

Clustering XML documents by structure

Theodore Dalamagas¹, Tao Cheng², Klaas-Jan Winkel³, and Timos Sellis¹

¹ School of Electr. and Comp. Engineering,
National Technical University of Athens, Greece
`{dalamag,timos}@dmlab.ece.ntua.gr`

² Dept. of Computer Science
University of California, Santa Barbara, USA
`taocheng@cs.ucsb.edu`

³ Faculty of Computer Science,
University of Twente, the Netherlands
`winkel@cs.utwente.nl`

Abstract. This work explores the application of clustering methods for grouping structurally similar XML documents. Modeling the XML documents as rooted ordered labeled trees, we apply clustering algorithms using distances that estimate the similarity between those trees in terms of the hierarchical relationships of their nodes. We suggest the usage of tree structural summaries to improve the performance of the distance calculation and at the same time to maintain or even improve its quality. Experimental results are provided using a prototype testbed.

Keywords: XML, structural similarity, tree distance, structural summary, clustering

1 Introduction

The XML language is becoming the standard Web data exchange format, providing interoperability and enabling automatic processing of Web resources. While the processing and management of XML data are popular research issues [1], operations based on the structure of XML data have not yet received strong attention. Applying structural transformations and grouping together structurally similar XML documents are examples of such operations. Structural transformations are the basis for using XML as a common data exchange format. Grouping together structurally similar XML documents refers to the application of clustering methods using distances that estimate the similarity between tree structures in terms of the hierarchical relationships of their nodes.

There are many cases where clustering by structure can assist application tasks. Many XML documents are constructed from data sources without DTDs. XTRACT [2] and DDbE⁴ are systems that automatically extract DTDs from XML documents. Identifying groups of XML documents of similar structure can be useful for such systems, where a collection of XML documents should be

⁴ <http://www.alphaworks.ibm.com/tech/DDbE>

first grouped into sets of structurally similar documents and then a DTD can be assigned to each set individually. Moreover, since the XML language can encode hierarchical data, clustering XML documents by structure can be exploited in any application domain that needs management of hierarchical structures. For example, the discovery of structurally similar macromolecular tree patterns, encoded as XML documents, is a critical task in bioinformatics [3, 4].

The main contribution of this work is a methodology for grouping structurally similar XML documents. Modeling XML documents as rooted ordered labeled trees, we face the ‘clustering XML documents by structure’ problem as a ‘tree clustering’ problem. We propose the usage of tree structural summaries that have minimal processing requirements instead of the original trees representing the XML documents. We present a new algorithm to calculate tree edit distances and define a structural distance metric to estimate the structural similarity between the structural summaries of two trees. Using this distance, we perform clustering of XML data sets. Experimental results indicate that our algorithm for calculating the structural distance between two trees, representing XML documents, provides high quality clustering and improved performance. Also, the usage of structural summaries to represent XML documents instead of the original trees, improves further the performance of the structural distance calculation without affecting its quality.

This paper is organized as follows. Section 2 presents background information on tree-like representation of XML data and analyzes tree editing issues. Section 3 suggests the tree structural summaries. Section 4 presents a new algorithm to calculate the tree edit distance between two trees and introduces a metric of structural distance. Section 5 analyzes the clustering methodology. Section 6 presents the evaluation results, and, finally, Section 7 concludes our work.

2 Tree editing

The *XML data model* is a graph representation of a collection of atomic and complex objects, that without the IDREFS mechanism becomes a *rooted ordered labeled tree* [1]. Since we use such rooted ordered labeled trees to represent XML data, we exploit the notions of *tree edit sequence* and *tree edit distance* originating from editing problems for rooted ordered labeled trees [3]:

Definition 1. *Let T_1, T_2 be rooted ordered labeled trees. A tree edit sequence is a sequence of tree edit operations (insert node, delete node, etc) to transform T_1 to T_2 .*

Definition 2. *Let T_1, T_2 be rooted ordered labeled trees. Assuming a cost model to assign costs for every tree edit operation, 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 .*

All of the algorithms for calculating the edit distance for two ordered labeled trees are based on dynamic programming techniques related to the string-to-string correction problem [5]. [6] was the first work that defined the tree edit

distance and provided algorithms to compute it, permitting operations anywhere in the tree. Selkow’s [7] and Chawathe’s (II) [8] algorithms allow insertion and deletion only at leaf nodes, and relabel at every node. The former has exponential complexity, while the latter is based on the model of edit graphs which reduces the number of recurrences needed. Chawathe’s (I) algorithm [9] starts using a pre-defined set of matching nodes between the trees, and is based on a different set of tree edit operations than Chawathe’s (II). It allows insertion and deletion only at leaf nodes. Zhang’s algorithm [10] permits operations anywhere in the tree. We believe that using insertion and deletion only at leaves fits better in the context of XML data. For example it avoids deleting a node and moving its children up one level. The latter destroys the membership restrictions of the hierarchy and thus is not a ‘natural’ operation for XML data. In this work, we consider Chawathe’s (II) algorithm as the basic point of reference for tree edit distance algorithms, since it permits insertion and deletion only at leaves and is the fastest available.

3 Tree structural summaries

Nesting and repetition of elements is the main reason for XML documents to differ in structure although they come from a data source which uses one DTD. A *nested-repeated node* is a non-leaf node whose label is the same with the one of its ancestor. Following a pre-order tree traversal, a *repeated node* is a node whose path (starting from the root down to the node itself) has already been traversed before. Figure 1 has an example of redundancy: trees T_1 and T_3 differ because of nodes A (nested-repeated) and B (repeated). We perform *nesting reduction* and *repetition reduction* to extract structural summaries for rooted ordered labeled trees which represent XML documents. Both kind of reductions need only a pre-order traversal each on the original tree.

Nesting reduction reduces the nesting in the original tree so that there will be no nested-repeated nodes. We traverse the tree using pre-order traversal to detect nodes which have an ancestor with the same label in order to move up their subtrees. This process may cause non-repeating nodes to be repeating nodes. This is why we deal first with the nesting reduction and then with the repetition reduction. Repetition reduction reduces the repeated nodes in the original tree. We traverse the tree using pre-order traversal, too, ignoring already existed paths and keeping new ones, using a hash table.

Figure 1 presents an example of structural summary extraction from T_1 . Applying the nesting reduction phase on T_1 we get T_2 , where there are no nested-repeated nodes. Applying the repetition reduction on T_2 we get T_3 which is the structural summary tree without nested-repeated and repeated nodes.

4 Tree structural distance

Our approach for the tree edit distance between structural summaries of rooted ordered labeled trees uses a dynamic programming algorithm which is close

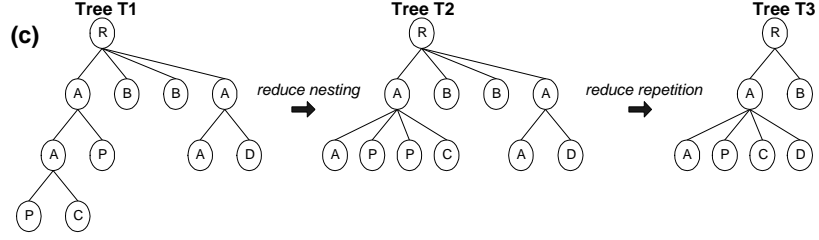


Fig. 1. Structural summary extraction.

to Chawathe's algorithm (II) [8] in terms of the tree edit operations that are used. However, the recurrence that we use does not need the costly edit graph calculation of the latter (see the timing analysis in Section 6.1). Permitted tree edit operations are:

1. *insertion*: Let n be a node with label l to be inserted as the i_{th} child of node p in tree T . After the insertion operation $ins(n, i, p, l)$, n is a new leaf node with label l . We assign cost $cost_{ins} = 1$ to the insertion operation.
2. *deletion*: Let n be a leaf node in tree T . The deletion operation $del(n)$ will remove n from T . We assign cost $cost_{del} = 1$ to the deletion operation.
3. *update*: Let n be a node with label l . The update operation $upd(n, m)$ will change the label l to m for node n . We assign cost 1 to the update operation if $l \neq m$ or 0 if $l = m$.

$CalculateDistance(r_1, r_2)$ calculates the tree edit distance of T_1 and T_2 , with roots r_1 and r_2 , respectively:

```

int CalculateDistance(TreeNode s, TreeNode t) {
    int[][] D=new int[numOfChildren(s)+1][numOfChildren(t)+1];
    D[0][0]=UpdateCost(LabelOf(s), LabelOf(t));
    for (int i=1; i<=numOfChildren(s); i++)
        D[i][0]= [i-1][0]+numOfNodes(si);
    for(int j=1; j<=numOfChildren(t); j++)
        D[0][j]=D[0][j-1]+numOfNodes(tj);
    for (int i=1; i<=numOfChildren(s); i++)
        for (int j=1; j<=numOfChildren(t); j++)
            D[i][j]=Min(D[i-1][j-1]+CalculateDistance(si, tj),
                        D[i][j-1]+numOfNodes(tj),
                        D[i-1][j]+numOfNodes(si));
    Return D[numOfChildren(s)][numOfChildren(t)];
}

```

where: s_i (t_j) is the i_{th} (j_{th}) subtree of node s (t), $numOfChildren(s)$ returns the number of child nodes of node s , $numOfNodes(s)$ returns the number of nodes of the subtree rooted at s , $UpdateCost(LabelOf(s), LabelOf(t))$ returns the cost to make the label of node s the same as the label of node t (1 if $LabelOf(s) \neq LabelOf(t)$ or 0 otherwise).

In the algorithm, $D[i][j]$ keeps the tree edit distance between tree T_1 with only its first i subtrees and tree T_2 with only its first j subtrees. $D[0][0]$ keeps the distance between T_1 and T_2 , having only their roots (initially 0, since the examined trees are assumed to have same roots). Since the cost of an insert or delete operation is 1, we use $numOfNode(s_i)$ to represent the cost to delete the i th subtree of node s and $numOfNodes(t_j)$ to represent the cost to insert the j th subtree of node t . The main `for` nested loop first calculates the tree edit distance between tree T_1 with only its first subtree and tree T_2 with only its first subtree, then the distance between T_1 with only its first two subtrees and T_2 with only its first subtree, etc. In the end, the algorithm returns the distance between T_1 with all its subtrees and T_2 with all its subtrees. We call the function `CalculateDistance` once for each pair of nodes at the same depth in the 2 structural summary trees, so the complexity is $O(MN)$, where M is the number of nodes in the tree rooted at s , and N is the number of nodes in the tree rooted at t .

Let \mathcal{D} be the tree edit distance between two trees T_1 and T_2 calculated from the previous algorithm. Using \mathcal{D} , we can now define the *structural distance* \mathcal{S} between two structural summaries for rooted ordered labeled trees which represent XML documents.

Definition 3. Let T_1 and T_2 be two structural summaries for rooted ordered labeled trees that represent two XML documents, $\mathcal{D}(T_1, T_2)$ be their tree edit distance and $\mathcal{D}_{max}(T_1, T_2)$ be the maximum cost between the costs of all possible sequences of tree edit operations that transform T_1 to T_2 . The structural distance \mathcal{S} between T_1 to T_2 is defined as $\mathcal{S}(T_1, T_2) = \frac{\mathcal{D}(T_1, T_2)}{\mathcal{D}_{max}(T_1, T_2)}$.

To calculate \mathcal{D}_{max} , we calculate the cost to delete all nodes from T_1 and insert all nodes from T_2 . The $\mathcal{S}(T_1, T_2)$ is low when the trees have similar structure and high percentage of matching nodes and high when the trees have different structure and low percentage of matching nodes (0 (1) is the min (max) value).

5 Clustering trees

We chose single link hierarchical method [11, 12] to be the basic clustering algorithm for the core part of the experiments for our work since it has been shown to be theoretical sound, under a certain number of reasonable conditions [13]. We implemented a single link clustering algorithm using Prim's algorithm for computing the minimum spanning tree (MST) of a graph [14]. Given n structural summaries of rooted labeled trees that represent XML documents, we form a fully connected graph G with n vertices and $n(n-1)/2$ weighted edges. The weight of an edge corresponds to the structural distance between the vertices (trees) that this edge connects. The single link clusters for a *clustering level* l_1 can be identified by deleting all the edges with weight $w \geq l_1$ from the MST of G . The connected components of the remaining graph are the single link clusters.

A stopping rule is necessary to determine the most appropriate clustering level for the single link hierarchies. C -index [15, 16] exhibits excellent performance. C -index is a vector of pairs $((i_1, n_1), (i_2, n_2), \dots)$, where i_1, i_2, \dots are

the values of the index and n_1, n_2, \dots the number of clusters in each clustering arrangement. We can calculate i_1 for the first pair (i_1, n_1) of C-index vector as follows: $i_1 = (d_w - \min(d_w)) / (\max(d_w) - \min(d_w))$, where

1. $d_w = \text{Sum}(d_{w_1}) + \text{Sum}(d_{w_2}) + \dots + \text{Sum}(d_{w_{N_1}})$, with $\text{Sum}(d_{w_i})$ to be the sum of pairwise distances of all members of cluster C_i , $1 \leq i \leq n_1$,
2. $\max(d_w)$ ($\min(d_w)$): the sum of the n_d highest (lowest) pairwise distances in the whole set of data, that is, sort distances, higher first, and take the Top- n_d (Bottom- n_d) sum, given that $n_d = c_1 * (c_1 - 1) / 2 + c_2 * (c_2 - 1) / 2 + \dots + c_{n_1} * (c_{n_1} - 1) / 2$ (c_i = number of members in cluster C_i).

Similarly we can calculate i_2, i_3, \dots . We adopt C -index in the single link procedure by calculating its values, varying the clustering level in different steps. The number of clusters with the lowest C -Index is chosen [16].

6 Evaluation

We implemented a testbed to perform clustering on synthetic and real data, using structural distances⁵. Two sets of 1000 synthetic XML documents were generated⁶ from 10 real-case DTDs⁷, varying the parameter *MaxRepeats* to determine the number of times a node will appear as a child of its parent node. For real data set we used 150 documents from the ACM SIGMOD Record and ADC/NASA⁸. We chose single link to be the basic clustering algorithm for the core part of the experiments.

While checking time performance is straightforward, checking clustering quality involves the calculation of metrics based on an a priori knowledge of which documents should be members of the appropriate cluster. Such knowledge, in turn, presumes that we have a mapping between original DTDs and extracted clusters. To get such a mapping, we derived a DTD D_c for every cluster C and mapped it to the most similar of the original DTDs, by calculating the structural distance between the tree derived from D_c and each of the trees derived from the original DTDs⁹.

To evaluate the clustering results, we used two metrics quite popular in the research area of information retrieval: *precision PR* and *recall R* [13]. For an extracted cluster C_i that corresponds to a DTD DTD_i let (a) a_i be the number of the XML documents in C_i that were indeed members of that cluster (correctly clustered), (b) b_i be the number XML documents in C_i that were not members of that cluster (misclustered) and (c) c_i be the number of XML documents not in C_i , although they should be C_i 's members. Then $PR = \frac{\sum_i a_i}{\sum_i a_i + \sum_i b_i}$ and $R = \frac{\sum_i a_i}{\sum_i a_i + \sum_i c_i}$. High precision means high accuracy of the clustering task

⁵ All the experiments were performed on a Pentium III 800MHz, 192MB RAM.

⁶ www.alphaworks.ibm.com/tech/xmlgenerator

⁷ from www.xmlfiles.com and <http://www.w3schools.com>

⁸ www.acm.org/sigmod/record/xml and xml.gsfc.nasa.gov respectively

⁹ using www.alphaworks.ibm.com/tech/DDbE.

for each cluster while low recall means that there are many XML documents that were not in the appropriate cluster although they should have been. High precision and recall indicate excellent clustering quality.

Notice that there might be the case where there are clusters not mapped to any of the original DTD. We treated all XML documents in such clusters as misclustered documents. Based on the above, we present the timing analysis for calculating structural distances and the clustering results.

6.1 Timing analysis

We compared (a) the time to derive the 2 structural summaries from 2 rooted ordered labeled trees representing 2 XML documents plus (b) the time to calculate the structural distance between those 2 summaries, vs the time to calculate the structural distance between 2 rooted ordered labeled trees of 2 XML documents.

We tested both Chawathe’s algorithm and our algorithm using randomly generated XML documents, with their nodes ranging from 0 to 2000. This timing analysis gives an indication of how fast a file for storing pairwise structural distances is constructed. Such a file can then be used as an input in any clustering algorithm to discover clusters. A clustering algorithm needs to calculate $N*(N-1)/2$ pairwise structural distances, N the number of documents to be clustered.

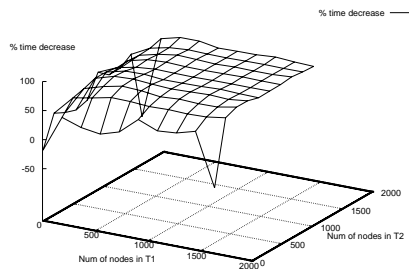


Fig. 2. % time decrease for structural distance calculation using tree summaries instead of using the original trees (Chawathe’s algorithm).

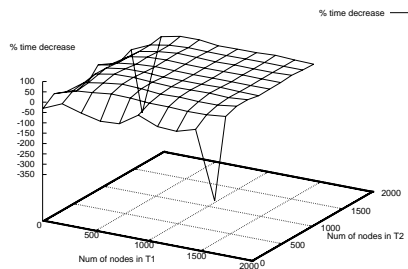


Fig. 3. % time decrease for structural distance calculation using tree summaries instead of using the original trees (our algorithm).

Figures 2 and 3 show the % time decrease for calculating the structural distance between 2 XML documents using their summaries instead of using the original trees, for Chawathe’s algorithm and our algorithm. Using summaries, the decrease lays around 80% on average for Chawathe’s and around 50% on average for our algorithm.

To give a sense about the scaling of the calculations, Figure 4 presents the % time decrease for calculating the structural distance between 2 XML documents, using our algorithm instead of Chawathe’s algorithm (52% on average).

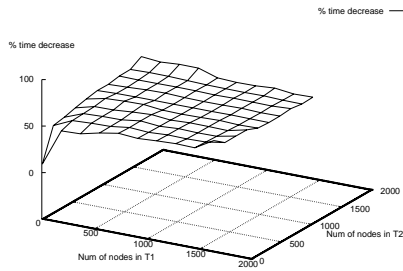


Fig. 4. % time decrease for structural distance calculation using our algorithm instead of Chawathe’s.

Chawathe’s algorithm is significantly slower than our algorithm due to the pre-calculation of the editgraph (see Section 2). We estimated that the editgraph calculation spends more than 50% of the time needed for the overall distance calculation.

6.2 Clustering evaluation

We performed single link clustering on synthetic and real data, using structural distances returned from Chawathe’s algorithm and our algorithm, with or without structural summaries, and calculated PR and R values.

Table 1 presents the PR and P values using Chawathe’s and our algorithm on synthetic and real data. For Chawathe’s algorithm, we note that for small trees ($maxRepeats = 3$) with only a few repeated elements and, thus, with the structural summaries being actually the original trees, the clustering results are the same with or without summaries. On the other hand, for larger trees ($maxRepeats = 6$) with many repeated elements there is a clear improvement using summaries, especially in the precision value (PR). For our algorithm, we note that summary usage keeps the already high quality clustering results obtained by clustering without using summaries. In any case, with or without summary, our algorithm shows better clustering quality either with small trees and only a few repeated elements or with larger trees and many repeated elements. We note that PR and R reach excellent values. For real data, summary usage maintains the already high quality clustering results obtained without using summaries. The evaluation results indicate that structural summaries maintain the clustering quality, that is they do not hurt clustering. Thus, using structural summaries we can clearly improve the performance of the whole clustering procedure, since the decrease on the time needed to calculate the tree distances using summaries is high (see Section 6.1). Furthermore, in any case, with or without summaries, our algorithm shows excellent clustering quality, and improved performance compared to Chawathe’s. Preliminary tests with other clustering algorithms showed similar results.

Synthetic data (CH: Chawathe's algorithm, D: our algorithm)	
CH without summaries (<i>maxRepeats</i> = 3)	CH with summaries (<i>maxRepeats</i> = 3)
<i>NumOfClusters</i> = 11, <i>Cluster.level</i> = 0.37	<i>NumOfClusters</i> = 11, <i>Cluster.level</i> = 0.37
<i>PR</i> = 0.71, <i>R</i> = 0.90	<i>PR</i> = 0.71, <i>R</i> = 0.90
CH without summaries (<i>maxRepeats</i> = 6)	CH with summaries (<i>maxRepeats</i> = 6)
<i>NumOfClusters</i> = 11, <i>Cluster.level</i> = 0.51	<i>NumOfClusters</i> = 12, <i>Cluster.level</i> = 0.50
<i>PR</i> = 0.58, <i>R</i> = 0.89	<i>PR</i> = 0.83, <i>R</i> = 0.96
D without summaries (<i>maxRepeats</i> = 3)	D with summaries (<i>maxRepeats</i> = 3)
<i>NumOfClusters</i> = 11, <i>Cluster.level</i> = 0.51	<i>NumOfClusters</i> = 11, <i>Cluster.level</i> = 0.51
<i>PR</i> = 1.00, <i>R</i> = 0.98	<i>PR</i> = 1.00, <i>R</i> = 0.98
D without summaries (<i>maxRepeats</i> = 6)	D with summaries (<i>maxRepeats</i> = 6)
<i>NumOfClusters</i> = 12, <i>Cluster.level</i> = 0.61	<i>NumOfClusters</i> = 11, <i>Cluster.level</i> = 0.56
<i>PR</i> = 1.00, <i>R</i> = 0.97	<i>PR</i> = 1.00, <i>R</i> = 0.98
Real data (CH: Chawathe's algorithm, D: our algorithm)	
CH without summaries	CH with summaries
<i>NumOfClusters</i> = 4, <i>Cluster.level</i> = 0.63	<i>NumOfClusters</i> = 3, <i>Cluster.level</i> = 0.63
<i>PR</i> = 1.00, <i>R</i> = 0.98	<i>PR</i> = 1.00, <i>R</i> = 1
D without summaries	D with summaries
<i>NumOfClusters</i> = 4, <i>Cluster.level</i> = 0.63	<i>NumOfClusters</i> = 3, <i>Cluster.level</i> = 0.63
<i>PR</i> = 1.00, <i>R</i> = 0.98	<i>PR</i> = 1.00, <i>R</i> = 1

Table 1. P, PR values for clustering synthetic and real data.

7 Conclusions

This work presented a framework for clustering XML documents by structure, exploiting distances that estimate the similarity between tree structures in terms of the hierarchical relationship of their nodes.

Modeling XML documents as rooted ordered labeled trees, we faced the ‘clustering XML documents by structure’ problem as a ‘tree clustering’ problem. We proposed the usage of tree structural summaries that have minimal processing requirements instead of the original trees representing the XML documents. Those summaries maintain the structural relationships between the elements of an XML document, reducing repetition and nesting of elements. Also, we presented a new algorithm to calculate tree edit distances and defined a structural distance metric to estimate the structural similarity between the summaries of two trees. We implemented a testbed to perform clustering on synthetic and real data, using structural distances. We provided timing analysis as well as precision PR and recall R values to evaluate each test case. Our results showed that structural summaries clearly improved the performance of the whole clustering procedure, since the decrease on the time needed to calculate the tree distances using summaries is high. On the other hand, summaries maintained the clustering quality. Moreover, our structural distance algorithm showed improved performance compared to Chawathe’s.

To the best of our knowledge, the only work directly compared with ours is [17]. Their set of tree edit operations include two new ones which refer to whole trees rather than nodes. They preprocess the trees to detect whether a subtree is contained in another tree. Their approach requires the same amount of computation with Chawathe’s algorithm. There are no results about PR and R values. In our work, we diminish the possibility of having repeated subtrees using

structural summaries instead of expanding the tree edit operations. Summaries are used as an index structure to speed up the tree distance calculation. Such an approach has the advantage of being useful to reduce the performance cost in every algorithm that estimates the structural distance between trees.

To conclude, this work successfully applied clustering methodologies for grouping XML documents which have similar structure, by modeling them as rooted ordered labeled trees, and utilizing their structural summaries to reduce time cost while maintaining the quality of the clustering results. As a future work, we will explore properties that tree distances present. Also, we will test how our approach scales using larger data sets of XML documents.

References

1. S. Abiteboul, P. Buneman, D. Suciu, *Data on the Web.*, Morgan Kaufmann, 2000.
2. M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, K. Shim, XTRACT: A system for extracting document type descriptors from XML documents, in: *Proceedings of the ACM SIGMOD Conference*, Texas, USA, 2000.
3. D. Sankoff, J. Kruskal, *Time Warps, String Edits and Macromolecules*, The Theory and Practice of Sequence Comparison, CSLI Publications, 1999.
4. H. G. Direen, M. S. Jones, *Knowledge management in bioinformatics*, in: A. B. Chaudhri, A. Rashid, R. Zicari (Eds.), *XML Data Management*, 2003, Addison Wesley.
5. R. Wagner, M. Fisher, The string-to-string correction problem, *Journal of ACM* 21 (1) (1974) 168–173.
6. K. C. Tai, The tree-to-tree correction problem, *Journal of ACM* 26 (1979) 422–433.
7. S. M. Selkow, The tree-to-tree editing problem, *Information Processing Letters* 6 (1977) 184–186.
8. S. S. Chawathe, Comparing hierarchical data in external memory, in: *Proceedings of the VLDB Conference*, Edinburgh, Scotland, UK, 1999, pp. 90–101.
9. S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, J. Widom, Change Detection in Hierarchically Structured Information, in: *Proceedings of the ACM SIGMOD Conference*, USA, 1996.
10. K. Zhang, D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, *SIAM Journal of Computing* 18 (1989) 1245–1262.
11. E. Rasmussen, Clustering algorithms, in: W. Frakes, R. Baeza-Yates (Eds.), *Information Retrieval: Data Structures and Algorithms*, Prentice Hall, 1992.
12. M. Halkidi, Y. Batistakis, M. Vazirgiannis, Clustering algorithms and validity measures, in: *SSDBM Conference*, Virginia, USA, 2001.
13. C. J. van Rijsbergen, *Information Retrieval*, Butterworths, London, 1979.
14. J. C. Gower, G. J. S. Ross, Minimum spanning trees and single linkage cluster analysis, *Applied Statistics* 18 (1969) 54–64.
15. L. J. Hubert, J. R. Levin, A general statistical framework for accessing categorical clustering in free recall, *Psychological Bulletin* 83 (1976) 1072–1082.
16. G. W. Milligan, M. C. Cooper, An examination of procedures for determining the number of clusters in a data set, *Psychometrika* 50 (1985) 159–179.
17. A. Nierman, H. V. Jagadish, Evaluating structural similarity in xml documents, in: *Proceedings of the WebDB Workshop*, Madison, Wisconsin, USA, 2002.