Knowl Inf Syst (2006)
DOI 10.1007/s10115-005-0230-9

Knowledge and Information Systems

REGULAR PAPER

K. Selçuk Candan · Jong Wook Kim · Huan Liu · Reshma Suvarna

Discovering mappings in hierarchical data from multiple sources using the inherent structure

Received: 22 August 2004 / Revised: 24 January 2005 / Accepted: 26 March 2005 / Published online: 30 January 2006
© Springer-Verlag 2006

Abstract Unprecedented amounts of media data are publicly accessible. However, it is increasingly difficult to integrate relevant media from multiple and diverse sources for effective applications. The functioning of a multimodal integration system requires metadata, such as ontologies, that describe media resources and media components. Such metadata are generally application-dependent and this can cause difficulties when media needs to be shared across application domains. There is a need for a mechanism that can relate the common and uncommon terms and media components. In this paper, we develop an algorithm to mine and automatically discover mappings in hierarchical media data, metadata, and ontologies, using the structural information inherent in these types of data. We evaluate the performance of this algorithm for various parameters using both synthetic and real-world data collections and show that the structure-based mining of relationships provides high degrees of precision.

Keywords Retrieval and mining of semantics · Extracting and mining semantics from multimedia databases · Knowledge discovery in XML · Hierarchical multimedia · Ontologies · Multimodal integration · Structure-based mining

1 Introduction

Universality, i.e., the need for accessing media, collected and indexed independently by various applications and organizations, necessitates uniform organization schemes that would allow easy access and integration of media. Instead, media is mostly available to users and applications in diverse structures

K. S. Candan (✉) · J. W. Kim · H. Liu · R. Suvarna
Department of Computer Science and Engineering, Arizona State University, Tempe
AZ 82857, USA
E-mail: {candan, jong.wook.kim, huan.liu, reshma.suvarna}@asu.edu

and formats. Furthermore, considering the multitude and diversity of these applications, it is not viable to expect a global unifying scheme.

Semantic networks of media, wherein different applications can exchange information and integrate multimodal data, require information about each media to be represented in a detailed and structured manner. To enable such information exchanges, various hierarchical metadata frameworks have been proposed. For instance, Resource Description Framework (RDF) [3], supported by the World Wide Web Consortium, aims at providing a means for the description of metadata, in an organized, informative, searchable, and accessible manner. RDF schemas are implemented for representing multimedia data such as image, audio, and video files. An example schema could consist of three different parts:

- Dublin Core [13] schema is used for identifying the photograph and describing properties such as creator, editor, and title.
- Technical schema is used for capturing technical data about the photo and the camera such as the type of camera, type of film, scanner, and software used for digitization.
- Content schema is used for categorizing the subject of the photo by means of a controlled vocabulary. This schema allows photos to be retrieved based on such characteristics as portrait, group portrait, landscape, architecture, sport, etc.

RDF and similar metadata description frameworks provide a common language through which metadata, such as media and application ontologies, are exchanged. This enables software systems to create a uniform structure to represent and organize data, which renders the integrated data manageable and retrievable. Consequently, many multimedia standards define objects as a structured collection of media objects.

The metadata (such as content descriptors and feature names), used to describe resources in RDF and other metadata description languages, are defined by various communities. The metadata creators, depending on the specific application, culture, use different terms for similar concepts. For the functioning of an automated multimodal media integration system, the semantics behind the metadata terms used by various authors and communities should be mined and automatically related. Hence, a mechanism to mine and relate semantically similar but syntactically different metadata is required.

In this paper, we develop algorithms to automatically mine concept mappings in hierarchical media data, metadata, and ontologies. We propose a solution which mines such relationships using the inherently available structural information. We use Multidimensional scaling (MDS) [13, 22, 23, 51] to map the nodes between the two different but similar structures (multimedia hierarchies, ontologies, or namespaces) such that the syntactically different but semantically similar components map to each other.

In Sect. 2, we present our motivation. In Sect. 3, we define the problem formally and in Sect. 4, we propose a solution and develop an efficient algorithm. Then, in Sect. 5, we evaluate the performance of the algorithm for various parameters and show that it is very effective in addressing this challenge.

2 Motivations

In this section, we present two related motivations for this work. This first one, integration of multimedia resources described in resource description framework, requires mining of mapping of hierarchical namespaces (or ontologies). The second application, XML multimedia document matching and integration, requires mining and mapping of components (elements or attributes) in various multimedia objects.

2.1 Integration of RDF described media resources

If application and media content experts could easily associate metadata with each resource they create, then this knowledge could be used by access and integration engines to increase their efficiency and precision.

Within the context of web information integration, many proposals are made to the World Wide Web Consortium (W3C) for representation of Web-related metadata. Initial solutions were based on the $< META >$ tag of the HTML. Currently, many companies, such as Microsoft, IBM, Motorola, Netscape, Nokia, OCLC, are actively participating in the field of metadata framework developments. In 1997, Netscape submitted a proposal, titled "Meta Content Framework", to W3C [19]. The two principles on which the meta content framework (MCF) is based are (1) there is no distinction between the representation needs of data and metadata, and (2) for interoperability and efficiency, schemas for different applications should share as much as possible in the form of data structure, syntax, and vocabulary. The culmination of various frameworks was the resource description framework [24]. RDF provides application developers with a foundation for the description of metadata for the next generation of interoperable applications [3].

Ontologies, formalisms that define the relationships among terms in a given application, describe the context in which metadata is applied. They are used to link, compare, and differentiate information provided by various application resources. RDF provides a data model where entities and relationships can be described. The relationships in RDF framework are first class objects, which means that relationships between objects may be arbitrarily created and be stored separately from the two objects. This nature of RDF is very suitable for dynamically changing, distributed, shared nature of the Web.

The metadata (property names) used to describe resources are generally application-dependent and must be associated with RDF schema. This, however, can cause difficulties when RDF descriptions need to be shared across application domains. For example, Fig. 1 represents an RDF statement for resource www.asu.edu. The property (metadata) used to define the resource University_1 are, *Name* and *Location*. However, the property *Location* can be defined in some other application domain as *Address*. Although the semantics of both property names are the same, syntactically they are different. In general a property name may have a broader or narrower meaning depending upon the needs of particular application domains. To prevent such conflicts and ambiguities, the terminology used by each application domain must be clearly identified.

RDF uniquely identifies property names by using the Namespace mechanism [41]. A namespace can be thought of as a context or an ontology that gives a

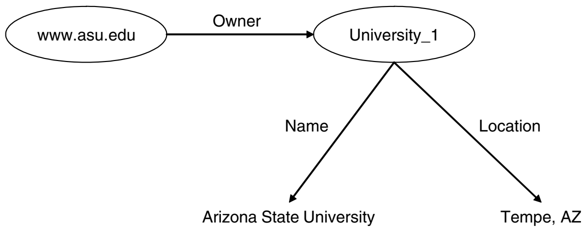


Fig. 1 Example RDF statement

specific meaning to what might otherwise be a general term. It provides a method for unambiguously identifying the semantics and conventions governing the particular use of property names by uniquely identifying the governing authority of the vocabulary. Although with the help of namespaces, we can uniquely identify and relate the metadata to a particular governing authority or a community, there is no straightforward way to map and relate terms or properties among different communities.

Consider the two hierarchical namespaces provided in Fig. 2 (the hierarchy usually corresponds to the concept/class hierarchy) of the underlying domains. As it is implied by the similar structures of these namespace hierarchies, the terms *Processor* and *CPU* are semantically related. Therefore, if the user integrates two data domains each using one of these two namespaces, whenever a query is issued using the property name *CPU*, the content having the property name *Processor* should also be retrieved.

Automatic mapping of the semantically similar but syntactically different terms from the different namespaces is one of the necessities for integration of content from independently created data sources. An automated mechanism needs to be devised that relate the common and uncommon terms of various metadata communities.

2.2 Matching of XML specified media objects

Many multimedia standards define objects as structured collections of media data. XML description of such multimedia objects and structures is very common. Examples include virtual reality modeling languages (X3D), media content description frameworks (MPEG7 [50]), e-commerce web documents, and geographic information systems. Extensible markup language (XML) [15] defines a generic syntax used to mark up data with simple, human readable tags. It provides

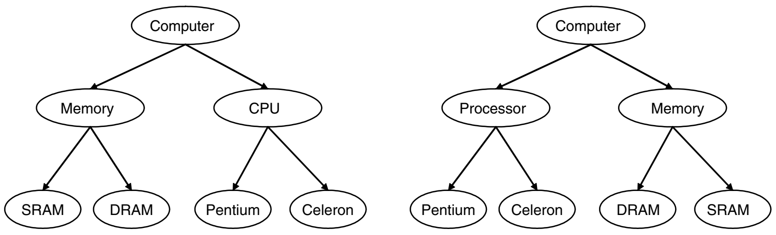
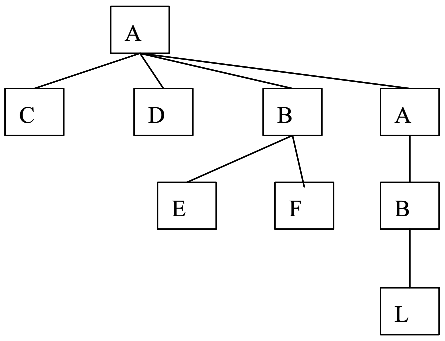


Fig. 2 Similar hierarchical namespaces



```
<!ELEMENT A(C*,D?,B*,A*)>
<!ELEMENT B(E?,F?,L?)>
<!ELEMENT C(#PCDATA)>
<!ELEMENT D(#ANY)>
<!ELEMENT E(#CDATA)>
<!ELEMENT F(#ANY)>
<!ELEMENT L(#CDATA)>
```

Fig. 3 An example XML document and the corresponding DTD

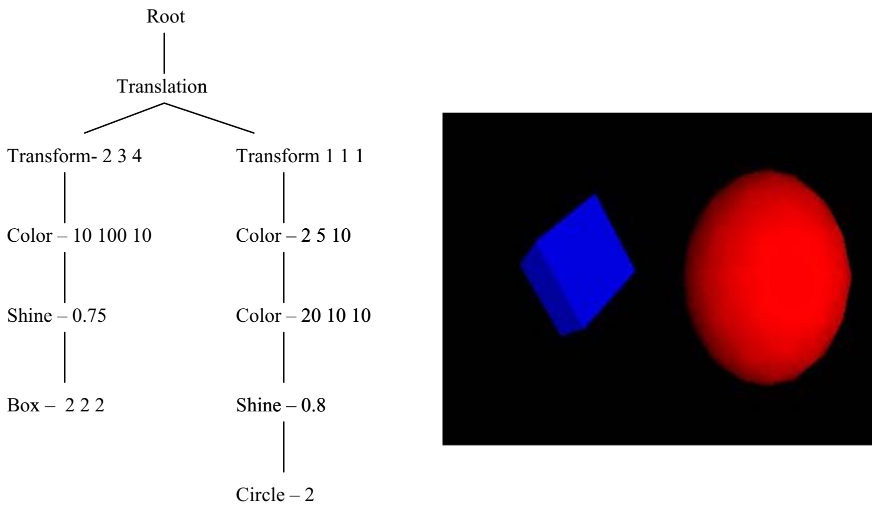


Fig. 4 An example X3D document and the corresponding 3D world

a standard format for computer documents. As shown in Fig. 3, an XML document is a tree-like structure, whose structure may be defined through a Document Type Definition (DTD) or through an XML-schema. XML is very flexible; its attributes and sub-elements can be either missing or repeated. DOM [12] and LORE [33] are two well-known tree-based data models for XML documents. Each node of the tree corresponds to an element or an attribute of an element in the XML document. The root node contains the document's root. A child node corresponds to a subelement or an attribute of the parent node. For each child of a node, besides the pointer to the child, there is a tag in the node that indicates the name of the child node. If the child is a subelement, the name is its element tag. If the child is an attribute, the name is the attribute name.

XML became the de facto standard for multimedia data representation. For example, X3D [14], a file format and related access services for describing interactive 3D objects and worlds, is based on XML. X3D nodes are expressed as XML elements, i.e., tagged names (see Fig. 4 for an example X3D document).

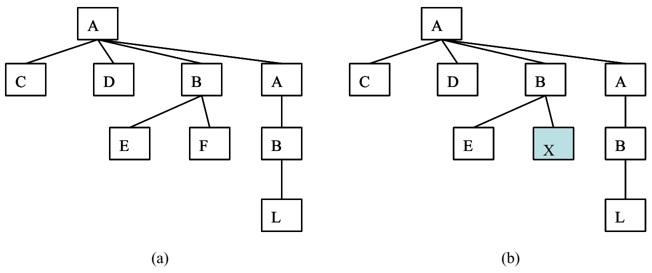


Fig. 5 Two similar XML documents; the node labels *F* and *X* are likely to denote semantically similar elements

XML tags can be used for describing the data in the form of a hierarchical structure. This provides flexibility and expressive power to the data description framework, but it also complicates the integration task as different domains could have different sets of rules, tags, and properties to represent the same data. Although, whenever they are available, DTDs and namespaces provide information about the structure of the XML files, not all XML documents have associated DTDs. In fact, one of the main reasons why XML is becoming a de facto information exchange standard is that each XML document (even when it does not have an associated DTD) is self-describing: the hierarchical structure and the tags in this hierarchy gives information about the relationships between the tags, i.e., the data elements and their attributes. Consequently, given two similar multimedia objects, where different tag names are used to denote similar concepts, it should be possible to make the association between these tags using an analysis of the hierarchical XML document structures. For example, in Fig. 5, nodes labeled *F* and *X* are likely to correspond to each other.

3 Problem statement

The problem we address in this paper is to mine mappings between the nodes of hierarchical media data, metadata, and ontologies. The main observation is that the structural similarity between two given trees (hierarchical media objects, XML documents, or name spaces) can provide clues about the semantic relationships between their nodes.

In general, the nodes in the two trees can be divided into common and uncommon nodes. The common nodes are those shared by the two trees and can either have the same labels or (in the case of multimedia data) they may have application-dependent features that provide high degrees of similarity [7]. In this paper, we do not focus on how common nodes are discovered. Our aim is to relate the uncommon nodes of a two given hierarchical structures. Therefore, formally, we can state the problem as follows.

Given

- two trees, *T1(V1, E1)* and *T2(V2, E2)*, where *V* denotes the nodes in the tree and *E* denotes the edges between the nodes,
- a partial mapping, *M*, between the nodes in *V1* and *V2*, (we call those nodes in *V1* and *V2* that have a mapping, *the common nodes*) and

– two unmapped nodes v_i in *V1* and v_j in *V2*,

compute the similarity between v_i and v_j.

For example, given the two trees in Fig. 5, the user or the content-integrator might want to find which node in the first tree corresponds to the node labeled X in the second tree. (In this example, purely based on the structures of the two trees, we can conclude that X corresponds to F in the first tree.)

The use of structural information for mining of semantic relationships is not new. We used structural information available on the web for mining web document associations, summarizing web sites, and answering web queries [5, 6, 28]. [25, 43, 46, 47] have used the language taxonomies and IS-A hierarchies to define the similarity and distance between terms in natural language. These mainly rely on the observation that given a tree, *T(V, E)* and two nodes, *a* and *b*, on the tree, we can compute a distance, *d(a, b)*, between the nodes by considering the structure of the tree, for instance by counting the number of edges between them. *The main challenge we face in this paper for finding the similarity between two nodes in two* different *trees, on the other hand, is that there is no common structure to help compare these two nodes*; since the two trees may have arbitrarily different structures, finding a mapping between the nodes is not trivial.

4 Proposed approach

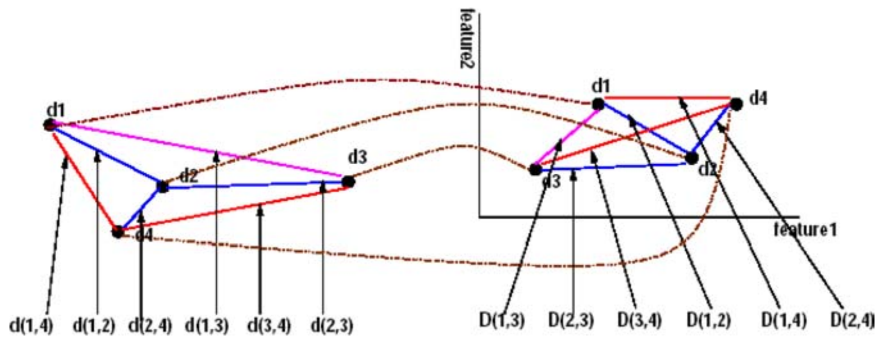
To match two nodes in two different trees, we need to find a mapping such that the distance values in two trees between the common nodes are preserved as much as possible. In this paper, we address this challenge by mapping the two trees into a common space using the matching nodes and comparing the unmapped nodes in this common space. The proposed solution can be broken down into four steps.

1. Map the nodes of *T1* and *T2* into two multidimensional spaces, *S1* and *S2*, both with the same number of dimensions.
2. Identify transformations required to align the space *S1* with the space *S2* such that the *common nodes* of the two trees are as close to each other as possible in the resulting aligned space.
3. Use the same transformations to map the *uncommon nodes* in *S1* onto *S2*.
4. Now that the nodes of the two trees are mapped into the same space, use clustering and nearest-neighbor algorithms to find the related *uncommon nodes* in the two trees.

4.1 Step I: Map both trees into multidimensional spaces

We map the trees based on the common nodes. For example, the common nodes of two given namespaces might include the shared terms such as *University* and *College* that are known to denote similar concepts.

Multidimensional scaling is a family of data analysis methods, all of which portray the structure of the data in a spatial fashion [13, 22, 23, 51]. MDS is used to discover the underlying spatial structure of a set of data items from the distance information among them. MDS works as follows, it takes as inputs (1) a set of N objects, (2) a matrix of $N \times N$, containing pairwise distance values, and (3) the



(a)

$d(1,2) \approx D(1,2) \quad d(1,3) \approx D(1,3) \quad d(1,4) \approx D(1,4)$
$d(2,3) \approx D(2,3) \quad d(2,4) \approx D(2,4) \quad d(3,4) \approx D(3,4)$

Fig. 6 MDS mapping of four data points onto a two-dimensional space

desired dimensionality k. Given these inputs, MDS tries to map each object into a point in the k-dimensional space (the mapping process of documents, given their distances with respect to one another, is shown in Fig. 6). The criterion for the mapping is to minimize a stress value which is calculated as

$$\text{stress} = \sqrt{\sum_{i,j} \frac{(d'_{i,j} - d_{i,j})^2}{\sum_{i,j} d_{i,j}^2}}$$

where d_{ij} is the actual distance between two nodes vi and vj and d'_{ij} is the distance between the corresponding points pi and in the k-dimensional space. Thus if we can maintain the distance between pi and pj the same as the distance between vi and vj then the stress is 0.

MDS starts with an *initial configuration* of points. In this work, we assume MDS uses a random configuration. It then applies the some form of steepest descent iteratively to minimize the stress.

In a tree-structured data, for example in a namespace, similar or related nodes are closer to each other and have less number of edges between them than the dissimilar nodes. The closer the nodes the shorter the distance between them in distance matrix [25, 43, 46, 47], hence similar or related nodes are mapped closer to each other in a multidimensional space. In other words, MDS maps the similarity between points: similar nodes are mapped closer to each other and dissimilar ones are mapped far off from each other.

4.2 Step II: Find a transformations required to map the common nodes of the two trees closer to each other in the space

Once both trees are mapped onto two separate k-dimensional spaces, we need to relate the common nodes of the two trees. To achieve this, we identify transformations required to map the common nodes from both trees to each other as close as possible in a shared space. To match the common nodes, we use the Procrustes alignment algorithm [18, 20]. Given two sets of points, the Procrustes algorithm uses linear transformations to map one set of points on the other set of points. The general Procrustes algorithm seeks the isotropic dilation and the rigid translation, reflection and rotation needed to best match one configuration to another [18]. In our case, the inputs to the algorithm are the nodes (terms) common to T1 and T2.

Note that this step uses an initial partial mapping between a subset of the nodes in the two input trees, and the structural information inherently available, to discover the required transformation. In this paper, as in many works [11, 37, 42], we assume that the matching between the two input trees relates each one element of one schema to only one elements of the other; thus, the initial partial mapping is 1:1. Furthermore, we assume that the input mapping is also non-fuzzy (i.e., a given node perfectly maps to the other one or does not map to that node at all). These assumptions are valid especially when the two tree structures represent metadata, such as schemas, where the mapping is naturally 1:1 and binary. However, when the two input trees being compared are media trees (such as MPEG7 [50] or X3D [14]), two complications that may arise. First of all, the mapping between the nodes may be $1{:}n$ or $n{:}n$, i.e., a given node may correspond to multiple nodes in the other tree. Second, the correspondence between the object nodes may be less than perfect, i.e., fuzzy. Hence, this second step of the proposed algorithm, where we identify transformations required to align the two input spaces such that the *common nodes* of the two trees are as close to each other, we need to consider the *many-to-many* and *fuzzy* nature of the common nodes. The intuitive solution to this problem, which we use in this paper, is to eliminate the *many-to-many* and *fuzzy* nature of the mappings by (1) not considering the mappings below a certain quality of match, (2) for each node in one of the trees, selecting the best mapping node as the corresponding peer, and (3) eliminating the rest of the low ranking mappings from further consideration. An alternative approach would be to use, instead of Procrustes, an alignment algorithm which takes the *many-to-many* and *fuzzy* nature of the mappings while identifying the appropriate linear transformations to map one set of points on the other set of points. In this paper, we do not investigate this second approach.

4.3 Step III: Use the same transformations to map the uncommon nodes

The previous step returns the transformations required to modify the given spaces such that the common nodes of both trees conform to each other as much as possible. This matching between the common nodes of both trees can, then, be used to define the similarity between the uncommon nodes. Using the transformations identified in the previous step, the uncommon nodes in two trees are mapped into the space in terms of their distances from the common nodes in respective trees. The uncommon nodes of both trees that are approximately at the same distance or at the same distance range from the common nodes in their respective trees are likely to be similar and will be mapped close to each other in the shared k-dimensional space.

4.4 Step IV: Use clustering to find the related uncommon nodes from the two trees

At this point, we have two trees whose nodes are mapped onto a shared k-dimensional space such that the common nodes are close to each other in the space. Hence, we can use clustering and nearest-neighbor approaches to identify related uncommon nodes.

To retrieve the related points from the multidimensional space, we use a k-means [31] based clustering algorithm, which requires centroids to be given to form clusters around. We use the nodes of one tree, as the centroids for the clustering and we use the distances in the Euclidean space to achieve clustering. As a result, returned clusters contain the node in one tree specified as the centroid and one or more nodes from the other tree that are closest in the shared space. Thus, in the form of a cluster, we have pairs of nodes from two different trees that are similar to each other.

5 Experimental evaluation

In this section, we provide an experimental evaluation of the proposed approach for mining mappings between the nodes of hierarchical media data, metadata, and ontologies. To evaluate the proposed approach and to observe the effects of various data parameters (like the number of nodes in the two trees and their degrees or fanouts), we needed a large number of trees. Furthermore, we needed to be able to vary these parameters in a controlled manner to observe the performance under different conditions. Therefore, we systematically generated a large number of tree-structured data (i.e., the ground truth) with varying parameters and use these trees in our initial experimental evaluation. After observing the effectiveness of our algorithm using this ground truth, we also used a real collection of data and verified our results.

5.1 Generating ground truth

The challenge addressed in this paper is to relate nodes of two trees that can differ from each other in terms of the number and density of nodes, or simply the node labels. Therefore, to generate two related but different trees, we

1. picked an original tree, and then
2. distorted the original tree by relabeling existing nodes, deleting nodes existing nodes in, and adding new nodes to the original tree (Fig. 7).

Thus, the original and the distorted trees act as two similar trees, different in terms of the number of nodes and the labels of some of the nodes. The two trees have some nodes that are common (undistorted) and some nodes that are uncommon.

5.1.1 Synthetic tree generation for controlled experiments

For the first set of experiments, where input data trees are generated in a controlled manner, we developed a tree generation program which creates a tree randomly

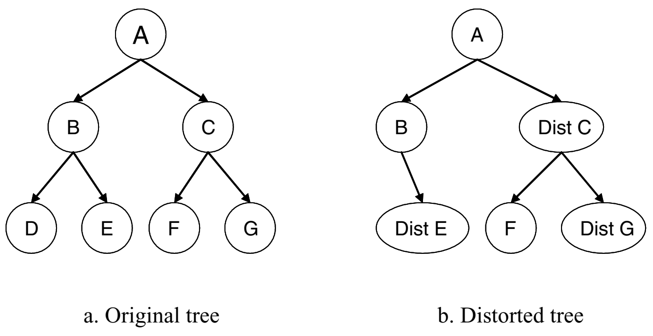


Fig. 7 Tree distortion process

based on two parameters: number of nodes and maximum fanout (degree). For our experiments, we have generated original trees with the configuration shown in Table 1. We generated $(4 \times 5 =)20$ sets of trees with a distinct combination of number of nodes and the fanout value. In our experiments, we used five different seed values for each combination; therefore, we have a total of 100 original trees for experiments. We report these results in Sects. 5.3, 5.4, and 5.5.

5.1.2 Experiments with real trees

In addition to synthetics trees, we also run our experiments with the TreeBank dataset [52], which has a deep recursive structure (whereas our synthetic trees were mostly balanced). We report these results in Sect. 5.6.

5.2 Terminology

Following are the terms used in discussing and explaining the experiment results.

– *Number of nodes*: The total number of nodes in the original tree.
– *Number of nodes that are mapped*: Number of nodes in the original tree + the number of nodes in the distorted tree.
– *Correct mapping*: When a given query node of a given tree does map to the corresponding node of the other tree, then the mapping is said to be correct mapping.
– *Erroneous mapping*: When a given query node of a given tree does not map to the corresponding node of the other tree, then the mapping is said to be an erroneous mapping. The types of erroneous mappings are

Table 1 Parameters for tree generation

Number of nodes in the tree	25, 50, 100, and 200
Fanout (degree)	1, 2, 4, 8, and 16

- mapping to a sibling of the correct node,
- mapping to the parent node of the correct node (the correct node does not have a sibling),
- mapping to the parent node of the correct node (the correct node has at least one sibling),
- mapping to the sibling of the parent,
- mapping to a distant node, and
- no mapping

– *Error percentage*: This is the ratio of the erroneous mappings in the number of mappings returned. Note that, at most one of the mapped nodes can be a correct map:

$$\left(\frac{\#\,\text{erroneous mappings}}{\#\,\text{of nodes mapped}}\right) \times 100$$

– *Precision*: The precision is measured as

$$\frac{m_1 + m_2 + \cdots + m_n}{k}$$

where k is the number of nodes returned and m_i is the degree of matching of node n_i in the result:

$$m_i = \frac{1}{1 + \text{err}_i}$$

Note that a node with a lower degree of error (err) contributes more to the precision. The degree of error is defined as follows for different types of erroneous mappings:

- mapping to a sibling of the correct node or [err=1],
- mapping to the parent node of the correct node (the correct node does not have a sibling) [err=1],
- mapping to the parent node of the correct node (the correct node has at least one sibling) [err=2],
- mapping to the sibling of the parent [err=3], and
- mapping to a distant node [err=4].

If the algorithm does not return any matches, the corresponding precision is defined as 0.

5.3 Experiment I: Label differences

In the first set of experiments, we aimed to see the performance of the proposed algorithm when the structures of the trees are identical, but some of the nodes are labeled differently.

For every original synthetic tree, we have generated four distorted trees using the *node rename* operation. The level of distortions experimented with are 5,

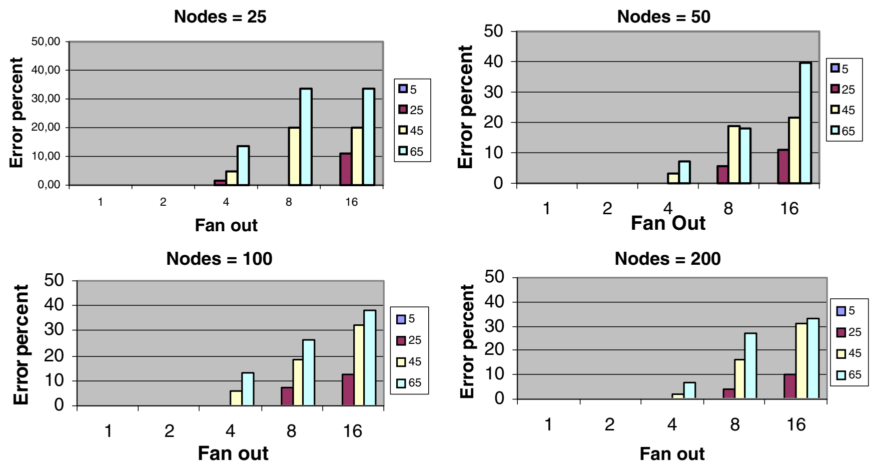


Fig. 8 Percentage mapping error in Experiment I

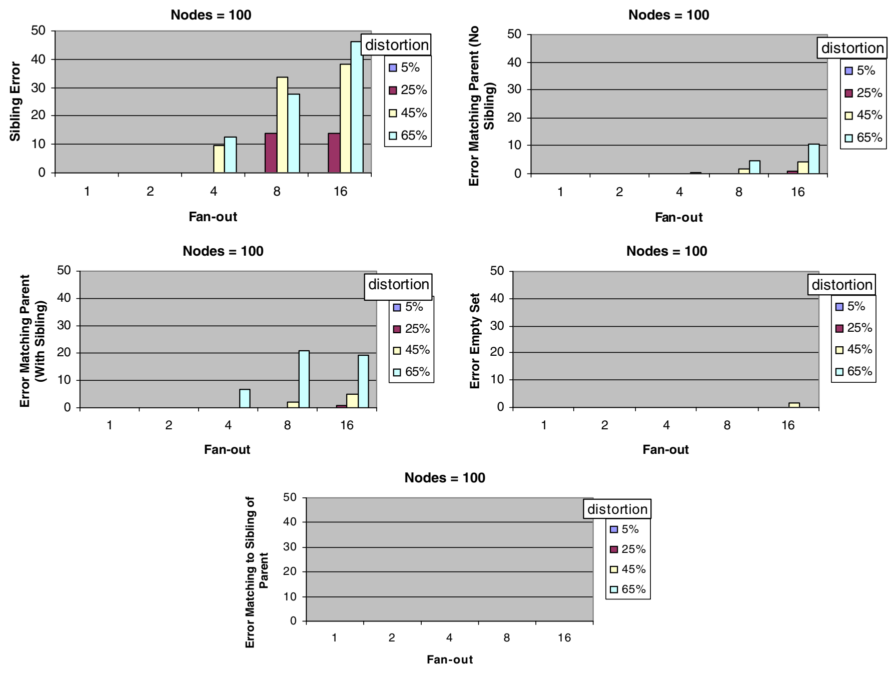


Fig. 9 Types of the errors (sibling match, parent match, parent/child match, empty set, sibling of parent match) for Experiment I

25, 45, and 65%. Therefore, we have a total of (100 × 4 =)400 test cases. For every query node, the implementation returns the matching nodes from the other tree. For every original tree we run four tests. Figures 8–10 provide the following observations.

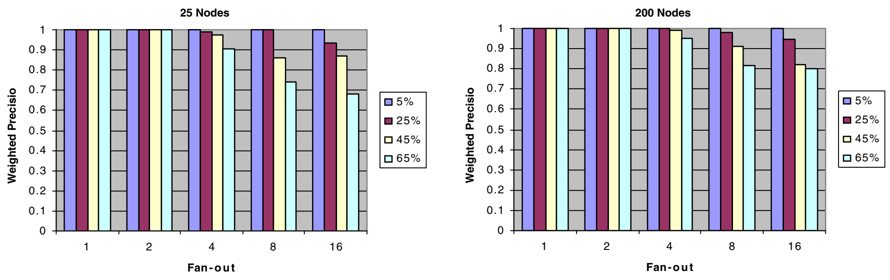


Fig. 10 Weighted precision in Experiment I

- As the distortion increases the error also increases. Most common errors are due to nodes that are mapped to a sibling of the correct node.
- The error pattern observed is similar in case of trees with a total number of nodes 25, 50, 100, and 200.
- As the fanout increases, the error also increases. For trees with fanout 1 or 2, no errors are observed.
- The precision is close to perfect (1.0) for low renaming distortions. For heavy renaming (65%), the weighted precision can drop slightly for trees with large fanouts.

5.3.1 The effect of distortion

The similarity mapping of the nodes of the original tree and the distorted tree is based upon two things: The distances between the nodes in each tree and the distances between the common nodes in both trees. The higher the number of common nodes between the two trees, the more similar the two trees are. Hence resulting mappings between the distorted nodes and the original nodes are better. Increase in the distortion reduces the number of common nodes between the two trees; as a result, the error rate increases.

5.3.2 The effect of fanout

For trees with maximum fanout 2, there is a high probability of correct mapping when only one of the two siblings is mislabeled. However, when the fanout is higher, the probability that siblings (especially the leaf siblings that are structurally identical to each other) will be erroneously mapped to each other increases. Hence, the rate of correct mapping decreases when the fanout increases.

5.3.3 Types of errors

Figure 9 presents different types of errors for trees with 100 nodes (the results are similar in other tree sizes as well). In Experiment I, the only operation that causes distortion is "renaming". Hence, although the names of some of the nodes are modified, the structure of the distorted tree is maintained. Consequently, in most error cases, the distorted nodes are simply mapped to a sibling of the correct node.

5.3.4 Precision

Figure 10, on the other hand, takes into account the degree of match even for those nodes that do not perfectly match the requested node. From this figure, it is again clear that the result precision is close to perfect (1.0) for low renaming distortions. In the case of heavy renaming (65%), the weighted precision drops as the fanout increases. However, the degree of drop is not significant, which means that even when the algorithm cannot find a perfect match, it returns a node close to what was expected.

More importantly, the results show that as the number of nodes in the tree increases, the weighted precision significantly improves. This shows that, as the number of available nodes increase, the distance-based mapping of nodes into the search space becomes more and more robust and this leads into better matches.

5.4 Experiment II: Structural differences

In Experiment II, we used combinations of addition, deletion, and rename operations to generate distortions in the original synthetic trees. This enabled us to observe the performance when the structures of the trees are also variable. For each of the 100 original trees, we apply three levels of distortions:

– 15% (5% of addition + 5% of deletion + 5% of rename),
– 30% (10% addition + 10% deletion + 10% rename),
– 45% (15% addition + 15% deletion + 15% rename).

Figures 11–13 provide the following observations:

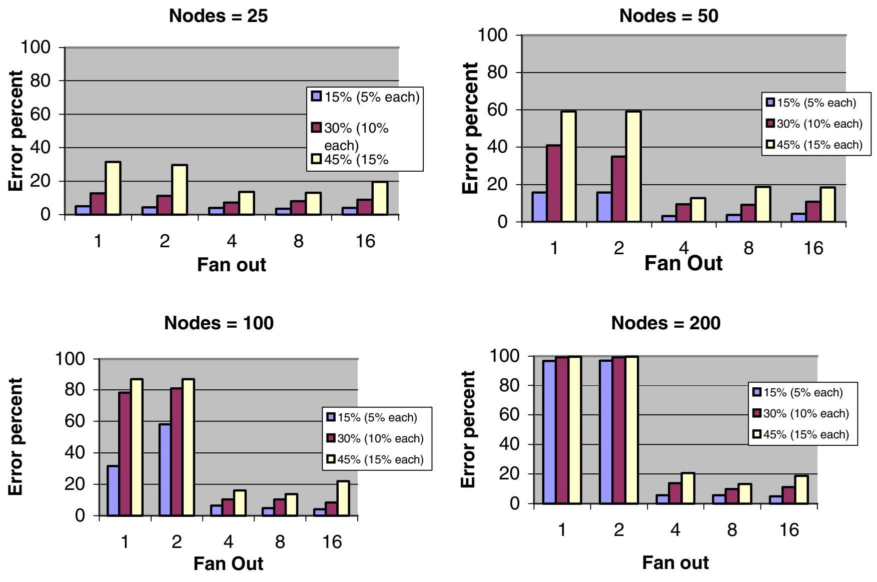


Fig. 11 Percentage mapping error in Experiment II

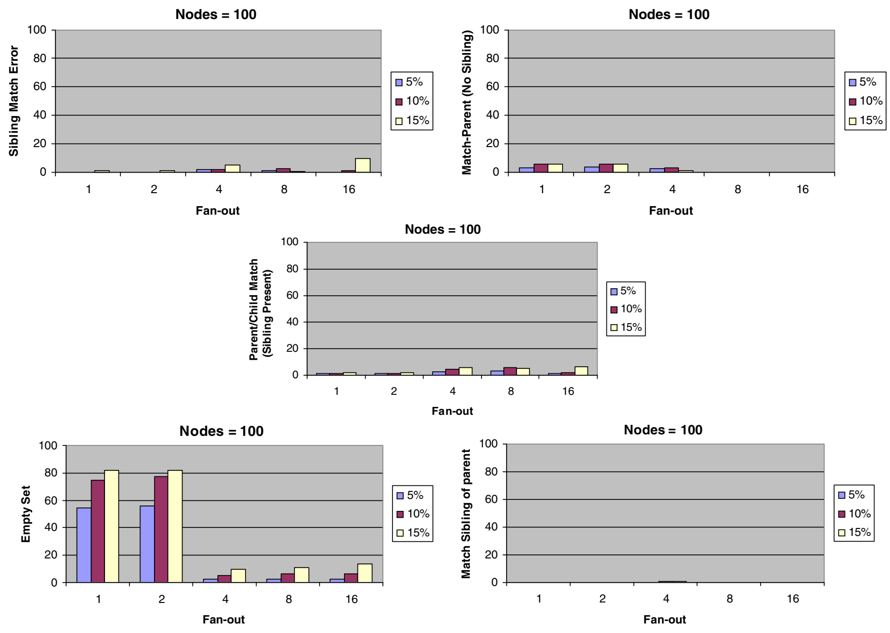


Fig. 12 Types of the errors (sibling match, parent match, parent/child match, empty set, sibling of parent match) for Experiment II

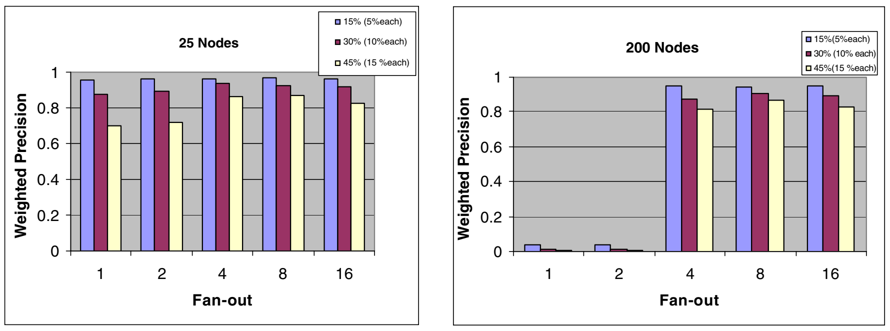


Fig. 13 Weighted precision in Experiment II

– As the distortion increases, the error also increases. The most significant errors are the empty matches.
– *Unlike* Experiment I, when the fanout is 1 or 2, the error percentage is the highest. The error percent drops sharply for fanout value of 4 and it remains more or less constant as fanout increases.

We next examine different effects in detail.

5.4.1 The effect of distortion

In Experiment II, we used a combination of addition, deletion, and rename operations to generate distortion. The expected observation in this experiment is that almost 75% of the test-runs return results that do not exhibit entirely correct mappings. These errors are due to the increase in the overall distortion. The higher the distortion is, the lower is the number of common nodes between the two trees; hence, the greater is the probability of wrong mapping. The type of distortion is also a key factor that influences the error proportion. As expected, the change in the tree structure (due to additions and deletions) has a negative effect on the error percentage.

5.4.2 The effect of fanout

In trees with lower fanouts, each node is closely related (nearby) to a few nodes. Each of these nearby nodes is highly important in achieving a correct mapping. If any of these nearby nodes is deleted, then the given node loses some important distance information. Hence, it becomes difficult to exactly map the node. As a result, either the node does not get mapped to any other node of the corresponding tree or it maps to the parent of the correct node (Fig. 11).

On the contrary, if the tree has high fanout, each node has a large number of siblings with which it is closely related. Even if one of these nodes is deleted, there are many other nodes for the given node to relate to. Although, there is an increased probability that the given node wrongly maps to a sibling, there is a relatively high probability of correct mapping.

5.4.3 Types of errors

Figure 12 presents different types of errors for trees with 100 nodes (again, the results are similar in other tree sizes). Unlike Experiment I, where most of the errors were caused by sibling matches, in this case, most of the errors are due to those nodes that do not match any other node in the distorted tree. As described, this is most prominent in trees with low fanout.

5.4.4 Precision

Figure 13 shows the weighted precision obtained by the proposed algorithm in the case of a combination of distortions. From this figure, it is clear that the result precision is large for large fanouts. An increase in the number of nodes in the tree, on the other hand, has different effects depending on the fanout of the nodes. If the fanout is low, a larger tree actually means a significant drop in the precision. If the fanout is large, however, a higher number of nodes in the tree actually improve the precision. This is in accord with the drop in error in combined distortions (Fig. 11).

Note that the precision values reported in Fig. 13 are computed based on the assumption that, if the algorithm returns an empty set of matches for a given node, then the corresponding precision is 0. However, since after deletion type of distortions, we should not expect the algorithm to return matches for every node, alternatively we could set the precision to 1 when the algorithm returns an empty set, but a matching node is known not to exist. Nevertheless, especially for low fanout cases, not all empty set errors are due to deletion type of distortions: thus, even if the definition of precision is modified, the precision value would increase only slightly.

5.5 Experiment I versus Experiment II

As expected, better results were observed in Experiment I as compared to Experiment II. In case of Experiment I, the only distortions were caused by renaming. Thus, the structures of the original and distorted tree were the same, resulting in better matches.

In these two experiments, we observed significantly different behaviors when it comes to the effects of fanouts.

– In Experiment I, a smaller fanout means smaller chance of mapping a node to the sibling of the correct node. Hence, a smaller fanout translates into a smaller error rate.
– In the case of Experiment II, on the other hand, when the fanout is very low, the overall tree structure could be drastically changed by a small amount of node deletions and additions. Since the proposed algorithm is based on the structure of the, the resulting error rate was considerably high in cases with low fanout. For large fanouts, however, too much renaming is more detrimental than structural change, as without enough matching nodes.

5.6 Experiment III: TreeBank collection

In addition to the synthetic trees we used in Experiments I and II, we also run additional experiments with the TreeBank dataset available at [52]. The deep recursive structure of this dataset (maximum depth, 36; average depth, 7.87), in contrast to the mostly balanced structures we used in experiments with synthetic trees, also provides opportunities for additional observations. For the experiments with real-world data, in order to observe the effects of distortion, we clustered the trees in the collection based on their numbers of nodes. Therefore, for instance, if we wanted to observe the precision of our algorithm for trees with 100 nodes, from the collection we selected trees that have around 100 nodes. Then, we applied various types of distortions on these trees.

5.6.1 Effects of label differences on TreeBank data

Figure 14a shows the weighted precisions obtained by the proposed algorithm in experiments with TreeBank data (with only node relabelings). The results show that the proposed algorithm is very robust with respect to relabeling errors in real data. Even when 65% of the nodes are relabeled, the approach is able to identify the correct node with up to 90% precision. When we compare the results presented in this figure with the results obtained using synthetic trees (Figs. 10 and 11), we see that for large fanouts, the precision the algorithm provides on real data

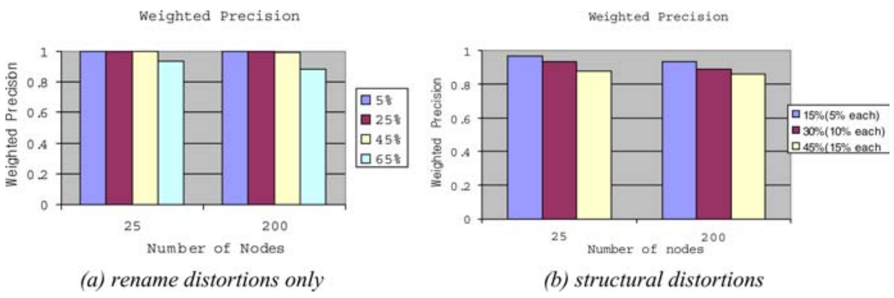


Fig. 14 Weighted precision in experiments with TreeBank data; **a** only rename distortions and **b** with structural distortions

is significantly larger (up to 90% precision even with 65% relabelings) than the precision obtained on synthetic tree sets (60% precision with 65% relabelings).

We observed that for trees with 200 nodes around 70% of the errors were due to nodes that did not match any other node, 14% of the errors were due to nodes that matched their siblings, and another 14% were due to nodes that matched sibling of their parents. This is in contrast with the results with synthetic data (Fig. 9) where the no-mapping errors were close to 0. Nevertheless, the overall precision was higher than the case for synthetic data; i.e., when there were nodes that are returned in the result set, the errors of these nodes were closer to 0.

5.6.2 Effects of structural differences on TreeBank data

Figure 14b shows the weighted precisions obtained by the proposed algorithm in experiments with TreeBank data (all types of distortions, including additions and deletions of nodes, are allowed). From this figure, it is clear that the precision behavior of the proposed algorithm in real data matches the precision behavior obtained using the synthetic tree set we have used in the previous experiments (Figs. 14 and 15).

As expected, due to structural distortions, the weighted precisions are lower than the case for only relabelings, but they are above 80% even with 45% combined distortions. As we discussed earlier, since due deletion type of distortions, we can not expect the algorithm to return matches for every node, an alternative definition of precision would be to set the precision to 1 when the algorithm returns an empty set and a matching node is known not to exist. When the definition of precision is modified this way, the precision values reported in Fig. 14 would rise above 90% even for high distortion cases, as the empty set errors are not overly penalized for deleted nodes.

5.7 Execution times

The final set of experiments is about the execution times required by the proposed algorithm under different matching scenarios for the TreeBank data and the results are presented in Tables 2 through 5. The experiments have been performed

Table 2 Execution times for matching under rename distortions

Rate of distortions (%)	25 nodes (s)	100 nodes (s)	200 nodes (s)
5	0.076	0.55	1.94
25	0.079	0.57	2.57
45	0.079	0.69	2.74
65	0.080	0.73	3.10

Table 3 Execution times for matching under structure distortions

Rate of distortions (%)	25 nodes (s)	100 nodes (s)	200 nodes (s)
$(5+5+5)$ 15	0.078	0.64	2.66
$(10+10+10)$ 30	0.082	0.66	3.09
$(15+15+15)$ 45	0.083	0.67	3.26

Table 4 The way time is split among the individual steps of the algorithm

Dist. (%)	Total (s)	Step I (%)	Step II (%)	Step III (%)	Step IV (%)
			25 Nodes		
15	0.078	10.9	2.6	0.6	85.9
30	0.082	10.9	3.0	0.6	85.5
45	0.083	10.8	2.4	0.6	86.2
			200 Nodes		
15	2.04	22.2	1.1	0.1	76.6
30	2.47	19.2	1.1	0.0	79.7
45	2.63	18.3	1.2	0.0	80.5

Table 5 The effect of the fanout on the execution time of the algorithm

Fanout	Step I (s)	Step II (s)	Step III (s)	Step IV (s)	Total (s)
1	0.62	0.01	0.001	0.59	1.22
2	0.62	0.01	0.001	0.61	1.24
4	0.60	0.05	0.001	2.60	3.25
8	0.59	0.09	0.003	3.79	4.47
16	0.59	0.12	0.001	5.13	5.85

on a PC with Pentium M CPU 1400 MHz and 512 MB main memory running Windows XP. The transformations were implemented using MatLab 6.5. Each value presented in these tables is computed as the average of results from 20 experiments.

Table 2 presents the case where the structures of the trees that are being matched are similar, but a certain portion of the nodes are relabeled. As shown in the table, the execution time increases with both the number of nodes in the trees that are being matched and the amount of distortion that has to be accounted for. Nevertheless, the algorithm scales well against the amount of rename distortions, while the required time is quadratic in the number of nodes in the trees that are being compared. Thus, the number of nodes is a more important factor in total execution time than the amount of distortion. Table 3 on the other hand presents the time needed for matching when the trees are also structurally distorted. Again, the execution time needed is similar to (only very slightly higher than) the case with only rename distortions.

Table 4 shows how the execution time is split among the four individual steps (mapping trees onto a space using MDS, finding transformations using Procrustes, mapping uncommon nodes using these transformations, and finding the appropriate matchings using k-means clustering) of the algorithm. As shown in this table, 80–85% of the time is spent on the final (clustering) step of the algorithm, while the first (MDS) step of the algorithm takes 10–20% of the total time. As seen prominently for the 200 nodes case, when the amount of distortions increases, the major contributor to the corresponding increase in the execution time is the clustering step of the algorithm. This is expected as, due to distortions, the precision drops; this causes more matches to be found and returned, slightly increasing the execution time. On the other hand, when we compare the two tables for 25 nodes and 200 nodes, respectively, we see that while the execution times for both first and last steps of the algorithm increase in absolute terms, the major contributor to the large increase in the overall execution time is the first step where MDS is used for mapping trees onto multidimensional spaces.

Finally, Table 5 shows the effect of the tree fanout change on the execution time of the algorithm. In order to observe this effect, we synthetically created trees with varying degrees of fanout. Here, we are reporting the results for a set of experiments with trees with 200 nodes and an overall 30% structural distortion rate, and the results for other scenarios are similar. The value of fanout affects the execution time, especially of the clustering phase, significantly. Given a fixed number of nodes, when the fanout is large, the distance between the nodes are smaller (more nodes are siblings or close relatives of each other); this leads to more work at the clustering and cluster-based retrieval phase of the algorithm, increasing the total execution time significantly.

6 Related work

Matching has been recognized as an important problem in diverse application domains. For instance, automated schema or model matching, which takes two schemas as input and produces a mapping between elements of the two schemas that correspond semantically to each other, has been investigated in various data management contexts, including scientific, business, and web data integration [11, 27, 34, 38]. A survey of the techniques for the automated schema matching problem, presented in [44], classifies these based on various dimensions, including whether data instances are used for schema matching, whether linguistic information, key constraints, or other auxilary information are used for matching, and whether the match is performed for individual elements (such as attributes) or for complex structures. Using the approach we presented in this paper, both data instances and hierarchical schemas can be matched. The approach does not need additional linguistic information or key constraints, though these can certainly help improving the overall precision. The proposed approach uses not only the individual elements, but the entire structures to produce the required mappings between the elements of the two schemas.

Clio [35] accepts XML and RDF documents, a name matcher provides initial element-level mapping, and a structural matcher provides the final mapping. LSD [11] uses machine-learning techniques to match a new data source against a previously determined global schema, thus it needs a user-supplied mapping as well as a training process to discover characteristic instance patterns and matching rules. SKAT [38, 39] uses first-order logic rules to express match and mismatch relationships between two ontologies. Name and structural matching is performed based on the is–a relationships between the intersection (or articulation) ontology and source ontologies. The work in [37] uses structures (schema graphs) for matching; matching is performed node by node starting at the top; thus this approach presumes a high degree of similarity (i.e., low structural distortion) between the schemas. Furthermore, unlike our approach, if no match is found for a given node, user intervention is required to select a match candidate. After performing linguistic matching of the nodes, Cupid [32] transforms the original schema into a tree and then performs bottom-up structure matching, based on the similarity of the leaf sets of pairs of elements. As in our work, the DIKE system [42] uses the distance of the nodes in the schemas to compute the mappings; while computing the similarity of a given pair of objects, other objects that are closely related to both count more heavily than those that are reachable only via long paths of relationships. Similar approaches, where closer entities in a given graph add more to the overall similarity than the far entities have also been used while mining web document associations [5, 6, 28] as well as for finding similarities between terms in a natural language [25, 43, 46, 47].

Although not directly related with finding mappings between elements in two given structures, recently there has been a large body of relevant work for efficient indexing and retrieval of tree-structured data [2, 10, 17, 26, 36, 45, 53, 55]. The DataGuide [17] is a structural summary of a database, and makes it possible to query XML documents based on their structure. The T-Index [36] on the other hand is a non-deterministic structure for both tree and graph databases. It is tailored to fit queries matching a given path template, thus not being an aid in subpath matching. The IndexFabric [10] indexes paths as well as the content of tree databases in a balanced hierarchy of Patricia tries. In general, most of these works focus on indexing and matching of paths and path-expressions on trees. As in our work, on the other hand, structural matching techniques between two labeled trees with potential "rename" mismatches and other distortions are used in [54, 56, 57]. These works use the tree edit-distance concept [29, 49] to measure how similar two trees are. A good survey of approaches to tree edit- and alignment-distance problems can be found in [1]. Unfortunately, the general undordered edit-distance problem has been shown to be NP-complete [56]. Certain special cases can be solved efficiently if appropriate local edit costs are available. [49, 56], for instance, provide postorder traversal based algorithms for calculating the editing distance between ordered, nodelabeled trees. [57] extends this works to connected, undirected, acyclic, graphs where only edges are labeled. It first shows that the problem is, as expected, NPhard and then it provides an algorithm for computing the edit distance between graphs where each node has at most two neighbors. Chawhate et al. [8, 9] provide alternative, and more flexible, algorithms to calculate the edit distance between ordered nodelabeled trees. Other research in tree similarity can be found in [16, 30, 40, 48]. Unlike our approach where the structural match between the nodes is captured wholisitically in the multidimensional space obtained through the MDS transformation, the edit-distance-based approaches associate explicit costs to each one of the local tree edit operations of deleting, inserting, and relabeling of the nodes and aim finding a minimum-cost sequence of such edit operations that would transform one of the input trees into the other. Thus, in addition to being costly in terms of execution time, these apporaches need appropriate local edit costs to function.

7 Conclusions

The functioning of an automated multimodal media integration system requires access to metadata that describe the individual media resources. The metadata are generally application-dependent. Therefore, an automated media integration mechanism needs to mine and relate the common and uncommon components. In this paper, we developed algorithms to automatically discover mappings in hierarchical media data, metadata, and ontologies. The proposed algorithm uses the structural information to mine and map the semantically similar but syntactically different terms and components. We extensively evaluated the performance of the algorithm for various parameters and showed that the algorithm is very effective in achieving a high degree of correct matches.

Acknowledgements This work has been supported by an NSF grant #ITR0326544 and a CEINT grant. This is an extended version of a work originally published at the Fifth International Workshop on Multimedia Data Mining [4].

References

1. Bille P (2003) A Tree edit distance, alignment distance and inclusion. IT University of Copenhagen, Technical Report Series, TR-2003-23
2. Bremer J, Gertz M (2003) An efficient XML node identification and indexing scheme. VLDB
3. Brickley D, Guha R (2000) Resource description framework (RDF) schema specification. http://www.w3.org/TR/RDF-schema
4. Candan KS, Kim JW, Liu H, Suvarna R (2004) Structure-based mining of hierarchical media data, meta-data, and ontologies. In: Proceedings of the 5th workshop on multimedia data mining in conjunction with the ACM conference on knowledge discovery & data mining, August 22–25. Seattle, WA, USA
5. Candan KS, Li WS (2000) Using random walks for mining web document associations. In: Proceedings of the Pacific-Asia conference on knowledge discovery and data mining (PAKDD), pp 294–305
6. Candan KS, Li WS (2001) Discovering web document associations for web site summarization. DaWaK 152–161
7. Candan KS, Li WS (2001) On similarity measures for multimedia database applications. Knowl Inf Syst 3(1):30–51
8. Chawathe S (1999) On the editing comparing hierarchical data in external memory. In: Proceedings of the 25th international conference on very large data bases. Edinburgh, Scotland, UK
9. Chawathe S, GarciaMolina H (1997) Meaningful change detection in structured data. In: Proceedings of the ACM SIGMOD international conference on management of data. Tucson, Arizona, pp 26–37

10. Cooper BF, Sample N, Franklin MJ, Hjaltason GR, Shadmon M (2001) A fast index for semistructured data. VLDB, pp 341–350
11. Doan A, Domingos P, Levy A (2000) Learning source descriptions for data integration. In: Proceedings of the WebDB workshop, pp 81–92
12. Document Object Model (DOM) (1997) http://www.w3.org/DOM/
13. Dublin Core Initiative and Metadata Element Set (1995) http://dublincore.org
14. Extensible 3D (X3D) Graphics (2000) http://www.web3d.org/x3d.html
15. Extensible Markup Language (XML) (2004) http://www.w3.org/TR/REC-xml
16. Farach M, Thorup M (1997) Sparse dynamic programming for evolutionarytree comparison. SIAM J Comput 26(1):210–223
17. Goldman R, Widom J (1997) Enabling query formulation and optimization in semistructured databases. VLDB, pp 436–445
18. Gower J (1975) Generalized procrustes analysis. Psychometrika 40:33–51
19. Guha RV, Bray T (1997) Meta content framework using XML. http://www.w3.org/TR/NOTE-MCF-XML-970624
20. Kendall DG (1984) Shape manifolds: procrustean metrics and complex projective spaces. Bull London Math Soc 16:81–121
21. Kruskal JB (1964) Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika 29(1):1–27
22. Kruskal JB (1964) Nonmetric multidimensional scaling: a numerical method. Psychometrika 29(2):115–129
23. Kruskal JB, Wish M (1978) Multidimensional scaling. Sage Publications, Beverly Hills
24. Lassila O (1997) Introduction to RDF metadata. http://www.w3.org/TR/NOTE-rdf-simple-intro
25. Lee J, Kim M, Lee Y (1993) Information retrieval based on conceptual distance in IS–A hierarchies. J Doc 49(2):188–207
26. Li Q, Moon B (2001) Indexing and querying XML data for regular path expressions, VLDB
27. Li W, Clifton C (1994) Semantic integration in heterogeneous databases using neural networks. In: Proceedings of the 20th international conference on very large data bases, pp 1–12
28. Li WS, Candan KS, Vu Q, Agrawal D (2002) Query relaxation by structure and semantics for retrieval of logical web documents. TKDE 14(4):768–791
29. Lu SY (1979) A tree-to-tree distance and its application to cluster analysis. IEEE Trans PAMI 1:219–224
30. Luccio F, Pagli L (1995) Approximate matching for two families of trees. Inf Comput 123(1):111–120
31. MacQueen J (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the 5th Berkeley symposium on mathematical Statistical Probability, vol 1, pp 281–297
32. Madhavan J, Bernstein PA, Rahm E (2001) Generic schema matching with cupid. In: Proceedings of the 27th international conference on very large data bases, pp 49–58
33. McHugh J, Abiteboul S, Goldman R, Quass D, Widom J (1997) Lore: a database management system for semistructured data. SIGMOD Rec 26(3):54–66
34. Miller R, Ioannidis Y, Ramakrishnan R (1994) Schema equivalence in heterogeneous systems: bridging theory and practice. Inf Syst 19(1):3–31
35. Miller RJ, Haas L, Hernandez MA (2000) Schema mapping as query discovery. In: Proceedings of the 26th international conference on very large data bases, pp 77–88
36. Milo T, Suciu D (1999) Index structures for path expressions. In: Proceedings of the ICDT'99. ICDT, pp 277–295
37. Milo T, Zohar S (1998) Using schema matching to simplify heterogeneous data translation. In: Proceedings of the conference on very large data bases, pp 122–133
38. Mitra P, Wiederhold G, Jannink J (1999) Semiautomatic integration of knowledge sources. In: Proceedings of Fusion'99. Sunnyvale, USA
39. Mitra P, Wiederhold G, Kersten M (2000) A graph oriented model for articulation of ontology interdependencies. In: Proceedings of the extending database technologies. Lecture Notes in Computer Science, vol 1777, pp 86–100
40. Myers E (1986) An O(ND) difference algorithms and its variations. Algorithmica 1(2):251–266

41. Namespaces in XML (1999) http://www.w3.org/TR/REC-xml-names
42. Palopoli L, Sacca D, Ursino D (1998) An automatic technique for detecting type conflicts in database schemas. In: Proceedings of the 7th international conference on information and knowledge management (CIKM), pp 306–313
43. Rada R, Mili H, Bicknell E, Blettner M (1989) Development and application of a metric on semantic nets. IEEE Trans Syst, Manage Cybern 19(1):17–30
44. Rahm E, Bernstein PA (2001) A survey of approaches to automatic schema matching. VLDB J 10:334–350
45. Rao P, Moon B (2004) PRIX: indexing and querying XML using Prufer sequences, ICDE
46. Resnik P (1995) Using information content to evaluate semantic similarity in a taxanomy. IJCAI, pp 448–453
47. Resnik P (1999) Sematic similarity in a taxanomy: an information-based measure and its application to problems of ambiguity in natural language. J Artif Intell Res 11:95–130
48. Selkow S (1977) The tree to tree editing problem. Inf Process Lett 6(6):184–186
49. Tai KC (1979) The tree-to-tree correction problem. J ACM 36:422–433
50. The Moving Picture Experts Group (MPEG) (2001) homepage http://www.chiariglione.org/mpeg/
51. Torgerson WS (1952) Multidimensional scaling. I. Theory and method. Psycometrika 17:401–419
52. University of Pennsylvania TreeBank Project collection at http://www.cs.washington.edu/research/xmldatasets/www/repository.html
53. Wang H, Park S, Fan W, Yu P (2003) ViST: a dynamic index method for querying XML data by tree structures. SIGMOD
54. Wang J, Zhang K, Jeong K, Shasha D (1994) A system for approximate tree matching. IEEE TKDE, pp 559–571
55. Zhang C, Naughton JF, DeWitt DJ, Luo Q, Lohman GM (2001) On supporting containment queries in relational database management
56. Zhang K (1989) The editing distance between trees: algorithms and applications. PhD Thesis, Courant Institute, Department of Computer Science
57. Zhang K, Shasha D (1989) Simple fast algorithms for the editing distance between trees and related problems. SIAM J Comput 18:1245–1262
58. Zhang K, Shasha D (1997) Approximate tree pattern matching. In: Apostolico A, Galil Z (eds) Pattern matching in strings, trees, and arrays. Oxford University, Oxford, pp 341–371
59. Zhang K, Wang JTL, Shasha D (1996) On the editing distance between undirected acyclic graphs. Int J Comput Sci 7(1):43–57

Author Biographies

K. Selçuk Candan is an Associate Professor at the Department of Computer Science and Engineering at the Arizona State University. He joined the department in August 1997, after receiving his Ph.D. from the Computer Science Department at the University of Maryland at College Park. He received the 1997 ACM DC Chapter award of Samuel N. Alexander Fellowship for his Ph.D. work. His research interests include development of indexing and retrieval schemes for multimedia and Web information and management of dynamic, heterogeneous, and distributed data. He has published various articles in respected journals and conferences in these areas. He also served as program committee member, chair person, and guest editor in various workshops, conferences, and journals. He received his B.S. degree, first ranked in the department, in computer science from Bilkent University in Turkey in 1993. `http://www.public.asu.edu/~candan`.

Jong Wook Kim received his B.S. from Korea University, Seoul, Korea in 1998, his M.S. from KAIST, Daejon, Korea, in 2000. He is currently a Ph.D. student at the Department of Computer Science and Engineering, Arizona State University, AZ, USA. His primary research interests are web data mining, information retrieval and database systems. His current research concentrates on mining in web communities like discussion board.

Huan Liu earned his Ph.D. in Computer Science in 1989 at University of Southern California, and Bachelor of Engineering in the Electrical Engineering and Computer Science Department at Shanghai Jiao Tong University in 1983. He conducted research at Telecom (Telstra) Australia Research Laboratories in Melbourne, Australia. In January 1994, he joined the School of Computing at the National University of Singapore, and became an Associate Professor. Since January 2000, he is with Department of Computer Science and Engineering at Arizona State University as an Associate Professor. He is a senior member of IEEE, member of ACM, and AAAI. His principal research interests include machine learning, feature and subset selection, data preprocessing, bioinformatics, and data (including text and web) mining. He has worked on real-world data mining applications and published extensively in journal and conference papers, book chapters, and books. He serves on the editorial board of journals, handbook of data mining, encyclopedia of data mining and warehousing.

Reshma Suvarna is currently employed at Honeywell as a Senior Software Engineer. Her work at Honeywell is in the area of Aerospace Electronic Systems. She recieved her Masters Degree in Computer Science and Engineering from Arizona State University in 2003. In addition to her current work in the Aerospace Electronic Systems, she is interested in data mining, web mining, and software engineering research.