schema-driven evaluation of approximate tree-pattern queries, in: Proceedings of the 8th International Conference on Extending Database Technology, LNCS 2287, Springer, 2002, pp. 514–532.
[6] J. Freund, D. Comaniciu, Y. Ioannis, P. Liu, R. McClatchey, E. Morley-Fletcher, X. Pennec, G. Pongiglione, Health-e-child: an integrated biomedical platform for grid-based paediatric applications, HealthGRID, 2006.
[7] I. Sanz, M. Mesiti, G. Guerrini, R. Berlanga-Llavori, Approximate subtree identification in heterogeneous XML documents collections, in: Proceedings of the Third International XML Database Symposium. LNCS 3671, Springer, 2005, pp. 192–206.
[8] T. Grust, Accelerating XPath location steps, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2002, pp. 109–120.
[9] W. Meier, eXist: An Open Source Native XML Database, in: Web, Web-Services, and Database Systems. NODe 2002 Web- and Database-Related Workshops, LNCS 2593, Springer, 2002.
[10] A. Nierman, H.V. Jagadish, Evaluating Structural Similarity in XML Documents, in: Proceedings of the Fifth International Workshop on the Web and Databases, 2002, pp. 61–66.
[11] P. Buneman, S.B. Davidson, M.F. Fernandez, D. Suciu, Adding structure to unstructured data, in: Proceedings of the Sixth International Conference on Database Theory, LNCS 1186, Springer, 1997, pp. 336–350.
[12] D. Barbosa, A. Mendelzon, J. Keenleyside, K. Lyons, ToXgene: a template-based data generator for XML, in: Proceedings of the Fifth International Workshop on the Web and Databases (WebDB), 2002.
[13] R.W. Luk, H. Leong, T.S. Dillon, A.T. Chan, W.B. Croft, J. Allen, A survey in indexing and searching XML documents, Journal of the American Society for Information Science and Technology 53 (2002) 415–438.
[14] G. Guerrini, M. Mesiti, E. Bertino, Structural similarity measure in sources of XML documents, in: J. Darmont, O. Boussaïd (Eds.), Processing and Managing Complex Data for Decision Support, IDEA Group Publishing, 2006, pp. 247–279.
[15] G. Guerrini, M. Mesiti, I. Sanz, An overview of similarity measures for clustering XML documents, in: A. Vakali, G. Pallis (Eds.), Web Data Management Practices: Emerging Techniques and Technologies, IDEA Group Publishing, 2006, pp. 56–78.
[16] E. Rahm, P.A. Bernstein, A survey of approaches to automatic schema matching, The VLDB Journal 10 (2001) 334–350.

[17] J. Kwon, O.H. Choi, C.J. Moon, S.H. Park, D.K. Baik, Deriving similarity for semantic web using similarity graph, Journal of Intelligent Information Systems 26 (2006) 149–166.
[18] T.T. Chinenyanga, N. Kushmerick, An expressive and efficient language for XML information retrieval, Journal of the American Society for Information Science and Technology 53 (2002) 438–453.
[19] N. Fuhr, K. Grossjohann, XIRQL: a query language for information retrieval in XML documents, in: Proceedings of the 24th ACM SIGIR Conference on Research and Development in Information Retrieval, 2001, pp. 172–180.
[20] A. Theobald, G. Weikum, Adding relevance to XML, in: D. Suciu, G. Vossen, (Eds.), Proceedings of the Third International Workshop on the Web and Databases. LNCS 1997 (2000), pp. 105–124.
[21] E. Damiani, L. Tanca, Blind queries to XML data, in: Proceedings of the11th International Conference on Database and Expert Systems Applications, 2000, pp. 345–356.
[22] Y. Kanza, Y. Sagiv, Flexible queries over semistructured data, in: Proceedings of the 20th ACM Symposium on Principles of Database Systems, 2001.
[23] T. Schlieder, F. Naumann, Approximate tree embedding for querying XML data, in: Proceedings of the ACM SIGIR Workshop on XML and Information Retrieval, 2000.
[24] D. Shasha, J.T.L. Wang, H. Shan, K. Zhang, ATreeGrep: approximate searching in unordered trees, in: Procedings of the 14th International Conference on Scientific and Statistical Database Management, 2002, pp. 89–98.

Ismael Sanz is a lecturer at the Department of Computer Science and Engineering at Universitat Jaume I (UJI) in Spain. After receiving a B.Eng. in Computer Science from UJI in 1997 he worked at STARLab in the Free University of Brussels and in the private sector as a software engineer. He holds a M.Sc. in Computer Science in 2003 from UJI, and is expected to complete a Ph.D. in 2007. His current research interests include flexible query processing for complex and heterogeneous data, and approximate retrieval of XML.



Marco Mesiti received the master and Ph.D. degrees in computer science in 1998 and 2003, respectively, from the University of Genova, Italy. Since 2005 he is assistant professor at the department of informatics and communication (DICO) at the University of Milano, Italy. His current research interests include management of semi-structured data, querying and classification of XML documents, schema evolution and access control mechanisms for XML. He has co-organized the dataX workshop at EDBT 2004 and EDBT 2006 and served as PC member of IEEE SAINT 2005, IEEE SAINT 2006, ADBIS 2006, EDBT Ph.D. Workshop 2006, EDBT Workshop ClustWeb 2004, and as reviewer for international conferences and journals.

Giovanna Guerrini received the Ph.D. degree in Computer Science in 1998 from the University of Genova, where she currently is associate professor in the Department of Computer and Information Sciences. She had been assistant professor at the University of Genova (1996–2001) and associate professor at the University of Pisa (2001–2005). Her current research interests include object-oriented, active, and temporal databases as well as semi-structured and XML data handling. She has served as PC member of international conferences and symposia, including ECOOP, ACM OOPSLA, EDBT, ACM CIKM, XSym. She has co-organized the Object Database Workshop at ECOOP'99 and ECOOP'00 and the dataX workshop at EDBT'04 and EDBT'06.

Rafael Berlanga-LLavori is an associate professor in Computer Science at University Jaume I, Spain for 12 years. He received the B.S. degree from Universidad de Valencia in Physics, and the Ph.D. degree in Computer Science in 1996 from the same university. He is author of several articles in international journals, such as Information Processing & Management, Decision Support Systems, Concurrency: Practice and Experience, Applied Intelligence, among others, and numerous communications in international conferences such as ICDE, DEXA, ECIR, CIARP, etc. His current research interests are knowledge bases, information retrieval, and temporal reasoning.