Paper title:

ON THE USAGE OF STRUCTURAL DISTANCE METRICS FOR MINING HIERARCHICAL STRUCTURES

Authors:

1) Theodore Dalamagas
(dalamag@dblab.ece.ntua.gr)
School of Electrical and Computer Engineering
National Techn. Univ. of Athens
Greece

2) Tao Cheng
(tcheng3@cs.uiuc.edu)
Department of Computer Science
University of Illinois at Urbana-Champaign
USA

3) Timos Sellis
(timos@dblab.ece.ntua.gr)
School of Electrical and Computer Engineering
National Techn. Univ. of Athens
Greece

ON THE USAGE OF STRUCTURAL DISTANCE METRICS FOR MINING HIERARCHICAL STRUCTURES

Abstract. The recent proliferation of XML-based standards and technologies demonstrates the need for effective management of hierarchical structures. Such structures are used, for example, to organize data in product catalogs, taxonomies of thematic categories, concept hierarchies, etc. Since the XML language has become the standard data exchange format on the Web, organizing data in hierarchical structures has been vastly established. Even if data is not stored natively in such structures, export mechanisms make data publicly available in hierarchical structures to enable its automatic processing by programs, scripts, and agents. Processing data encoded in hierarchical structures has been a popular research issue, resulting in the design of effective query languages. However, the inherent structural aspect of such encodings has not received strong attention till lately, when the requirement for mining tasks, like clustering/classification methods, similarity ranking, etc., on hierarchical structures has raised. The key point to perform such tasks is the design of a structural distance metric to quantify the structural similarity between hierarchical structures. The chapter will study distance metrics that capture the structural similarity between hierarchical structures, and approaches that exploit structural distance metrics to perform mining tasks on hierarchical structures.

1 INTRODUCTION

Hierarchical structures have been extensibly used in the past in the form of tree or graph structures to organize data in thematic categories, taxonomies, catalogs, SGML files, etc. Since the XML language is becoming the standard data exchange format on the Web, the idea of organizing data in hierarchical structures to enable its automatic processing by programs, web scripts, agents, has been re-visited. Vast amount of data from many knowledge domains is available or processed to be available on the Web, encoded in hierarchical structures under the XML format.

While the processing and management of data encoded in such structures have been extensivelly studied (Abiteboul, Buneman, & Suciu, 2000), operations based on the structural aspect of such an encoding have received strong attention only lately. Structural distance metrics is a key issue for such operations. A structural distance metric can quantify the structural similarity between hierarchical structures. Thus, it is a tool that can support mining tasks for such structures. Examples of these tasks are clustering methods, classification methods and similarity ranking. For instance, a clustering task can identify sets of structures such that each set includes similar (in terms of their form or the way that they organize data) structures. A similarity ranking mechanism can detect hierarchical structures which are similar to a hierarchical structure given as a test pattern, and also quantify such a similarity.

In this chapter we will first study distance metrics that capture the structural similarity between hierarchical structures. Then we will present approaches that exploit structural distance metrics to perform mining tasks on hierarchical structures. We concentrate on hierarchical structures encoded as XML documents, due to the proliferation of the XML language for encoding data on the Web.

The chapter is organized as follows. Section 2 gives examples of mining tasks for hierarchical structures and discusses background issues. Section 3 presents various ways of defining and calculating structural distance metrics. Section 4 discusses approaches that exploit structural distance metrics to perform mining tasks for hierarchical structures, and, finally, Section 5 concludes the chapter and discusses future perspectives.

2 BACKGROUND

Vast amount of data from various knowledge domains has been encoded in hierarchical structures and become available on the Web. This is due to the enormous growth of the Web and the establishment of the XML language as the standard data exchange format. The XML language provides a simple syntax for data that is human-readable and machine-readable, and its model is perfectly suited for organizing data in hierarchical structures (see next subsection). Mining these structures is a useful task for many domains, as the examples in the following paragraphs show.

Spatial data is often organized in data model catalogs expressed in hierarchical structures. For instance, areas that include forests with lakes, rivers and farms can be represented as tree-like structures. Clustering by structure is a mining task that can identify spatial entities with similar structure, e.g. entities with areas that include forests with lakes. For example, in Table 1, areas encoded by D_1 and D_2 are structurally similar, since D_2 only misses the *river* element. On the other hand, area encoded by D_3 is organized in a different way than D_1 and D_2. Examples on using XML representation for organizing geographical data in hierarchical structures are presented in (Wilson et al., 2003).

Bioinformatics is another application area where mining hierarchical structures can be applied. The main concern in this domain is the discovery of structurally similar macromolecular tree patterns. The detection of homologous protein structures (i.e. sets of protein structures sharing a similar structure) is such an example (Sankoff & Kruskal, 1999). Nowadays, such structures have been also encoded as XML documents. Such encodings, as well as general hierarchical encodings for life sciences are presented in (Direen & Jones, 2003).

But even for the XML data management itself, mining hierarchical structures is an important issue. XML documents can optionally have a Document Type Descriptor (DTD). A DTD serves as a grammar for an XML document, determining its internal structure and enabling exchange of documents through common vocabulary and standards. However, many XML documents are

(D1)	(D2)	(D3)
---	---	---
<?xml version="1.0"?>	<?xml version="1.0"?>	<?xml version="1.0"?>
<area type="rectangle"	<area type="rectangle"	<area type="rectangle"
x1="100" y1="200">	x1="130" y1="210">	x1="300" y1="500">
<forest type="rectangle"	<forest type="rectangle"	<forest type="rectangle"
x1="20" x2="20">	x1="30" x2="10">	x1="50" x2="70">
<lake type="circle"	<lake type="circle"	<lake type="circle"
x1="5" y1="10" r1="5">	x1="2" y1="15" r1="10">	x1="35" y1="14" r1="30">
The lake	The lake	The lake
</lake>	</lake>	</lake>
<farm type="rectangle">	<farm type="rectangle">	<river type="line">
The farm	The farm	The river
</farm>	</farm>	</river>
</forest>	</forest>	</forest>
<river type=line>	</area>	<farm type="rectangle">
The river		The farm
</river>		</farm>
</area>		</area>

Table 1. Examples of spatial information hierarchically structured in the form of XML documents.

constructed massively from data sources like RDBMSs, flat files, etc, without DTDs. XTRACT (Garofalakis, Gionis, Rastogi, Seshadri, & Shim, 2000; Garofalakis, Gionis, Rastogi, Seshadri, & Shim, 2003) and IBM AlphaWorks DDbE[1] are DTD discovery tools that automatically extract DTDs from XML documents. Such tools fail to discover meaningful DTDs in case of diverse

[1] http://www.alphaworks.ibm.com/tech/DDbE

XML document collections (Garofalakis et al, 2003). Consider for example news articles from portals, newspaper sites, news agency sites, etc., hierarchically structured in the form of XML documents. Such documents, although related, may have such a different structure that one cannot define a meaningful DTD for all of them. See for example the four XML documents in Table 2. A unique DTD for these documents should define an element, which might be either *article* or *news_story*. This element should contain a *title* element and, then, either an *author* or a *picture* element. However *picture* in D_3 doc is identical to *image* in *D1*. Also, *picture* in D_3 comes before the *author* element, while the equivalent *image* in D_1 comes after the *author* element. Such irregularities make the construction of a unique, meaningful DTD a hard task. For this reason, identifying groups of XML documents that share a similar structure is crucial for DTD discovery systems. If a collection of XML documents that encode hierarchical data is first grouped into sets of structurally similar documents, then a meaningful DTD can be assigned to each set individually. For example, the XML documents in Table 2 can be grouped in two sets: the first set includes documents D_1 and D_2, and the second one includes documents D_3 and D_4.

(D1)	(D2)	(D3)	(D4)
---	---	---	---
<?xml version="1.0"?>	<?xml version="1.0"?>	<?xml version="1.0"?>	<?xml version="1.0"?>
<article>	<article>	<news_story>	<news_story>
<title>…</title>	<title>…</title>	<title>…</title>	<title>…</title>
<author>	<author>	<picture>…</picture>	<author>
<first_name>…</first_name>	<first_name>…</first_name>	<author>	<name>…</name>
<last_name>…</last_name>	<last_name>…</last_name>	<name>…</name>	</author>
</author>	</author>	</author>	<summary>…</summary>
<image>…</image>	<summary>…</summary>	<summary>…</summary>	<body>…</body>
<summary>…</summary>	<main>…</main>	<body>…</body>	</news_story>
<main>…</main>	</article>	</news_story>	
</article>			

Table 2. Examples of news articles hierarchically structured in the form of XML documents.

Documents in each set are structurally similar. For example, D_2 misses only the element *image* (inside *article*), compared to D_1. On the other hand, D_1 and D_3 are not structurally similar: *picture* in D_3 comes before the *author* element, while the equivalent *image* in D_1 comes after the *author* element.

We next show how hierarchical structures are used to organize data on the Web.

2.1 Semistructured data

In the context of the Web, the notion of *semistructured data* has been introduced to capture schemaless, self-describing and irregular data. The term *semistructured* indicates that there is not a clear distinction between the organization of data (i.e. their structure) and data itself. To capture this characteristic, models for semistructured data have been introduced (Abiteboul et al., 2000). Such models are simple, flexible and describe data and as one entity with graph-based or tree-based hierarchical structures. The *object exchange model* (*OEM*) and the *XML data model* are examples of semistructured data models.

The object exchange model (OEM) is a graph representation of a collection of objects, introduced in the TSIMMIS project (Garcia-Molina et al., 1997). Every OEM object has an identifier and a value, atomic or complex. An atomic value is an integer, real, string or any other data, while a complex value is a set of oids, each linked to the parent node using a textual label. Objects with atomic values are called atomic objects and objects with complex values are called complex objects. The XML data model is another graph representation of a collection of atomic and complex objects. However, while the OEM model denotes graphs with labels on edges, the XML data model denotes graphs with labels on nodes. The XML data model provides a mechanism to

define references that are unique. Using references, one can refer to an element using its identifier.

Figure 1(a) presents an example of an OEM database with six complex objects (i.e. objects 1, 2, 3, 4, 5, and 6) and seven atomic objects (i.e. 7, 8, 9, 10, 11, 12, 13). In Figure 1(b), the example of Figure 1(a) is re-expressed using the XML data model. Without references (e.g. like the thick line in Figure 1(b)) the XML data model becomes a rooted ordered labeled tree.

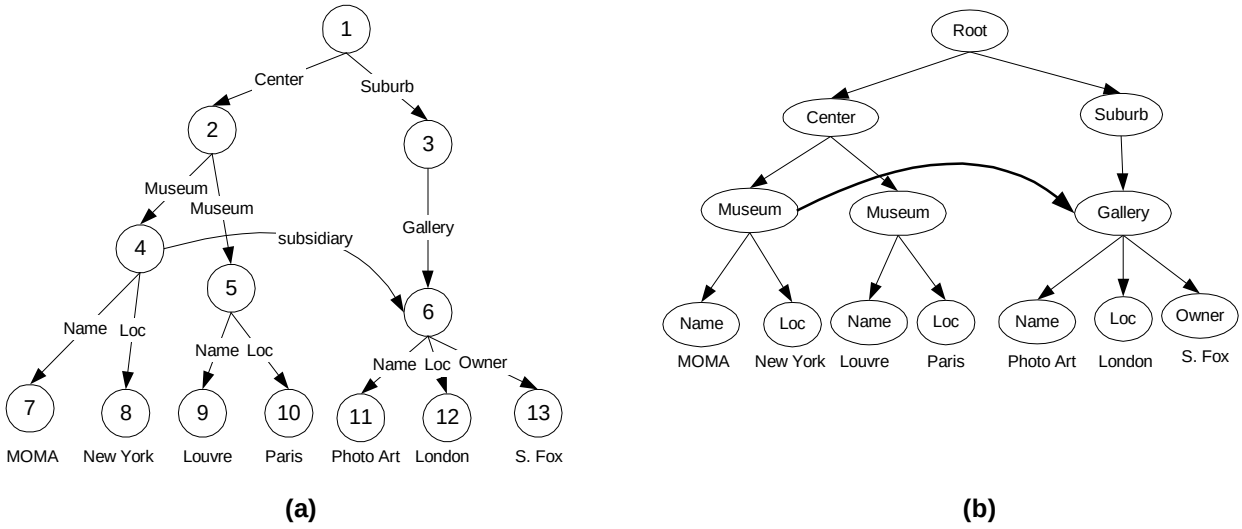

Figure 1. (a) OEM example, (b) XML data model example.

3 STRUCTURAL DISTANCE METRICS

A structural distance metric quantifies the structural similarity between hierarchical structures. Since such structures are actually tree or graph structures, a popular way to define distances is by using tree/graph edit sequences or graph matching techniques. For example, the minimum number of operations needed to transform a tree to another one is an indication of how similar these trees are. Or the number of common edges between two graphs is an indication of how similar these graphs are.

Distance metrics, in general, fulfill certain algebraic properties compared to distance or similarity measures:

1. $d(A,B) \geq 0$ (positivity)

2. $d(A,B) = d(B,A)$ (symmetry)

3. $d(A,B) + d(B,C) \leq d(A,C)$ (triangular equation)

Such properties are considered important, since the metrics are exploited in mining tasks like clustering and classification, where the quality of those metrics clearly affects the quality of the results obtained. To this extent, we do not consider measures used in change detection methods (see (Cobena, Abiteboul, & Marian, 2002) for a comparative study). These methods can detect sets of edit operations with cost close to the minimal with significantly reduced computation time. However, minimality is important for the quality of any measure to be used as a distance metric.

In the next two subsections, we discuss in detail methods to define and calculate structural distance metrics based on tree edit distances and on graph/subgraph isomorphism.

3.1 Structural Distances based on Tree Edit Distances

This subsection introduces tree edit distances and shows how they can be used to define and calculate structural distance metrics.

3.1.1 Background

The notion of tree edit operations is central in all approaches that utilize tree edit distances to define structural distance metrics. An *atomic tree edit operation* on a tree structure is an operation that manipulates a node as a single entity, for example deletion, insertion, replacement

of a node. A *complex tree edit operation* is a set of atomic tree edit operations, treated as one single operation. An example of a complex tree edit operation is the insertion of a whole tree as a subtree in another tree, which is actually a sequence of atomic node insertion operations. We next present variations of tree edit operations.

1. insert node
    a. Variation I ($Ins^{l}(x,y,i)$): In this variation, a new node x is inserted as the i_{th} child of node y. All children of y should be leaf nodes.
    b. Variation II ($Ins(x,y,i)$): The restriction that new nodes are inserted only as a leaf node is relaxed. A new node x which is inserted as the i_{th} child of node y takes a subsequence of the children of y as its own children. Thus, given p, node y will have $y_1 \ldots y_j, x, y_{p+1}, \ldots y_n$ as children and x will have $y_{j+1}, y_{j+2}, \ldots y_p$ as children.
2. delete node
    a. Variation I ($Del(y)$): Deletion can be applied to any node. The children of the deleted node become the children of its parent.
    b. Variation II ($Del^{l}(y)$): Only leaf nodes can be deleted.
3. replace node ($Rep(x,y)$): Node y replaces node x.
4. move subtree ($Mov(x,y,k)$): The subtree rooted at node x is moved to become the k_{th} child of node y.

A *tree edit sequence* is a sequence of tree edit operations that transforms T_1 to T_2. Assuming a cost model to assign costs for tree edit operations, the *tree edit distance* between T_1 and T_2, is the minimum cost between the costs of all possible tree edit sequences that transform T_1 to T_2. Based

on the notion of tree edit distance, we can now define the structural distance between tree structures.

Definition 3.1.1. *Let T1 and T2 be two tree structures, $D(T_1,T_2)$ be their tree edit distance and $D'(T_1,T_2)$ the cost to delete all nodes from T_1 and insert all nodes from T_2. The structural distance S between T_1 to T_2 is defined as $S(T_1,T_2)=D(T_1,T_2) / D'(T_1,T_2)$.*

The structural distance is (a) *0* when the trees have exactly the same structure and the same labels in their matching nodes, (b) *1* when the trees have totally different structure and not even two pairs of matching nodes with the same ancestor/descendant relationship, (c) low when the trees have similar structure and high percentage of matching nodes, and (d) high when the trees have different structure and low percentage of matching nodes. Figure 2 presents the sequence of tree edit operations to transform tree T_1 to T_2 with minimum cost, assuming unit cost for every tree edit operations. In this example, $D'(T_1,T_2)=12$, since five nodes must be deleted from T_1 and seven nodes must be inserted from T_2. Also, $D(T_1,T_2)=5$, since five, unit-cost operations are needed to transform tree T_1 to T_2.

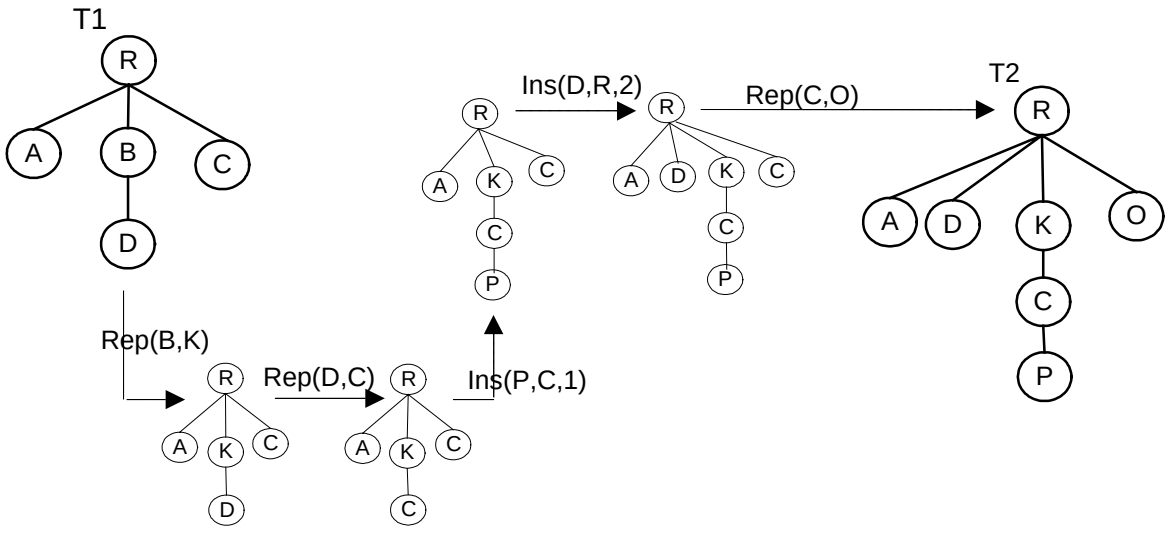

Figure 2. An example of tree edit sequence.

The next subsubsection gives a detailed survey of tree edit techniques that can be used to define structural distance metrics.

3.1.2 Related Techniques and Algorithms

Selkow in (Selkow, 1977) suggests a recursive algorithm to calculate the tree edit distance between two rooted ordered labeled trees. An *insert node* or *delete node* operation is permitted only at leaf nodes. Any node can be updated using the *replace node* operation. So, the set of permitted tree edit operations is $\{Ins^{l}(x,y,i), Del^{l}(y), Rep(x,y)\}$. The cost to delete a whole subtree rooted at node y is denoted by $W_d(y)$, and is the sum of the costs spent to delete all nodes of the subtree, starting from the leaves. Similarly, $W_i(x)$ denotes the cost to insert a whole subtree.

The algorithm to compute the edit distance D between the two trees T_1 and T_2 calculates recursively the distance between their subtrees. The idea of the main recursion is that the calculation of the distance between two (sub)trees t_1 and t_2 requires the calculation of four distances: (a) t_1 without its last subtree and t_2, (b) t_1 and t_2 without its last subtree, (c) t_1 without its last subtree and t_2 without its last subtree, and (d) last subtree of t_1 and last subtree of t_2. If M and N are the total number of nodes for T_1 and T_2, respectively, then the complexity of the algorithm is exponential (4min(N,M)).

Let r be the root of current subtree t_1 of T_1, k the number of subtrees in r, and i the last node of last subtree of t_1 ($i=i_k$) according to the preorder sequence. Similarly, let s be the root of current subtree t_2 of T_2, l the number of subtrees in s, and j the last node of last subtree of t_2 ($j=j_l$). $D(r,i:s,j)$ denotes the tree edit distance between t_1 and t_2. Analytically, the algorithm proceeds as follows:

1. if (r==i) and (s==j) then $D=0$:

If t_1 and t_2 consist only of one node each (their roots), then the cost to transform t_1 to t_2 is equal to *0* (roots are the same).

2. if (s==j) then $D=W_d(i_{k-1}+1) + D(r,i_{k-1}:s,j)$:

If t_2 consists only of one node then the cost to transform t_1 to t_2 is equal to the cost to delete the k_{th} subtree of t_1 (which is the last subtree of the root of t_1), plus the cost to transform $t_{1'}$ (which is t_1 without its k_{th} subtree) to t_2.

3. if (r==i) then $D=W_i(j_{l-1}+1) + D(r,i:s,j_{l-1})$:

If t_1 consists only of one node then the cost to transform t_1 to t_2 is equal to the cost to insert the l_{th} subtree of t_2 (which is the last subtree of the root of t_2) in t_1 plus the cost to transform t_1 to $t_{2'}$ (which is t_2 without its l_{th} subtree).

4. In any other case find the minimum between the following three costs, $D=\min(d_1,d_2,d_3)$:

   a. $d_1=W_d(i_{k-1}+1) + D(r,i_{k-1}:s,j)$:

   the cost to delete the k_{th} subtree of t_1 (which is the last subtree of the root of t_1) plus the cost to transform $t_{1'}$ (which is t_1 without its k_{th} subtree) to t_2.

   b. $d_2=Wi(j_{l-1}+1) + D(r,i:s,j_{l-1})$:

   the cost to insert the l_{th} subtree of t_2 (which is the last subtree of the root of t_2) in t_1 plus the cost to transform t_1 to $t_{2'}$ (which is t_2 without its l_{th} subtree).

   c. $d_3=D(r,i_{k-1}:s,j_{l-1}) + c_r(i_{k-1}+1,j_{l-1}+1) + D(i_{k-1}+1,i_k:j_{l-1}+1,j_l)$:

   the cost to transform $t_{1'}$ (which is t_1 without its k_{th} subtree) to $t_{2'}$ (which is t_2 without its l_{th} subtree), plus the cost (i.e. c_r) to replace the root of the k_{th} subtree of t_1 with the root of the l_{th} subtree of t_2, plus the cost to transform the k_{th} subtree of t_1 to the l_{th} subtree of t_2.

Zhang and Shasha (1989) suggests a recursive algorithm to calculate the tree edit distance between two rooted ordered labeled trees, permitting tree edit operations anywhere in the trees. So, the set of permitted tree edit operations is {*Ins*(*x*,*y*,*i*), *Del*(*y*), *Rep*(*x*,*y*)}. The algorithm is based on the notion of mappings. A *tree mapping M* between two trees T_1 and T_2 is an one-to-one relationship between nodes of T_1 and nodes of T_2. A mapping *M* includes a set of pairs (*i*,*j*). For any two pairs (i_1,j_1) and (i_2,j_2) in *M*: (a) $i_1=i_2$ iff $j_1=j_2$, (b) $t_1[i_1]$ is to the left of $t_1[i_2]$ iff $t_2[j_1]$ is to the left of $t_2[j_2]$, and (c) $t_1[i_1]$ is an ancestor of $t_1[i_2]$ iff $t_2[j_1]$ is an ancestor of $t_2[j_2]$ ($t[i]$ refers to the node *i* of *T*, according to the postorder sequence). Every mapping *M* corresponds to a

sequence of edit operations. Nodes in T_1 which are untouched by M correspond to $Del(y)$ operations in T_1. Nodes in T_2 which are untouched by M correspond to $Ins(x,y,i)$ operations in T_1. Nodes in T_1 related to nodes in T_2 by M correspond to $Rep(x,y)$ operations.

The algorithm calculates the minimum cost between the costs of the sequences of edit operations that transform a tree T_1 to the tree T_2, produced by all possible valid mappings on T_1 and T_2. Let $D(T_1[i':i], T_2[j':j])$ be the distance between trees $T_1[i':i]$ and $T_2[j':j]$ (a tree T is denoted as $T[i:j]$, where i is the label of its root and j is the label of its rightmost leaf, according to the postorder sequence). We note that $t[i]$ refers to the node i of T, and $l[i]$ refers to the postorder number of the leftmost leaf of the subtree rooted at $t[i]$, according to the postorder sequence. Then:

1. $D(0, 0) = 0$ (one-node trees, roots are labeled as 0)
2. $D(T_1[l(i_1):i],0) = D(T_1[l(i_1):i-1],0) + cost\_to\_delete\_node(t_1[i])$
3. $D(0,T_2[l(j_1):j]) = D(0,T_2[l(j_1):j-1]) + cost\_to\_insert\_node(t_2[j])$
4. $D(T_1[l(i_1):i],T_2[l(j_1):j]) = \min(d_1,d_2,d_3)$:
    a. $d_1=D(T_1[l(i_1):i-1],T_2[l(j_1):j]) + cost\_to\_delete\_node(t_1[i])$
    b. $d_2=D(T_1[l(i_1):i],T_2[l(j_1):j-1]) + cost\_to\_insert\_node(t_2[j])$
    c. $d_3=D(T_1[l(i_1):l(i)-1],T_2[l(j_1):l(j)-1]) + D(T_1[l(i):i-1],T_2[l(j):j-1]) +$ $cost\_to\_replace\_node(t_1[i],t_2[j])$ (where i and j are descendants of $t_1[i_1]$ and $t_2[j_1]$, respectively)

The recursion is similar to the one in Selkow's algorithm presented before. However, deletions and insertions are permitted anywhere in the tree. If M and N are the total number of nodes for T_1 and T_2 and b and d are their depths, respectively, then the complexity of the algorithm is $O(MNbd)$.

Chawathe, Rajaraman, Garcia-Molina, and Widom (1996) and Chawathe (1999) suggest algorithms to calculate tree edit distances. An *insert node* ($Ins^l(x,y,i)$) or *delete node* ($Del^l(y)$) operation is permitted only at leaf nodes. Any node can be updated using the *replace node*

(*Rep*(*x*,*y*)) operation. In (Chawathe et al., 1996), a *move subtree* (*Mov*(*x*,*y*,*k*)) operation is also available. However, the distance calculation needs a predefined set of matching nodes between the trees. On the other hand, the distance calculation in (Chawathe, 1999) is based on shortest path detection on an *edit graph*. An edit graph can represent tree edit sequences. The edit graph for two trees T_1 and T_2 is an ($M+1$)x($N+1$) grid of nodes, having a node at each (x,y) location, x in [$0\ldots$ ($M+1$)] and y in [$0\ldots$ ($N+1$)]. Directed lines connect the nodes. A horizontal line (($x-1$,y), (x,y)) denotes deletion of $T_1[x]$, where $T_1[x]$ refers to the x_{th} node of T_1 in its preorder sequence. A vertical line ((x,$y-1$), (x,y)) denotes insertion of $T_2[y]$, where $T_2[x]$ refers to the x_{th} node of T_2 in its preorder sequence. Finally, a diagonal line (($x-1$,$y-1$), (x,y)) denotes update of $T_1[x]$ by $T_2[y]$.

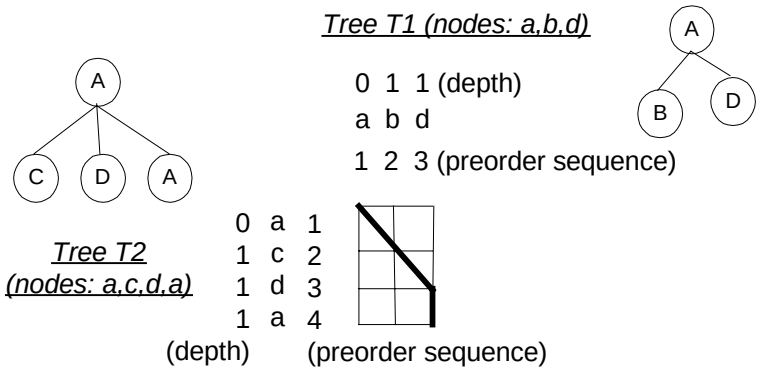

Figure 3. An example of an edit graph.

Figure 3 shows an example of an edit graph which represents an edit sequence to transform tree T_1 to tree T_2. Notice that T_1 becomes T_2 by (*Rep*(T_1[*2*],*c*), *Rep*(T_1[*3*],*d*), *Ins*(T_2[*4*],T_1[*1*],*3*). Every edit sequence that transforms T_1 to T_2 can be mapped to a path in an edit graph. The tree edit distance between two rooted ordered labeled trees is the shortest of all paths to which edit sequences are mapped in an edit graph. An edit graph G is constructed as a ($M+1$)x($N+1$)x3 matrix, whose cells contain the cost of the corresponding edit operation. The third dimension is

used to determine the direction of the line drawing, that is the type of the operation, for example [*0*] for horizontal lines, i.e. *delete node*, [*1*] for vertical lines, i.e. *insert node*, and [*2*] for diagonal lines, i.e. *replace node*. If a line is missing from the edit graph, the corresponding cell contains the infinite value. For example $G[4][6][0]= \infty$ means that there is no horizontal line from node *4* to node *6* in the edit graph.

Consider any path that connects the node (*0,0*) to node *n* (*x,y*) in an edit graph. Node *n* (*x,y*) is the last node in the path. The distance *D* of *n* from (*0,0*) cannot be greater than that distance of its left node plus the cost of the line connecting that node to *n*. Similarly, *D* can neither be greater than that distance of *n*'s top node plus the cost of the line connecting that node to *n*, nor greater than that distance of *n*'s diagonal node plus the cost of the line connecting that node to *n*. Based on the above remarks, the following recurrence calculates the shortest path *D*[*x,y*] from (*0,0*) to (*x,y*) in an edit graph *G*:

$$D(x,y)=\min(m_1,m_2,m_3)$$

where

1. m_1 = D[x-1,y-1] + cost_to_replace_node(T_1[x], T_2[y]), if ((x-1,y-1),(x,y)) in G (the distance of (*x,y*)'s diagonal node in *G* plus the cost to replace $T_1[x]$ with $T_2[y]$), or ∞ otherwise,
2. m_2 = D[x-1,y] + cost_to_delete_node(T_1[x]), if ((x-1,y),(x,y) in G (the distance of (*x,y*)'s left node in *G* plus the cost to delete $T_1[x]$), or ∞ otherwise,
3. m_3 = D[x,y-1] + cost_to_insert_node(T_2[y]), if ((x,y-1),(x,y) in G (the distance of (*x,y*)'s top node in *G* plus the cost to insert $T_2[y]$), or ∞ otherwise.

In the algorithm, *D*[*i,j*] keeps the tree edit distance between tree T_1 with only its *i* nodes, assuming preorder traversal, and tree T_2 with only its *j* nodes assuming preorder traversal. If *M*

and *N* are the dimensions of the matrix that represents the edit graph, then the complexity of the algorithm is *O(MN)*.

3.1.3 Overview

All of the algorithms for calculating the edit distance for tree structures are based on dynamic programming techniques related to the string-to-string correction problem (Wagnera & Fisher, 1974). The key issue of these techniques is the detection of the set of tree edit operations which transform a tree to another one with the minimum cost (assuming a cost model to assign costs for every tree edit operation). Selkow's algorithm (Selkow, 1977) allows insertion and deletion only at leaf nodes, and relabel at every node. Its main recursion leads to increased complexity. Chawathe's (II) algorithm (Chawathe, 1999) allows insertion and deletion only at leaf nodes, and relabel at every node, too. It is based on the model of edit graphs which reduces the number of recurrences needed, compared to Selkow's. This algorithm is the only one that has been extended to efficiently calculate distances in external memory in case that tree sizes are prohibitively large, as presented in (Chawathe, 1999). Chawathe's (I) algorithm (Chawathe et al., 1996) is based on a different set of tree edit operations than Chawathe's (II). It allows insertion and deletion only at leaf nodes. Its main characteristic is the need of a pre-defined set of matching nodes between the trees. This set acts like a seed for the algorithm. Zhang's algorithm (Zhang & Shasha,1989) permits operations anywhere in the tree and uses a similar recurrence as Selkow's algorithm (Selkow, 1977).

We note that permitting insertion and deletion only at leaves prevents the destruction of membership restrictions of tree structures. For example, in the case of a deletion, we can avoid deleting a node and moving its children up one level. Actually, the deletion of an internal node

requires deletions of all nodes in its path, starting from the leaf node and going up to the internal node, a task which is assigned a high cost due to the deletion of all these nodes.

Finally, we note that tree edit distances can satisfy positivity, symmetry and triangular equation, and therefore they can be metrics under certain conditions, for example having equal weights for all allowed tree edit operations.

3.2 Structural Distances based on Graph Matching

This subsection discusses how graph and subgraph isomorphism can be used to define structural distance metrics.

3.2.1 Background

An alternative to view hierarchical structures is to regard them as graphs. Therefore, we could leverage the existing powerful graph matching algorithms. Graph isomorphism (Read & Corneil, 1977), subgraph isomorphism (Ullman, 1976) and maximum common subgraph (Bunke, Jiang, & Kandel, 1983; Levi, 1972) are basic concepts for graph matching. In (Bunke & Shearer, 1998), a graph similarity measure based on the maximum common subgraph of two graphs is proposed. A relatively new concept, minimum common supergraph is introduced in (Bunke, Jiang, & Kandel, 2000), together with a graph similarity measure based on it. In (Fernandez & Valiente, 2001), another approach comes up with a graph distance metric by combining both maximum common subgraph and minimum common supergraph. In this subsection, we will mainly focus on these three methods (Bunke & Shearer, 1998; Bunke et al., 2000; Fernandez & Valiente, 2001).
Another way to measure the similarity of two graphs is by defining graph edit distances. A graph edit distance is a generalization of tree edit distance. Algorithms for graph edit distances and related similarity measures have been discussed in (Tsai & Fu, 1979; Shapiro & Haralick, 1981;

Sanfeliu & Fu, 1983; Bunke et al., 1983). An advantage of distance metrics based on graph matching over edit distances is the independency on edit costs. Using those new distance metrics we can avoid the intensive calculations needed for obtaining edit costs.

3.2.2 Related Techniques and Algorithms

For the ease of introducing the three methods for deriving distance measures, we first give some basic graph definitions (we use the definitions from (Fernandez & Valiente, 2001)).

Definition 3.2.1. *Let L be a finite alphabet of vertex and edge labels. A graph is a tuple $G = (V, E, \alpha)$, where V is the finite set of vertices, $E \subseteq V \times V$ is the finite set of edges, $\alpha : V \cup E \to L$ is the vertex and edge labeling function.*

Let $|G| = |V| + |E|$ denote the size of G. The empty graph such that $|G| = 0$ will be denoted by ϕ.

Definition 3.2.2. *A graph $G_1 = (V_1, E_1, \alpha_1)$ is a subgraph of a graph $G_2 = (V_2, E_2, \alpha_2)$, denoted by $G_1 \subseteq G_2$, if $V_1 \subseteq V_2$, $E_1 \subseteq E_2$, $\alpha_1(x) = \alpha_2(x)$ for all $x \in V_1 \cup E_1$. A graph G_2 is called a supergraph of G_1 if $G_1 \subseteq G_2$.*

Definition 3.2.3. *Let $G = (V, E, \alpha)$ be a graph, V_2 a set of vertices, and $f : V_1 \to V_2$ a bijective function, where $V_1 \subseteq V$ and $V_1 \cap V_2 = \phi$. A renaming of G by f, denoted by $f(G)$, is the graph $G_f = (V_f, E_f, \alpha_f)$ defined by $V_f = V_2 \cup (V \setminus V_1)$, $E_f = \{(f'(u), f'(v)) \mid (u, v) \in E\}$, $\alpha_f(f'(v)) = \alpha(v)$ for all $v \in V$, $\alpha_f(e) = \alpha(e)$ for all $e \in E$, where $f' : V \to V_f$ is defined by*

$$f'(v) = \begin{cases} f(v), & \text{if } v \in V_1 \\ v, & \text{otherwise} \end{cases}$$

Definition 3.2.4. *Two graphs* G_1 *and* G_2 *are isomorphic, denoted by* $G_1 \cong G_2$*, if there is a renaming* f *of* G_1 *such that* $f(G_1) = G_2$*, and in this case it is said that* $f : G_1 \to G_2$ *is a graph isomorphism.*

Definition 3.2.5. *A graph* $\hat{G}$ *is a common subgraph of two graphs* G_1 *and* G_2 *if there exist subgraphs* $\hat{G}_1 \subseteq G_1$ *and* $\hat{G}_2 \subseteq G_2$ *such that* $\hat{G} \cong \hat{G}_1 \cong \hat{G}_2$*. It is maximum if there is no other common subgraph of* G_1 *and* G_2 *larger than* $\hat{G}$*.*

Definition 3.2.6. *A graph* $\overset{\backprime}{G}$ *is a common supergraph of two graphs* G_1 *and* G_2 *if there exist graphs* $\overset{\backprime}{G}_1 \subseteq \overset{\backprime}{G}$ *and* $\overset{\backprime}{G}_2 \subseteq \overset{\backprime}{G}$ *such that* $\overset{\backprime}{G}_1 \cong G_1$ *and* $\overset{\backprime}{G}_2 \cong G_2$*. It is minimum if there is no other common supergraph of* G_1 *and* G_2 *smaller than* $\overset{\backprime}{G}$*.*

All the three graph distance measures we are going to show below are proved to satisfy positivity, symmetry and triangular equation, and therefore are metrics.

In (Bunke & Shearer, 1998) a graph distance based on maximum common subgraph is proposed. The distance of two non-empty graphs G_1 and G_2 is defined as

$$d(G_1, G_2) = 1 - \frac{|\hat{G}|}{\max(|G_1|, |G_2|)}, \tag{1}$$

where $\hat{G}$ is the maximum common subgraph of G_1 and G_2. Notice that given G_1 and G_2, their maximum common subgraph is not unique.

The relationship between graph edit distance, maximum common subgraph and the minimum common supergraph is discussed in (Bunke et al., 2000). It is observed that under certain assumptions of the cost function (used for graph edit distance), the graph edit distance and the size of both maximum common subgraph and minimum common supergraph are equivalent to

each other. More specifically, knowing one of them together with the cost function and the size of the two underlying graphs, the other can be immediately calculated. If the cost function is restricted such that the insertion and deletion of nodes are unit operations, we could rewrite equation (1), and have the following graph similarity measure based on minimum common supergraph:

$$d(G_1, G_2) = 1 - \frac{|G_1| + |G_2| - |\overset{\smile}{G}|}{\max(|G_1|, |G_2|)}, \quad (2)$$

where $\overset{\smile}{G}$ is the minimum common supergraph of G_1 and G_2. Notice that given G_1 and G_2, their minimum common supergraph is not unique.

In (Fernandez & Valiente, 2001), the authors further study the relationship between the maximum common subgraph and the minimum common supergraph of two graphs. Simple constructions allow obtaining the maximum common subgraph from the minimum common supergraph, and visa versa. The maximum common subgraph and the minimum common supergraph are combined into a new graph distance metric.

$$d(G_1, G_2) = |\overset{\smile}{G}| - |\hat{G}|, \quad (3)$$

where $\hat{G}$ is the maximum common subgraph of G_1 and G_2 and $\overset{\smile}{G}$ is the minimum common supergraph of G_1 and G_2.

3.2.3 Overview

In this subsection we examined how graph matching algorithms can be used to calculate structural distance metrics for hierarchical structures. More specifically, we investigated three techniques which use maximum common subgraph, minimum common supergraph, and a

combination of both. Maximum common subgraph tries to eliminate the superfluous structural information, whereas minimum common supergraph takes into account the missing structural information. Superfluous and missing structural information are both taken into consideration when the measure is based on both maximum common subgraph and minimum common supergraph. Graph matching methods to calculate structural distance metrics scale well compared to methods based on tree edit distances. The latter needs the calculation of tree edit sequences, which is a quite intensive task.

3.3 The Approach of IR Community

This subsection introduces approaches used in IR community to detect the similarity between hierarchical structures encoded as XML documents. Bitmap indexing (Chan & Ioannidis, 1998) has been introduced to improve performance of information retrieval. Similarity measures could be defined using bitmap indexes on hierarchical structures.

3.3.1 Related Techniques and Algorithms

In (Yoon, Raghavan, & Chakilam, 2001), an XML document is regarded as a sequence of ePaths (Element Path) associated with content. An ePath is defined as the path leading to the content information associated with leaf nodes from the XML root in the hierarchical structure. A set of XML documents in a database can be indexed by a document-ePath bitmap index. In such a bitmap index, a column represents a unique ePath, and a row represents an XML document.

Below we show an example to describe such a bitmap index (Yoon et al., 2001). All the possible ePaths in the three XML documents in Table 3 are: P_0=/e0/e1, P_1=/e0/e2/e3, P_2=/e0/e2/e4, P_3=/e0/e2/e5, P_4=/e0/e2/e4/e6, P_5=/e0/e2/e4/e7, P_6=/e0/e8, P_7=/e0/e9

In a document-ePath bitmap index, if a document has a certain ePath, then the corresponding bit is set to 1, otherwise it is set to 0. By examining the three documents in Table 3, we could see that D_1 contains paths P_0, P_1, P_2, P_3; D_2 contains paths P_0, P_1, P_2, P_4, P_5, P_6; D_3 contains paths P_0, P_1, P_2, P_3, P_7.

(D_1)	(D_2)	(D_3)
<e0> <e1>V1</e1> <e2> <e3>V2 V3 V4</e3> <e4>V3 V8</e4> <e5/> </e2> </e0>	<e0> <e1>V1</e1> <e2> <e3>V3 V7</e3> <e4>V9 <e6>V4</e6> <e7>V6</e7> </e4> </e2> <e8>V6 V12</e8> </e0>	<e0> <e1>V11</e1> <e2> <e3>V2 V7</e3> <e4>V3 V9</e4> <e5/> </e2> <e9>V5</e9> </e0>

Table 3. A set of simple XML documents

The bitmap index for the three documents in Table 3 is shown below in Figure 4. The rows represent the three XML documents, and the columns represent all the eight possible ePaths in these three documents.

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
D_1	1	1	1	1	0	0	0	0
D_2	1	1	1	0	1	1	1	0
D_3	1	1	1	1	0	0	0	1

Figure 4. A bitmap index for documents in Table 3.

Once we have the bitmap index, the distance (Hamming Distance) between two documents is defined as follows:

$$d(D_i, D_j) = | XOR(D_i, D_j) |, \quad (4)$$

where XOR is a bit-wise exclusive OR operator. The operator || denotes the number of 1's in a vector. For example, the distance between two documents D_1 and D_2 is 4.

In the above distance measure, only structural information is taken into consideration and content information is not used. In (Carmel, Maarek, Mandelbrod, Mass, & Soffer, 2003), XML documents are compared by combining both structural and content information together. Although the similarity measure introduced is intended to find the relevance score between a query and a document, it could be naturally used for finding the similarity between two documents. This is because the queries are represented as XML fragments, which are essentially XML documents.

In the regular vector space model, documents are represented by vectors in a space whose dimensions correspond each to a distinct unit. Typical units are words, phrases, which are all content information. The term $w_x(t)$ stands for the "weight" of term t in document x within the given collection. The weight is typically calculated from the document and collection statistics. One effective measure that could be used for calculating the similarity between two documents is the cosine measure (Salton & McGill, 1983), where:

$$\rho(D_i, D_j) = \frac{\sum_{t \in Di \cap Dj} w_{Di}(t) * w_{Dj}(t)}{|| D_i || * || D_j ||}, \quad (5)$$

This model works well with plain documents but for structural documents, the structural information is lost. By introducing distinct indexing units not as single terms but as pairs of the form (t, c), where t is qualified by the context c in which it appears, structural information is included. The context of appearance of a term is represented by the path leading to the term from

the XML root in the hierarchical structure of the document. This way, the weight of individual terms in Formula (5) should be replaced by the weight of terms in context, denoted as $w_x(t,c)$. Context matching is also relaxed by introducing a function cr that calculates the context resemblance between contexts. Formula (5) becomes:

$$\rho(D_i, D_j) = \frac{\sum_{(t,c_m)\in Di} \sum_{(t,c_n)\in Dj} w_{Di}(t,c_m) * w_{Dj}(t,c_n) * cr(c_m,c_n)}{\| D_i \| * \| D_j \|}, \quad (6)$$

Various instantiations could be used to measure the context resemblance. Four examples of cr functions are introduced in the paper.

Perfect Match:

$$cr(c_m,c_n) = \begin{cases} 1 & c_m = c_n \\ 0 & \text{otherwise} \end{cases}$$

Only (t,c) pairs that appear in both in of the two documents will be counted.

Partial Match:

$$cr(c_m,c_n) = \begin{cases} \dfrac{1+|c_m|}{1+|c_n|} & c_m \text{ subsequence of } c_n \\ 0 & \text{otherwise} \end{cases}$$

This measure requires context c_m be contained in context c_n. $|c_m|$ is the number of tags in context c_m and $|c_n|$ is the number of tags in context c_n. For example,

$$cr("/article/bibl","/article/bm/bib/bibl/bb") = 3/6 = 0.5$$

Fuzzy Match:

This type of matching allows for even finer grain similarity between contexts, such as treating context as strings and using string-matching techniques.

Flat:

$\forall c_m, c_n, cr(c_m, c_n) = 1$

This model completely ignores contexts by regarding all the contexts as the same context.

3.3.2 Overview

This subsection discussed the existing techniques used in IR community to find the similarity between hierarchical structured text databases. In (Yoon et al., 2001), bitmap indexing is used to index XML documents based on their structural information. A distance measure based on the bitmap index is derived. The advantage of this approach is efficiency by using the bit-wise operations (such as XOR) once the bitmap index is calculated. The authors in (Carmel et al., 2003) extend the regular vector space model for plain documents. Structural information and content information are used simultaneously to derive the new model and a similarity measure based on this new model is defined. The similarity function depends on the weighting function as well as the context matching scheme.

4 MINING TASKS

Structural distance metrics (either based on tree edit distances, or on graph matching, or on IR approaches) can be used in mining tasks on hierarchical structures, and specifically on their inherent structural aspect. Specifically, we explore clustering, classification and similarity ranking.

Clustering is a statistical technique that generates groups of data. Members of the same group should have a high degree of association (i.e. to be similar to each other), while members of different groups should have a low degree of association. In particular, clustering of hierarchical structures results in groups of structurally similar hierarchical structures. Structural similarity is captured by the structural distance metrics discussed in the previous section.

Classification differs from clustering, since, in the former, the groups are known a priori. In particular, classification assigns new hierarchical structures to pre-defined groups of structurally similar hierarchical structures. As in clustering, structural similarity is captured by the structural distance metrics discussed in the previous section.

In a similarity ranking task, a kind of a test pattern that describes data is provided and data similar to that pattern is retrieved. Retrieved data is presented in an order according to their degree of similarity with the test pattern. In particular, similarity ranking for hierarchical structures retrieves structures which are similar to a pre-defined one. Again, structural similarity is captured by structural distance metrics.

From those mining tasks, clustering hierarchical structures is quite popular in the research literature. Most of the approaches presented actually deal with clustering. For this reason we first give an overview of clustering methods.

4.1 Clustering Methods

Clustering methods are divided into two broad categories. Non-hierarchical methods group a data set into a number of clusters. They have low computational requirements, ($O(kn)$, if for example n documents need to be grouped into k clusters), but certain parameters like the number of formed clusters must be known a priori. A popular example of such method is the *K-Means*

algorithm (MacQueen, 1967; Rasmussen, 1992). Given a pre-defined number of clusters *c*, the algorithm creates *c* cluster centers which correspond to *c* clusters. Then, each object from the data set to be clustered is assigned to the cluster whose center is the nearest. Then the cluster centers are re-computed, and the process is continued until the cluster centers do not change.

Hierarchical methods produce nested sets of data (hierarchies), in which pairs of elements or clusters are successively linked until every element in the data set becomes connected. They are computationally expensive ($O(n^3)$) if *n* documents need to be clustered. However, hierarchical methods have been used extensively as a means of increasing the effectiveness and efficiency of retrieval (Voorhees, 1985). *Single link, complete link* and *group average link* are known as hierarchical clustering methods. All these methods are based on a similar idea. Each element of the data set to be clustered is considered to be a single cluster. The clusters with the minimum distance (i.e. maximum similarity) are merged and the distance between the remaining clusters and the new, merged one is recalculated. The process continues until there is only one cluster. In single link (complete link), the distance between two non-single clusters is defined as the minimum (maximum) of the distances between all pairs of elements so that one element is in the first cluster and the other element is in the second cluster. In group average link, the distance between two non-single clusters is defined as the mean of the distances between all pairs of elements so that one element is in the one cluster and the other element is in the other cluster. For a wide ranging overview of clustering methods one can refer to (Rasmussen, 1992;Halkidi, Batistakis, & Vazirgiannis, 2001).

We next describe approaches that exploit structural distance metrics to perform mining tasks on hierarchical structures.

4.2 Related Techniques and Algorithms

In (Nierman & Jagadish, 2002), tree structures encoded as XML documents are clustered using the group average link hierarchical method. The distance metric used to quantify structural similarity is a variation of the tree edit distance suggested in (Chawathe, 1999). Specifically, the set of tree edit operations used to calculate the tree edit distance includes two new ones which refer to whole trees (*insert_tree* and *delete_tree* operations) rather than nodes. Trees are pre-processed for checking whether a subtree is contained in another tree. Such pre-processing is needed to precalculate costs for sequences of single *insert_tree* operations, or combinations of *insert_tree* operations and *insert node* operations. Their experiments on real and synthetic structures show low number of misclustered documents.

In (Dalamagas, Cheng, Winkel, & Sellis, 2004), clustering of hierarchical structures encoded as XML documents is performed exploiting tree structural summaries. These summaries maintain the structural relationships between the elements of hierarchical structures and at the same time have minimal processing requirements instead of the original structures. To this extent, structural summaries are representatives of the original structures. Figure 5 presents an example of structural summary extraction. The extraction task is based on a nesting reduction phase followed by a repetition reduction phase. The phases eliminate nested and repeated nodes that provide the same structural information many times, causing redundancy. The distance metric used to quantify structural similarity is a variation of the tree edit distance suggested in (Chawathe, 1999) that avoids the pre-computation of the edit graph (see Section 3.1.2). Experiments with single link hierarchical clustering on real and synthetic data show high quality clustering. The results are confirmed with non-hierarchical clustering algorithms as well as with the kNN classification algorithm.

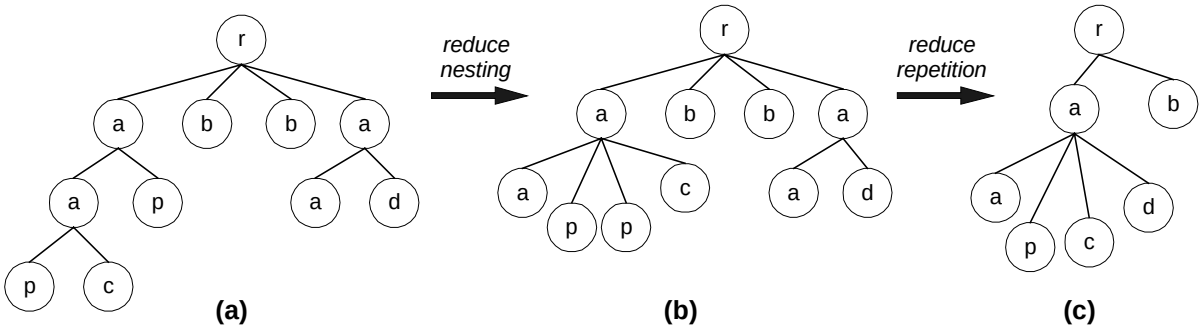

Figure 5. Structural summary extraction.

The notion of representative structures but for clusters and not for the original hierarchical structures is introduced in (Costa, Manco, Ortale, & Tagarelli, 2004). Clustering is performed on tree structures encoded as XML documents. The structural distance metric exploited in the clustering algorithm is not based on tree edit distances but on the percentage of common paths between the structures. Matching trees and merge trees are used to extract cluster representatives. A matching tree for two tree structures is defined as the tree that includes only their common nodes and maintain their structural relationships (i.e a lower-bound structure). A merge tree for two tree structures is defined as the tree that includes all their nodes and maintain their structural relationships (i.e an upper-bound structure). A cluster representative is constructed from the cluster merge tree by deleting nodes in such a way that the distance between the refined merge tree and the other members of the cluster is minimized. In Figure 6(e) presents an example of a representative tree structure for trees in (a) and (b). Note that tree in (c) is the lower-bound tree (i.e. matching tree) and tree in (d) is the upper-bound tree (i.e. merge tree) (originally taken from (Costa et al., 2004).

In (Lian, Cheung, Mamoulis, & Yiu, 2004), the graph distance metric introduced in (Bunke & Shearer, 1998; Bunke et al., 2000) (see Section 3.2.2) is used in a hierarchical clustering

algorithm to group together structurally similar XML documents. Such grouping is then used to improve the cost of query processing and evaluation in case these documents are stored in tables of relational database systems. Specifically, the grouping decreases the number of join operations needed between tables during the query evaluation. The approach is highly scalable, compared to the approaches based on tree edit distances presented before. The actual distance calculation between two XML documents is done on *s-graphs*, which includes all distinct nodes and edges appearing in either document.

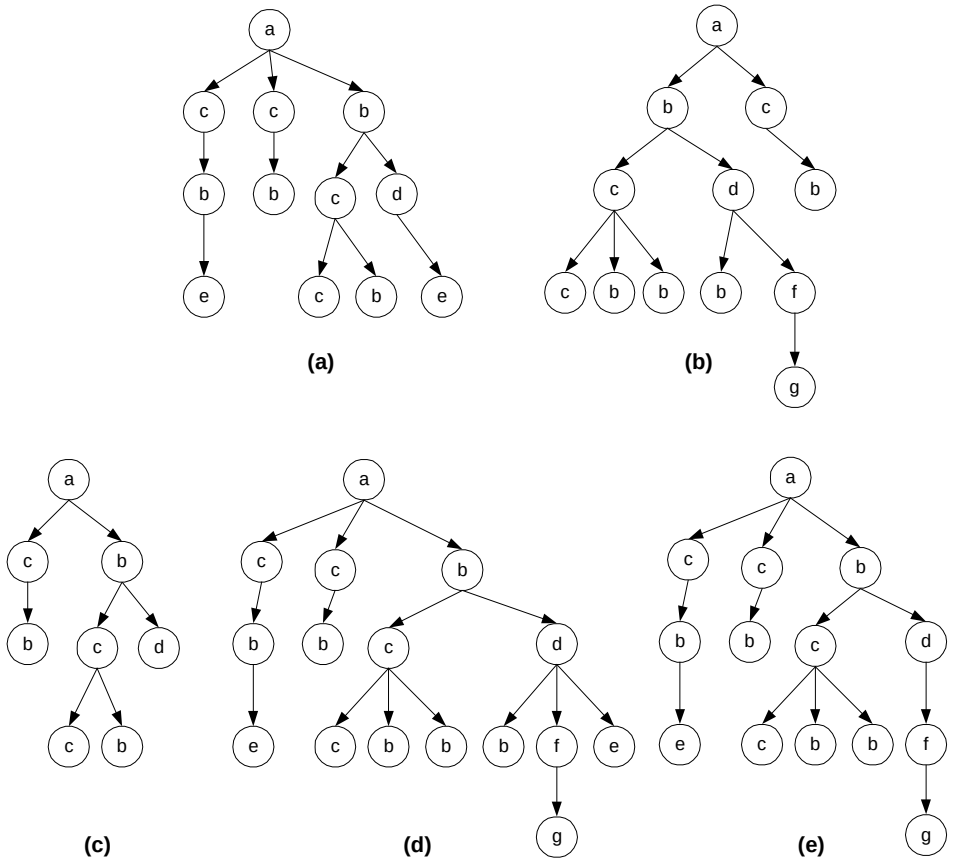

Figure 6. (a)/(b) example tree structures, (c) lower-bound, (d) upper-bound, (e) representative tree.

Similarity ranking techniques among XML documents and DTDs are presented in (Bertino, Guerrini, & Mesiti, 2004). It differs than the previous approaches since it explores to what extent

a hierarchical structure satisfies constraints imposed by grammars like DTDs. The similarity measure does not consider the structural aspect directly (as those based on tree edit distances) but only through the number of the so-called *plus*, *common* and *minus* element. Plus elements are those that appear in an XML document but not in the DTD used to rank the documents. Common elements are those that appear in an XML document and in that DTD. Minus elements are those that do not appear in an XML document but appear in that DTD. The level of each element is also taken into account: elements at higher levels are given higher weight compared to elements at lower level.

4.3 Overview

From the approaches presented, it is clear that the key issue in mining hierarchical structures using structural distance metrics is not the design and study of new mining tools, like new clustering tools. The strong technical background established in the past on such tools is directly exploited and applied in the context of hierarchical structures. Most of the approaches use a kind of hierarchical clustering algorithm with a structural distance metric based on variations of tree-edit distances. All approaches result in high quality clustering, indicating that structural distance metrics can really capture similarities and dissimilarities between different hierarchical structures.

An interesting point is the way that most of the approaches evaluate their methods. Since it is quite a hard task to manually examine the results of a clustering procedure, evaluation is based on a-priori knowledge of the cluster which a hierarchical structure belong to. The XML technology provides the means for such a task. XML documents that encode hierarchical structures are generated automatically from DTDs. Clustering is performed for these documents, and the clusters detected should ideally correspond (one by one) to the DTDs used. In case that real documents are used, it is enough to know the DTDs that these documents conform to.

Finally, we note that some methods introduce structures which summarize the original hierarchical structures. This speeds up the mining tasks, but on the other hand different hierarchical structures might have the same summary structure, posing difficulties in capturing fine structural dissimilarities.

5 CONCLUSIONS AND FUTURE PERSPECTIVES

Hierarchical structures is a popular means of organizing data, especially after the proliferation of the XML language. Management of data encoded in such structures have been a popular research issue. However, the structural aspect has received strong attention only lately with the usage of structural distance metrics. A structural distance metric can quantify the structural similarity between hierarchical structures. For example, it can estimate how similar two spatial entities (e.g. spatial configurations with forests, lakes, etc.) are. Since hierarchical structures are tree or graph structures, structural distance metrics can be defined using the notion of tree edit distance or graph matching. Actually, the majority of the approaches suggested in the research literature, and presented in detail in this chapter, are based on variations of tree edit distances to define structural distance metrics.

Structural distance metrics can be used in clustering algorithms, classification algorithms and similarity ranking mechanisms to support mining tasks for hierarchical structures. For instance, clustering by structure is a mining task that can identify spatial entities with similar structure, e.g. entities with areas that include forests with lakes. The chapter presented approaches that exploited structural distance metrics to perform such kind of mining tasks.

Structural distance metrics based on tree edit distances are quite popular, and they can capture fine structural dissimilarities. However, their calculation is quite intensive. On the other hand,

structural distance metrics, like for example the one based on maximum and minimum common subgraphs (Bunke et al., 2000; Lian et al., 2004), are scalable enough. However, they cannot capture fine structural dissimilarities. Vector-based approaches that capture the hierarchical relationships of tree structures should be extensively explored as a basis to design appropriate indexes for the efficient calculation of structural distance metrics. For example, in (Flesca, Manco, Masciari, Pontieri, & Pugliese, 2002) tree structures are linearized intro numerical sequences, and then discrete fourier transform compares the encoded structures in the domain of frequencies.

An important issue is that certain applications may restrict the context of interest for calculating structural distance metrics to certain parts instead of the whole hierarchical structure. Thus, flexible models to manipulate hierarchical structures taking into consideration their granularity should be examined. Such models have been used for calculating distance metrics to estimate content similarity in older works for structured text database retrieval (Baeza-Yates & Navarro, 1996).

Concerning the definition of structural distance metrics, there are certain requirements that should be taken into account. An example is the need or not for ordering in the objects of hierarchical structures. For instance, the ordering of two objects *A* and *B* which are children of the object *C* might be important if *A*, *B* and *C* encode macromolecular tree patterns, but might be of no interest if they encode entities from other knowledge domains. Another example is the difference in the importance of the objects as structural primitives in a hierarchical structure. In (Bertino et al., 2004), for instance, the level of each XML element is taken into account: elements at higher levels are given higher weight compared to elements at lower level.

In any case, mining hierarchical structures using structural distance metrics is a problem which does not really requires the design and study of new mining tools, like new clustering tools. Mining tasks like clustering/classification algorithms and similarity ranking have been widely studied in the research literature. The strong technical background established can be directly exploited and applied in the context of hierarchical structures. The key issues for successful application are the definition of an appropriate structural distance metric and its efficient calculation.

6 REFERENCES

Abiteboul, S., Buneman, P. and Suciu, D. (2000). Data on the Web. Morgan Kaufmann Publishers.

Bertino, E., Guerrini, G. and Mesiti, M. (2004). A matching algorithm for measuring the structural similarity between an XML document and a DTD and its applications. *Information Systems*, 29(1).

Bunke, H., Jiang, X., and Kandel, A. (1999). A metric on graphs for structural pattern recognition. In H.W.Schüssler (ed.): *Signal Processing II: Theories and Applications*, Elsevier Science Publishers B.V. (North Holland).

Bunke, H. and Shearer, K. (1998). A graph-distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3-4).

Bunke, H., Jiang, X. and Kandel, A. (2000). On the minimum common supergraph of two graphs. *Computing*, 65(1).

Carmel, D., Maarek, Y., Mandelbrod, M., Mass, Y., and Soffer, A. (2003). Searching XML documents via XML fragments. Proceedings of the *International Conference on Research and Development in Information Retrieval (SIGIR 2003)*, Toronto, Canada.

Chan, C. and Ioannidis, Y. (1998). Bitmap index design and evaluation. Proceedings of the *International Conference on Management of Data (ACM SIGMOD'98)*, Seattle, WA, USA.

Chawathe, S. S., Rajaraman, A., Garcia-Molina, H. and Widom, J. (1996). Change detection in hierarchically structured information. Proceedings of the *International Conference on Management of Data (ACM SIGMOD'96)*, Montreal, Canada.

Chawathe, S. S. (1999). Comparing hierarchical data in external memory. Proceedings of the *International Conference on Very Large Databases (VLDB'99)*, Edinburgh, Scotland, UK.

Cobena, G., Abiteboul, S. and Marian, A. (2002). Detecting changes in XML documents. Proceedings of the *International Conference on Data Engineering (ICDE'02)*, San Jose, California, USA.

Costa, G., Manco, G., Ortale, R. and Tagarelli, A. (2004). A tree-based approach to clustering XML documents by structure. Proceeding of the *European Conference on Principles and Practice of Knowledge Discovery in Databases Knowledge (PKDD'04)*, Pisa, Italy.

Dalamagas, T., Cheng, T., Winkel, K. J. and Sellis, T. (2004). Clustering XML documents using structural summaries. Proceedings of the *EDBT Workshop on Clustering Information over the Web (ClustWeb'04)*, Heraklion, Greece.

Direen, H. G. and Jones, M. S. (2003). Knowledge management in bioinformatics. In Chaudhri, A. B., Rashid, A. and Zicari, R. (ed.): XML Data Management. Addison Wesley.

Fernandez, M.-L. and Valiente, G. (2001). A Graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6-7).

Flesca, S., Manco, G., Masciari, E., Pontieri, L. and Pugliese, A., (2002). Detecting structural similarities between XML documents. Proceedings of the *Workshop on The Web and Databases (WebDB'02)*, Madison, Wisconsin, USA.

Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J. D., Vassalos, V. and Widom, J. (1997). The TSIMMIS approach to mediation: data models and languages. *Journal of Intelligent Information Systems*, 8(2).

Garofalakis, M., Gionis, A., Rastogi, R., Seshadri, S. and Shim, K. (2000). XTRACT: A system for extracting document type descriptors from XML documents. Proceedings of the *International Conference on Management of Data (ACM SIGMOD'00)*, Texas, USA.

Garofalakis, M., Gionis, A., Rastogi, R., Seshadri, S. and Shim, K. (2003). XTRACT: learning document type descriptors from XML document collections. *Data Mining and Knowledge Discovery*, 7.

Halkidi, M., Batistakis, Y. and Vazirgiannis, M. (2001). Clustering algorithms and validity measures. Proceedings of the *Scientific and Statistical Databases Management Conference (SSDBM'01)*, Virginia, USA.

Levi, G. (1972). A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9.

Lian, W., Cheung, D. W., Mamoulis, N. and Yiu, S. M. (2004). An efficient and scalable algorithm for clustering XML documents by structure. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 16(1).

MacQueen J. B. (1967). Some methods for classification and analysis of multivariate observations. Proceedings of the *Symposium on Math. Statist. And Probability*, University of California Press, Berkeley.

McGregor, J.J. (1982). Backtrack search algorithms and the maximal common subgraph problem. *Software Practice and Experience*, 12.

Nierman, A. and Jagadish, H. V. (2002). Evaluating structural similarity in XML documents. Proceedings of the *Workshop on The Web and Databases (WebDB'02)*, Madison, Wisconsin, USA.

Rasmussen, E. (1992). Clustering algorithms. In Frakes, W. and Baeza-Yates, R. (ed.): Information retrieval: data structures and algorithms. Prentice Hall.

Read, R.C., Corneil, D.G., (1977). The graph isomorphism disease. *J. Graph Theory*, 1, 1977.

Salton, G. and McGill, M. J. (1983). Introduction to modern information retrieval. McGraw-Hill, New York.

Sanfeliu, A. and Fu, K.S. (1983). A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13.

Sankoff, D. and Kruskal, J. (1999). Time warps, string edits and macromolecules, the Theory and Practice of Sequence Comparison. CSLI Publications.

Selkow, S. M. (1977). The tree-to-tree editing problem. *Information Processing Letters*, 6.

Shapiro, L.G. and Haralick, R.M. (1981). Structural descriptions and inexact matching. *IEEE Trans. On Pattern Analysis and Machine Intelligence*, 3.

Tsai, W. H. and Fu, K. S. (1979). Error-correcting isomorphisms of attributed relational graphs for pattern analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(12).

Ullman, J.R., (1976). An algorithm for subgraph isomorphism. *Journal of ACM*, 23(1).

Voorhees, H. (1985). The effectiveness and efficiency of agglomerative hierarchic clustering in document retrieval. PhD Thesis, Cornell University, Ithaca, New York.

Wagner, R. and Fisher, M. (1974). The string-to-string correction problem. *Journal of ACM*, 21(1).

Wilson, R., Cobb, M., McCreedy, F., Ladner, R., Olivier, D., Lovitt, T., Shaw, K., Petry, F. and Abdelguerfi, M. (2003). Geographical data interchange using XML-enabled technology within the GIDB System. In Chaudhri, A. B., Rashid, A. and Zicari, R. (ed.): XML Data Management, Addison Wesley.

Baeza-Yates, R. and Navarro, G. (1996). Integrating contents and structure in text retrieval. *ACM SIGMOD Record*, 25(1).

Yoon, J., Raghavan, V., Chakilam, V. (2001). Bitmap indexing-based clustering and retrieval of XML documents. Proceeding of the *ACM SIGIR Workshop on Mathematical/Formal Methods in IR*, New Orleans, Louisiana, USA.

Zhang, K. and Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18.