Overview of Nearest Neighbor Subtree Search Methods

László Kovács, Tibor Répási, Erika Baksa-Varga

Department of Information Technology, University of Miskolc

kovacs@iit.uni-miskolc.hu, repasi@iit.uni-miskolc.hu, iitev@uni-miskolc.hu

Abstract. *In many scientific areas there is a frequent need to extract a common pattern from multiple data. In most cases, however, an approximate but low cost solution is preferred to a high cost exact match. To establish a fast search engine an efficient heuristic method should be implemented. Our investigation is devoted to the approximate nearest neighbor search (ANN) for unordered labeled trees. The proposed modified best-first algorithm provides a* $O((N_q+N_b)\cdot M + K\cdot N_q\cdot N_b/M)$ *cost function with simple implementation details. According to our test results, realized with smaller trees where the brute-force algorithm could be tested, the yielded results are a good approximation of the global optimum values. Based on the results of the tests, the execution cost for the base best-first algorithm is about one order of magnitude larger than the cost for the porposed modified best-first approximation method.*

Keywords: tree matching, approximate nearest neighbor search

1 Introduction

In this section we review the different researched approaches to comparing trees, as well as the algorithms developed so far to solve these problems. **P. Bille** published an extensive survey [1] on comparing trees with exact searching methods. As a conclusion from his work it turned out that all of the unordered versions of the problems in general are NP-hard. Indeed, the tree edit distance and alignment distance problems are even MAX SNP-hard. However, using special constraints polynomial time algorithms are available, just like for the ordered versions of the problems. These are all based on the classic technique of dynamic programming.

The *general ordered tree edit distance*, also called tree-to-tree correction problem was also fully reviewed in Technical Report 95-372 [2]. The problem was introduced by **Tai** [3] as a generalization of the string edit distance problem. His algorithm, which solved the problem without recursion, has its time and space complexity in O($nmd^2d'^2$), where n and m are the maximum number of children from any node in each of the trees, while *d* and *d'* are the maximum depth of the trees. **Zhang and Shasha** [4] numbered the trees using postorder traversal instead of preorder, so the algorithm's space complexity is O(*nm*), while its time complexity is O(*nmdd'*). **Klein** [5] solved the problem in $O(n^3 logn)$ time and O(*nm*) space. In his paper he proved that the algorithm can be extended to unrooted ordered trees within the same time and space bounds. **Chen** [6] applied fast matrix multiplication to solve the problem. The *unordered* version of the problem is NP-complete even for binary trees with a label alphabet of size 2. It was shown in [7] that under special restrictions polynomial time algorithms exist.

There are other variants of the edit distance problem as well. One of them is the *unit cost edit distance*, where unit cost is defined as the number of edit operations required. In [8] the ordered version of the problem is considered and an algorithm with $O(u^2 min\{n,m\} min\{l,l'\})$ time need is introduced, where *l* and *l'* are the number of leaves of the trees. The algorithm uses techniques from **Ukkonen** [9], and **Landau** and **Vishkin** [10]. The recursive solution of **Selkow** [11] used the basic operations, but insertions and deletions were restricted to leaf nodes only, which made the algorithm very simple and therefore its time complexity is O(*nm*). This is therefore sometimes referred to as the *1-degree edit distance*. **Chawathe** [12] utilizes the same restrictions, but in cases when external memory is needed to calculate the edit distance.

Tree inclusion, a special case of edit distance, is the problem to decide if tree T_1 can be included in T_2. T_1 is included in T_2 if there is a sequence of delete operations performed on T_2 which make T_2 isomorphic to T_1. For the *ordered* tree inclusion problem **Kilpeläinen and Mannila** [13] presented the first polynomial time algorithm using O(*nm*) time and space. A more space efficient version of this was given in [14] using O(*nd'*) space. Later **Richter** [15] and **Chen** [16] developed more complex algorithms. In [13], [17] it is shown that the *unordered* tree inclusion problem is NP-complete. In spite of this an algorithm using $O(mni2^{2i})$ time exists.

Torsello and Hancock [18] prove, that a tree *t'* can be generated from a tree *t* with a sequence of node removal operations if and only if *t'* is an obtainable subtree of the directed association graph. Consequently the minimum cost edited tree isomorphism between two trees is a maximum common consistent subtree of the two directed association graphs if the node removal cost is uniform, and this result can also be extended to non-uniform cost. The background for this lies in [19], where the relationship between graph edit distance and the size of the maximum common subgraph is shown, and also their computational equivalence is demonstrated. This is an important observation since it has been established by **Barrow** and **Burstall** [20] that the maximum common subgraph problem may be transformed into a maximum clique problem using a derived structure referred to as the association graph. **Pelillo et al.** [21], for instance, transform the tree isomorphism problem into a single max clique problem, a technique already used for the generic graph isomorphism problem. To obtain a maximal tree match, i.e. a maximal solution to the max clique problem, they use relaxation labeling. **Wang et al.** [22] considers the *largest approximately common subtree* problem for ordered, labeled trees using the edit distance to measure the dissimilarity of two trees. They present a dynamic programming algorithm, which runs as fast as the fastest known algorithm for computing the edit distance of trees.

This problem was investigated for *unordered* trees by **Khanna, Motwani and Yao** [23]. They created an algorithm for trees of bounded degree with performance ratio O($n\log\log n/\log^2 n$) and then extended this to trees of unbounded degree with at most poly-log labels, obtaining a ratio of O($n(\log\log n)^2/\log^2 n$). **Akutsu and Halldórsson** [24] also considers the approximation of the *largest common subtree* (and its special variation, the largest common edge subgraph) and *largest common point set* problems for unordered trees (and for ordered trees as a special case), and a general search algorithm is presented which approximates both problems within a factor of O($n/\log n$). For trees of bounded degree an improved algorithm is developed which approximates the largest common subtree within a factor of O($n/\log^2 n$). A large amount of work has been performed for comparing unordered trees based on various distance measures, especially on edit distance as the most commonly used distance measure. **Shasha et al.** [25], however, proposed a new approach, called Atree-Grep. They addressed the *approximate nearest neighbor search* problem for unordered labeled trees. Their algorithm, called 'pathfix', consists of two phases. First, the paths of the trees are stored in a suffix array and then the number of mismatching paths are counted between the query tree and the data tree. To speedup the search, they use a hash-based technique to filter out unqualified data trees at an early stage of the search. The algorithm has been implemented into two special Web-based search engines and proved to be fast, particularly when the dictionary size of node labels is large.

2 Distance measures for tree comparison

As can be seen, most of the proposals in subtree matching are based on the edit distance between trees. This distance metric is a natural extension of the edit distance concept used for string comparisons. This metric provides an exact distance measurement between the trees. The drawback of these algorithms is the high cost of the computations. In the case of online applications with large tree datasets, the execution time is a crucial factor. In these kinds of applications, an approximate but low cost solution is preferred to a high cost exact solution. Our investigation is devoted to the approximate nearest neighbor search (ANN) for unordered labeled trees. Our goal is to construct an efficient heuristic method for the ANN problem. Since the ANN problem for edit distance metric is an NP-problem as is proven in [7], a modified distance definition is introduced.

Let D denote a domain set. This contains the node labels. The symbol T denotes an unordered, labeled tree. The following denotations related to the tree structure are used in the paper:

n	a node of the tree
$l(n)$	the label of node n, $l(n) \in D$
T_D	set of unordered, labeled trees on D
$V(T)$	vertices of T
$E(T)$	edges of T
$r(T)$	the root node of T

In the case of edit distance, a set of elementary transformation functions is defined on T_D. This set is denoted as E_D. The cost value of the elementary transformations is a non-negative real number. The corresponding cost function is denoted by

$$c : E_D \rightarrow R^+.$$

It is assumed that T_D is closed to E_D, i.e.

$$e : T_D \rightarrow T_D \ \forall e \in E_D,$$

$$\forall T_1, T_2 \in T_D : \exists e_1, e_2, \ldots, e_m \in E_D: e(T_1) = e_m o e_{m-1} o..e_2 o e_1(T_1) = T_2.$$

Let us denote the set of chain of transformations from T_i to T_j by $E_{i,j}$. The cost of chain e is defined as the sum of the single transformation steps:

$$c(e) = \Sigma c(e_i).$$

The edit distance between T_i and T_j is defined as the minimal cost of transformation chains from T_i to T_j:

$$c_{i,j} = min\{ c(e) \mid e \in E_{i,j} \}.$$

Usually, the following elementary e operations are defined for tree objects:

- *relabel*: assigns a new node name to the root of the tree
- *insert*: inserting a new node into the children of the root node
- *delete*: deleting a node from the children of the root node
- *insert tree*: inserting a tree under the root node
- *delete tree*: deleting a tree from the children of the node

The list of elementary transformations with minimal cost is usually generated with a dynamic programming method. According to [1, pp.7], the tree distance value can be calculated using the following recursive formula:

$$d(0,0) = \quad 0$$

$$d(F,0) = \quad d(F\text{-}v,0) + c(v,0)$$

$$d(0,F) = \quad d(0,F-v) + c(0,v)$$

$$d(F_1,F_2) = \min \begin{cases} d(F_1-v,F_2) + c(T(v),0) \\ d(F_1,F_2-v) + c(0,T(v)) \\ d(F_1-T(v),F_2-T(w)) + c(T(v),T(w)) \end{cases}$$

where F denotes a tree and $T(v)$ denotes a tree with root element v. The computation cost of the basic dynamic programming method for trees is $O(|T|^4)$. This is a very high cost value for an ANN problem, as the distance computation should be calculated for a large number of pairs. It is proved in [25] that the ANN problem for edit distance metric is an NP-complete problem. In spite of this difficulty, most of the proposals for ANN searching for trees use the edit distance measure. There are very few proposals that apply a simplified distance function to provide a lower cost solution.

A good example for this approach is [25], where the distance from T_1 to T_2 is measured with the total number of root-to-leaf paths in T_1 that do not appear in T_2. The nodes in T_2 that do not appear in T_1 can be freely removed. As can be seen, this definition introduces an asymmetric distance concept. In the definition T_1 denotes the query tree while T_2 is the searched tree. In our approach, another simplified distance function was selected.

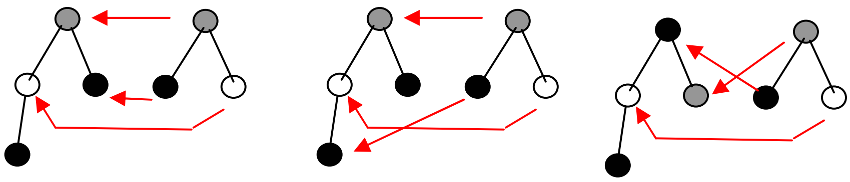
3 Modified best-first algorithm

Two trees are said to be similar if they have similar vertices with similar edges. During the editing process every vertex of the query tree is either transformed into a vertex of the base tree or it is deleted. Based on this transformation, every vertex of the query tree can be mapped either to a target vertex or to the sink symbol. Using this approach, a generalized mapping can be defined between the query and the base tree. We define *m( )* as the distance mapping from T_1 to T_2 in the following way:

1. $m : V(T_1) \rightarrow V(T_2) \cup \varepsilon$
2. $\forall v, m(v) \in V(T_2) : l(v) = l(m(v))$
3. $\forall v_1 \neq v_2, m(v_1), m(v_2) \in V(T_2) : m(v_1) \neq m(v_2)$
4. $\forall v_1 \neq v_2, m(v_1), m(v_2) \in V(T_2): v_1 < v_2 \Leftrightarrow m(v_1) < m(v_2)$

According to the first property, every node in T_1 is mapped either to a node in T_2 or is deleted, i.e. it is mapped to the ε symbol. The second property says that a vertex should be mapped only to nodes of the same label. Due to the third property, the different query vertices can not be mapped to the same base vertex. The fourth property is called ancestor condition, the ancestor-descendants relationship among the query vertices must be preserved in the target tree, too.

Other types of relationships among the query vertices are neglected and not preserved. In this approach, the sibling vertices may be mapped to parent-child vertices, if the existing parent-child relationships are preserved. The parent-child relationships are the only important information stored in the query tree. The absence of an edge means in our approach a 'do not know' information. In this case, we don not care about the existence of an edge between the mapped vertices in the base tree. Figure 1 shows an example for this mapping.



a) *b)* *c)*

Figure 1

Figure 1a) and Figure 1b) show valid mappings. The sibling nodes in the query tree are mapped to sibling nodes in Figure 1a), and to parent-child nodes in Figure 1b). Figure 1c) shows an invalid mapping as the parent-child relationship is not preserved. This kind of distance value differs from the usual edit distance in the following aspects: 1) it does not take the relabeling operation into account, and 2) only one side of the operands can be deleted.

Based on this mapping, a distance value can be defined between two trees. The cost of mapping m is defined as the sum of the vertex mappings related to the query tree:

$$cost(m) = \Sigma_{n \in V(T)} c(n),$$

where

$$c(n) = \begin{cases} C_2, \text{ if } m(n) = \varepsilon \vee m(r(T)) = \varepsilon \\ 0, \text{ if } n = r(T) \wedge m(r(T)) \neq \varepsilon \\ C_1 \ (d(m(n),m(pp(n)) - 1) \text{ otherwise.} \end{cases}$$

In this definition, $pp(n)$ denotes the nearest ancestor of n in the query tree which is mapped to a non-ε element. If the root of the query tree is mapped to ε then $c(n)$ is C_2, otherwise the path from n to $r(T)$ (excluding n and including $r(T)$) contains minimum one vertex mapped to a non-ε value. In this case both $m(n)$ and $m(pp(n))$ are non-ε elements. The $d(\ )$ function denotes the length of path from $m(pp(n)) - m(n)$ in the base tree. As mapping m preserves the parent-child relationship, $m(pp(n))$ is an ancestor of $m(n)$. Thus $d(\ )$ yields a positive integer value. C_1 and C_2 are cost units. C_1 corresponds to gap-lengths between two preserved vertices and C_2 denotes the cost for vertex deletion. In our approach, C_2 is greater than C_1

since the absence of an element means a larger difference than the relocation of the element.

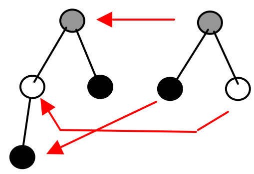


Figure 2

As an example, let the calculation of the mapping cost for Figure 2 stay here. The cost for root mapping is 0. The cost for white node is also 0 (there is no gap in the mapped path). The cost for black node is 1 (one node length gap). The total cost is 0 + 0 + 1 = 1. We remark that in some applications it seems useful to introduce a weight factor in the cost expression. In this case the different edges may have different importance factors.

It can be seen that the distance measure based on this cost value does not meet the requirements of a metric space. The metric distance function should be symmetric while the given cost function is asymmetric. The roles of the query and base trees are distinguished. This corresponds to our intent, as we try to find a best matching sub-tree included in the base tree. The goal is to find a mapping with minimal cost value.

Taking a query tree T_1 with N_q nodes and a base tree T_2 with N_b nodes, the number of potential mappings is $O(N_b! / (N_b-N_q)!)$. Although the ancestor criteria restricts the set of potential mappings, the number of possible enabled mappings is too high. It would be very costly to test all of the possible mappings. Thus some kind of heuristics should be applied to speed up the matching process. In our investigation a variant of the base best-first search method was selected.

The best first search method works on a state-tree. Each node of the tree is assigned to a cost value. The goal is to find the path with the minimal cost value. The best-first search divides the nodes into three distinct groups: the nodes tested (G_1), the nodes ready to be tested (G_2), and the rest (G_3). Initially, G_1 is empty and G_2 contains only the root element. In a loop, the node from the ready state with the best (minimal) cost value is selected to be tested. During the test, the children of the node are evaluated and moved from the G_3 group into the G_2 group. The loop terminates if a leaf node is selected for testing.

In the applied variant, called *m-best-first*, the nodes of the state-tree are assigned not to the vertices but to the vertex mappings of the query tree. Thus each node represents a decision about the mapping function. The state-tree is expanded and traversed in the following way:

1. Generating a label vector for every node. The label vector contains the counter values for the different labels related to the nodes in the

descendant set. This vector works similar to one-grams used in the string distance problem. In the example shown in Figure 3a), the description vector for the root node is *lv*(3,2,1,3), where the first dimension is assigned to the green label, the second to the red label, the third to the blue label and the fourth to the black label.

2. Calculation of the label vectors for the query tree.
3. Selecting maximum K nodes in the base tree with the same label as the root of the query tree and with the first K best distance values regarding the label vectors. The distance value for label vectors is defined by

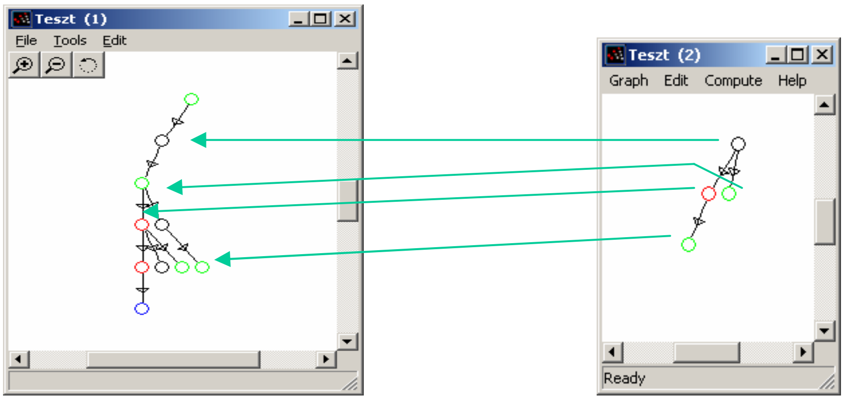
$$d(l_q,l_b) = \sum_j \max(l_{qj}-l_{bj},0)$$

   where l_q belongs to the query tree and l_b to the base tree.
4. Loop on the selected nodes. Let w denote the vertex actually tested. Map the root of the query tree to w. Empty G_2 and G_1.
5. Insert the mapping of w into G_2.
6. Take the element x from G_2 with the lowest cost value. Move x from G_2 into G_1. Disable the other mappings in G_2 from x or to $m(x)$.
7. Test the children vertices of x considering the query tree. For every vertex generate the set of possible mappings. Evaluate these mappings and insert them into G_2.
8. If G_2 is empty, the procedure terminates. The sum of cost values for the selected mappings is the approximation of the best mapping cost value for w, denoted by $C(w)$. Go back to step 4.
9. Return $min\{C(w)\}$ as the approximation of the optimal mapping cost.

The cost of generating the label vectors is $O((N_q+N_b)M)$ as every vertex should be accessed only once. The label vector of a node can be built from the label vectors of its children. In the cost expression M denotes the number of different label values. M corresponds to the length of the label vectors. During the best-first search N_q vertices are tested and expanded. A vertex from the query tree may be mapped to $O(N_b/M)$ vertices in average. As the best-first search is repeated by K times, the cost estimation for the algorithm is $O((N_q+N_b)\cdot M + K\cdot N_q\cdot N_b/M)$. Thus the cost is linear in both N_q and N_b. This cost is a significant reduction compared with the $O(M\cdot N_b! / (N_b-N_q)!)$ value for the brute force search method.

4 Results

The implementation tests show a similar linearity for the computation costs. The test programs are implemented in the Scilab language. The next small example illustrates the cost relations between the brute-force and the heuristic method. The base tree has 10 vertices and is shown in Figure 3a). The query tree has 4 vertices and is shown in Figure 3b). The number of labels is 4. The trees were generated randomly.



a) base tree *b) query tree*

Figure 3

The elements of the best mapping are given in Figure 3 with green arrows. The cost value is only 1. Both methods can detect this optimal mapping but with a very different cost value. Table 1 shows the execution cost values for the investigated methods related to this example query.

Method	**Cost**
Brute-force	69.48 sec
M-best-first	00.08 sec
Selkow	02.46 sec

Table 1

To test the cost values for examples of larger tree sizes, a test run was implemented with values $N_q = 12$, $N_b \in [15..500]$. The cost values are shown in Figure 4.

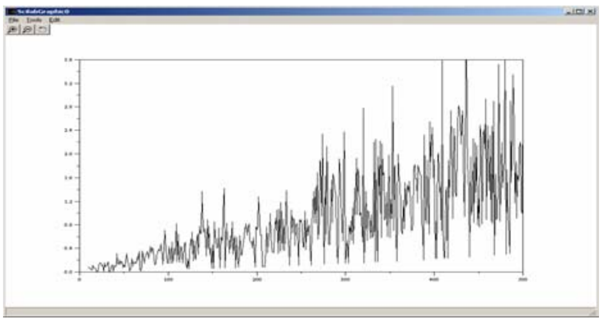


Figure 4

The x-axis denotes the N_b value, while the y-axis shows the computation cost (where the maximum value is 3.6 sec). The trees were generated randomly. In Figure 4 the linearity of the cost function is well demonstrated.

Selkow's algorithm calculates the editing distance between two forests of ordered trees. The measured editing distance is very similar to the editing distance of strings. It is simplified to a one-level comparison, so edition is necessary each time the root nodes are not identical. As an ordered tree is a special case of an unordered tree, Selkow's algorithm can be used for unordered trees as well. It is a basic algorithm which needs a tremendous computation power of $O(N^2*M^2)$, where *N* and *M* are the node numbers of the trees, to find the editing order of the least cost. In case of unordered trees the algorithm has to take each order of the tree in account, so the computation cost will grow by $O(2^N*2^M)$ for the repetitions on each possible permutation of the trees.

The generalization of the algorithm to forests is inevitable due to the fact, that the algorithm is recursive and that deleting the root node of a tree will result in a forest. We have tested an implementation of Selkow's algorithm on small trees. Running our implementation of the algorithm with the trees shown on Figure 3 we got the editing distance of 10 units, considering 1 as the cost of each edit operation. The time needed to complete the calculation in the same environment was 2.463 sec as is shown in Table 1. Running the algorithm with randomly generated trees will show a very noisy cost function, however, the limits should show the $O(N^2*M^2)$ characteristics.

In the frame of the investigation, the proposed algorithm was compared not only with the brute-force method but with the method based on a wider base best-first searching. This algorithm allows the extension of the search tree with nodes where the related vertices have been already visited in some of the previous steps. The only restriction is to ensure the parent-child mapping direction. If m^1 and m^2 are two mapping nodes in the search tree, then m^2 is a child node of m^1 only if the corresonding vertices in the query tree own the same parent-child relationship. Despite this natural restriction, this method yields in a wider and larger search tree but can provide a better approximation. Table 2 shows some typical results of the comparison.

	T	P	T	P	T	P	T	P
m-best-first	0.8	12	0.4	16	0.3	13	0.2	14
base best-first	42	12	40	16	18	12	18	10

Table 2

Based on the results of the tests, the execution cost for the base best-first algorithm is about one order of magnitude larger than the cost for the porposed modified best-first approximation method.

Conclusions

The approximate sub-tree search for trees with edit distance metric is an NP-complete problem. To establish a fast search engine an efficient heuristic method should be implemented. The proposed modified best-first method provides a $O((N_q+N_b)\cdot M + K\cdot N_q\cdot N_b/M)$ cost function with simple implementation details. According to our test results, realized with smaller trees where the brute-force algorithm could be tested, the yielded results are a good approximation of the global optimum values. Based on the results of the tests, the execution cost for the base best-first algorithm is about one order of magnitude larger than the cost for the porposed modified best-first approximation method.

References

[1] P. Bille: Tree Edit Distance, Alignment Distance and Inclusion, IT University of Copenhagen, Technical Report Series TR-2003-23, ISSN 1660-6100, March 2003

[2] Barnard, Clarke, Duncan: Tree-to-Tree Correction for Document Trees, Technical Report 95-372, Queen's University Canada, January 1995

[3] Tai: The Tree-to-Tree Correction Problem, Journal of the Association for Computing Machinery (JACM), 26:422-433, 1979

[4] Zhang, Shasha: Simple fast algorithms for the editing distance between trees and related problems, SIAM Journal of Computing, 18:1245-1262, 1989

[5] Klein: Computing the edit-distance between unrooted ordered trees, in Proceedings of the 6th annual European Symposium on Algorithms (ESA) 1998, pages 91-102, Springer Verlag 1998

[6] Chen: New algorithm for ordered tree-to-tree correction problem, Journal of Algorithms, 40:135-158, 2001

[7] Zhang, Statman, Shasha: On the editing distance between unordered labeled trees, Information Processing Letters, 42:133-139, 1992

[8] Shasha, Zhang: Fast algorithms for the unit cost editing distance between trees, Journal of Algorithms, 11:581-621, 1990

[9] Ukkonen: Finding approximate patterns in strings, Journal of Algorithms, 6:132-137, 1985

[10] Landau, Vishkin: Fast parallel and serial approximate string matching, Journal of Algorithms, 10:157-169, 1989

[11] Selkow: The tree-to-tree editing problem, Information Processing Letters, 6(6):184-186, 1977

[12] Chawathe: Comparing hierarchical data in extended memory, in Proceedings of VLDB, pages 90-101, 1999

[13] Kilpeläinen, Mannila: Ordered and unordered tree inclusion, SIAM Journal of Computing, 24:340-356, 1995

[14] Kilpeläinen: Tree Matching Problems with Applications to Structured Text Databases, PhD Thesis, University of Helsinki, Department of Computer Science, 1992

[15] Richter: A new algorithm for the ordered tree inclusion problem, in Proceedings of the 8th Annual Symposium on Combinatorial pattern Matching (CPM), in Lecture Notes of Computer Science (LNCS), volume 1264, pages 150-166, Springer 1997

[16] Chen: More efficient algorithm for ordered tree inclusion, Journal of Algorithms, 26:370-385, 1998

[17] Matousek, Thomas: On the complexity of finding iso- and other morphisms for partial k-trees, Discrete Mathematics, 108:343-364, 1992

[18] Torsello, Hancock: Computing approximate tree edit distance using relaxation labeling, Pattern Recognition Letters 2003, PII: S0167-8655(02)00255-6, 2002

[19] Bunke, Kandel: Mean and maximum common subgraph of two graphs, Pattern Recognition Letter 21, pp. 163-168, 2000

[20] Barrow, Burstall: Subgraph isomorphism, matching relational structures and maximal cliques, Information Processing Letters 4, pp. 83-84, 1976

[21] Pellilo et al.: Matching hierarchical structures using association graphs, IEEE PAMI 21, pp. 1105-1120, 1999

[22] Wang et al.: An Algorithm for Finding the Largest Approximately Common Substructures of Two Trees

[23] Khanna, Motwani, Yao: Approximation algorithms for the largest common subtree problem, Technical Report, Stanford University, 1995

[24] Akutsu, Halldórsson: On the Approximation of Largest Common Subtrees and Largest Common Point Sets, Science Institute University of Iceland, October 1997

[25] Shasha et al.: AtreeGrep – Approximate Searching in Unordered Trees, in Proceedings of SSDBM 2002, Edinburgh, July 2002, pp. 89-98