A similarity algorithm of trees with complicated labels

Wang Jihua
College of Information Science and Engineering,
Shandong Normal University,
Shandong Provincial Key Laboratory for Distributed Computer Software Novel Technology,
Jinan, China
e-mail: jihuaaw@126.com

Liu Yiliang
College of Information Science and Engineering,
Shandong Normal University,
Shandong Provincial Key Laboratory for Distributed Computer Software Novel Technology,
Jinan, China
e-mail: ytyl_liu@aliyun.com

***Abstract*—A tree with complicated labels can widely represent objects in the real world. In this paper, the sum of matched pair similarity is used only to measure the similarity of such trees, while omitting inserted nodes and deleted nodes in the maximum mapping conditions of edit operations. The principle of path decompositions of trees is resolved and illustrated to be based essentially on the same mapping requirements. The leftmost path decomposition is implemented as one example of the algorithms for the similarity of trees with complicated labels.**

Keywords-similarity of trees; tree edit operation; mapping conditions; complicated labels; path decompositions

I. Introduction

Comparing trees has been applied extensively in many fields such as web page retrieval, organization management, product similarity, computational biology, natural language processing, and various others over the years[1,2].
The tree edit distance between two ordered labeled trees is the most common method to measure the similarity of trees. Computation of this distance focuses on finding the minimum cost of a sequence of deletions, insertions, and re-labels in both trees so that they can be transformed into isomorphic trees.

The edit distance method originated from the string-to-string correction problem proposed by Wagner in 1974[3]. It was then developed into the tree-to-tree correction problem by Tai in 1979, with a time complexity of $O(n^6)$[4]. In 1989, K. Zhang proposed the simple fast algorithms for editing the distance between trees to reduce the complexity to $O(n^4)$[5]. In 1998, Klein used the heavy path decomposition algorithm to make the calculation cost of edit distance $O(n^3\log(n))$[6]. In 2005, S. Dulucq's method changed the complexity to $O(n^2\log^2(n))$[7]. In 2009, E. D. Demaine analyzed the family of decomposition strategies and presented the optimal decomposition algorithm whose time complexity is $O(n^3)$[8].
At present, the tree similarity can be computed by edit distance, and the trees in the previously mentioned papers are rooted, ordered, and labeled with a single character, in which the two nodes in both trees are clearly equal. In reality, the nodes of trees as abstract representations of practical problems cannot just be labeled with simple symbols; the nodes need be expressed by more complicated symbols. Similar trees should also be measured only through the mapping pairs or the relabeled and equal nodes in isomorphic trees without considering the deleted and inserted nodes.

In this paper, the similarity of the mapping pairs is given to measure the similarity of trees with complicated labels. Both methods of edit distance and mapping pair similarity are based on the same mapping theorem. The former calculates the cost of three kinds of edit operations while the latter computes the sum of the similarity of mapping pairs to measure the similarity of trees.

Our roadmap: In Section 2, the formal presentation of a tree is given with the nodes labeled using complicated symbols such as functions and hierarchical codes, and the definition of similarity between both nodes. In Section 3, the mapping theorem and the similarity between such trees are presented. The similarity of the algorithms of specific trees based on path decompositions and the mapping theorem is resolved in Section 4. The implementation of the leftmost path decomposition as one example of decomposition algorithm and its complexity are discussed in Section 5. Final conclusions are presented in Section 6.

II. Trees with Complicated Labels and the Similarity of Nodes

The information of a complex object can be clearly demonstrated by a tree structure, in which the nodes need to be labeled with complicated symbols. These symbols contain more messages than a single character; in this case, the cost of re-labeling or deleting is inaccurate to describe the differences of one node from another.

In Figure 1, two images are modeled by the same tree structure. The values of their flower nodes are (R87 G196 B60) and (R67 G188 B77), respectively. Their shapes and structures look similar, but their node labels are not the same.

In Figure 2, two nodes in function trees have the same type but different parameters. Thus, determining the similar degree of the nodes using only edit operations such as changing, inserting, or deleting is difficult.

The above nodes are similar but not identical, and their edit cost cannot be confirmed simply by 1 or 0. The similarity of two nodes in the range of [0, 1] contrary to the edit cost is defined.

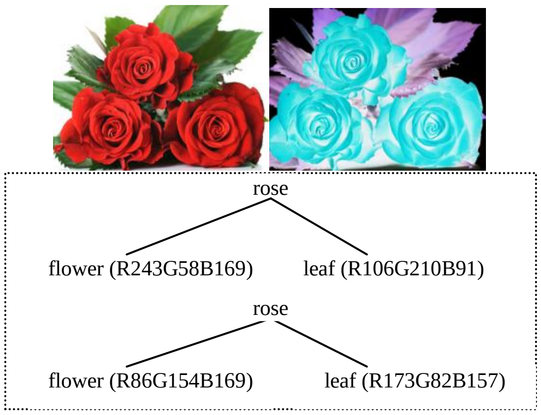

Figure 1 Tree structures of the image

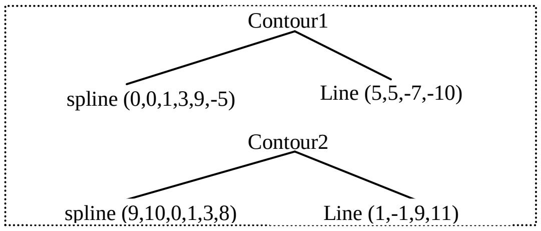

Figure 2 Function trees

Without loss of generality, let the label of every node be a function expression f(), and let T and T\` be trees. T[i] denotes the node of T whose position is i in the given ordering for nodes of T, such as pre-order or post-order. |T|=m is the number of nodes of T. T\`[j] and |T\`|=n are similarly defined.

Suppose that the label of a node T[i] is f(i), and the label of a node T\`[j] is k(j). Thus, the similarity of T[i] and T\`[j] is defined as S_NODE(i, j)=1-D((f(i), k(j))/A, in which D is a kind of mathematical distance such as Euclidean distance, and A is a specified constant.

If the node is null with notation ø, we set S_NODE(i,j)∈[0,1], S_NODE(ø,j)=0 and S_NODE(i,ø)=0, S_NODE(ø, ø)=0.

III. A Mapping between Trees and the Similarity of Trees

A mapping between trees produced by edit operations is a graphical specification of what edit operations apply to each node in the two trees. Consider the diagram of a mapping in Fig. 3.

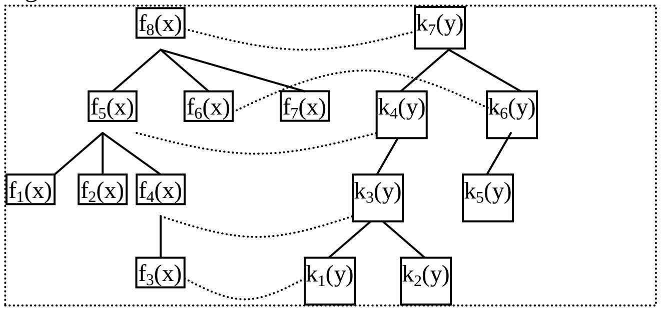

Figure 3 Mapping between trees

A dotted line from T[i] to T\`[j] indicates that 0≦S_NODE(i, j)≦1 and (T[i], T\`[j]) or (i, j) are mapping pairs. Nodes of T and T\` not touched by dotted lines remain unchanged and do not affect comparing trees, where S_NODE(i, ø)=0 and S_NODE(ø, j)=0. Such a diagram is called a mapping expressed as a triple (M, T, T\`), where M is any set of mapping pairs of integers (i, j) from T to T\`, 1≤i≤|T|, 1≤j≤|T\`|, satisfying the following theorem:

For any pair of (i_1, j_1) and (i_2, j_2) in M:

(a) $i_1 = i_2$ if and only if $j_1 = j_2$ (one-to-one),

(b) i_1 is to the left (or right) of i_2 if and only if j_1 is to the left (or right) of j_2 (sibling order), and

(c) i_1 is an ancestor (or descendant) of i_2 if and only if j_1 is an ancestor (or descendant) of j_2 (ancestor order).

In the condition that tree structures keep intact without inserting and deleting operations, the accumulative similarity of mapping pairs in accordance with the above conditions can adequately reflect the similarity of two trees. The mapping similarity S_MAP(M) or S_MAP(T,T\`) between T and T\` is defined as the sum of the similarity of mapping pairs:

$$S\_MAP(M) = S\_MAP(T,T`) = \sum_{(i,j)\in M} S\_NODE(i,j) \quad (1)$$

In any sequence of edit operations of changing, inserting, and deleting that transforms T into T\`, a mapping M exists such that the cost (M) is minimum, and the nodes with inserting and deleting operations contribute little to the similarity of trees. The maximum number of mapping pairs mean the maximum S_MAP(M) and the minimum cost (M), and the trees are most similar if every S_NODE(i, j) in M is equal.

Thus, the similarity of two trees S_TREE(T, T\`) is determined by a maximum S_MAP(M) from T to T\`, which should be equivalent to a minimum cost mapping under conditions that all insert, delete, and change operations cost the same value as one. The edit distance d(T, T\`) from T to T\` is defined as the minimum cost of one in all sequences of edit operations, or d(T, T\`) is the min cost(M). Thus, d(T, T\`) is in fact consistent with the similarity of trees S_TREE(T, T\`).

To compute S_TREE(T, T\`) and the set of mapping pairs M, which is the largest common substructure of both compared trees, the maximum mapping similarity needs to be found, i.e.,

$$S\_TREE(T,T`) = MAX(S\_MAP(M)) = MAX(\sum_{(i,j)\in M} S\_NODE(i,j)) \quad (2)$$

IV. Analysis of the Computation of Similarity of Trees on the Mapping Theorem epare

This paper still uses dynamic programming to compute S_TREE(T, T\`) by decomposing two trees according to the mapping theorem.

The nodes in T and T\` are post-order numbered below. Let T(i_1:i_2) denote the portion of T consisting of nodes T[i_1], T[i_1+1], …T[i_2]. When T[i_1] is the leftmost leaf node of the tree rooted at T[i_2], T(i_1:i_2) is called a sub-tree denoted as LT(i_2), and T[i_1] is represented as L(i_2). T\`($j_1$:$j_2$), LT\`(j_2), and L(j_2) are similarly defined.

Let S_TREE(i, j) be the similarity of tree T(1:i) and tree T\`(1:j). M is the set of mapping pairs from T(1:i) to T\`(1:j). Thus, $S\_TREE(i,j) = MAX(\sum_{(i,j)\in M} S\_NODE(i,j))$, S_TREE(T, T\`)= S_TREE(|T|, |T\`|).

Let S_TREE(LT(i),LT\`(j)) or S_TREE((L(i):i, L(j):j) be the similarity of sub-tree T(L(i):i) and sub-tree T(L(j):j), which is also called SUB_TREE(i,j). $M_L$ is the set of mapping pairs from LT(i) to LT\`(j).

$$\begin{aligned} SUB\_TREE(i,j) &= S\_TREE(L(i):i, L(j):j) \\ &= MAX(\sum_{(i,j)\in M_L} S\_NODE(i,j)) \end{aligned} \quad (3)$$

In dynamic programming, at least one of the following three cases must hold:

Case 1: T[i+1] is not touched by a dotted line, then S_TREE(i+1,j+1)=S_TREE(i,j+1) +S_NODE(i+1, ∅) =S_TREE(i,j+1).

Case 2: T\`[j+1] is not touched by a dotted line, then S_TREE(i+1,j+1)=S_TREE(i+1,j) +S_NODE(∅, j+1) =S_TREE(i+1,j).

Case 3: T[i+1] and T\`[j+1] are touched by the same dotted line (i+1, j+1). If (i+1,j+1) does not conflict with M according to the mapping conditions, then S_TREE(i+1, j+1)= S_TREE(i, j)+ S_NODE(i+1, j+1); otherwise, the formula must be revised under the following circumstances.

In the above three determined cases: S_TREE(i+1,j+1) =MAX(S_TREE(i,j)+ S_NODE(i+1,j+1), S_TREE(i+1,j), S_TREE(i,j+1)).

In case 3, both the key and the challenge lie in the computation whether (i+1, j+1) is satisfied with the mapping conditions and in the set of mapping pairs M, which is determined by the statuses of the descendants and siblings of the current node in the post-order. The paths are the most critical in determining mapping pairs. In Figure 4, (i_2, j_2) in M hold under the condition of (i_1, j_1) in M only by comparing path1 and path\`1, but not path1 and path\`2. (i_2, $j\`_2$) in M hold only by comparing path1 and path\`2.

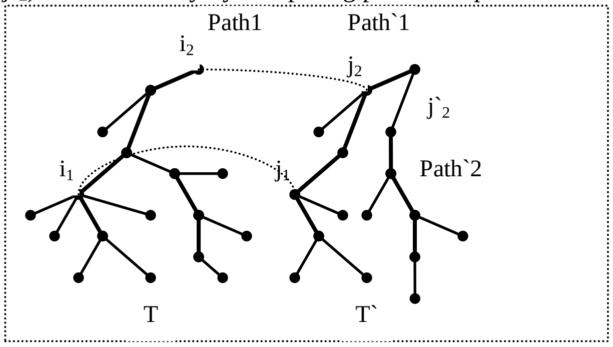

Figure 4 Paths and ancestry

Hence, whether (i+1, j+1) meets the mapping conditions in comparing T and T\` under MAX(S_MAP(M)) can be completed by the contrast of one path in T with any path in T\`. Thus, only two cases need be solved:

1) Node i+1 and node j+1 are in the two compared paths simultaneously.
2) Node i+1 and node j+1 are not in the two compared paths simultaneously.

The following basic situations contribute to analyze the problem further, as shown in Figure 5.

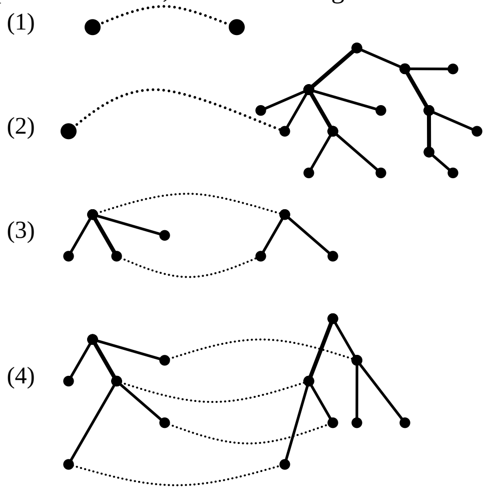

Figure 5 Typical situations

A tree with at least one node is called a single-node tree, a tree that is a two-level tree with a parent node and son nodes is called a basic tree, and a tree with three levels or more is called a complex tree.

When T and T\` are basic trees, if the parent node i of T maps with one of son(leaf) nodes 1,2,…,j-1 of T\`, (i,1) or (i,…) or (i,j-1) in M can hold, and (i,j) in M cannot hold, and S_TREE(T,T\`)=MAX(S_NODE(i,1), S_NODE(i,2),…, S_NODE(i,j-1)), which is same as the second situation; if the parent node i of T maps with the parent node j of T\`, (i,j) in M can hold, and S_TREE (T,T\`) = MAX(S_TREE(i-1, j-1)+S_NODE(i,j), S_TREE(i,j-1), S_TREE(i-1,j)) and vice versa.

When T and T\` are complex trees, firstly, the similarity S_TREE(i,j) of the basic trees rooted at the penultimate nodes of the two compared paths as sub-tree 1 and 1\` in T and T\` is computed like the third situation. Then, the next nodes i+1 and j+1 are in basic tree 2 and 2\`, respectively, whose roots are the siblings of the roots of the first sub-trees. Whether (i+1,j+1) in M holds depends only on comparing tree 2 and 2\` according to their descendant mapping conditions, which is the same as the computation of tree 1 and 1\` due to SUB_TREE(i+1,j+1)= S_TREE (T(L(i+1) :(i+1)),T\`((L(j+1):(j+1))), so S_TREE(i+1, j+1)=S_TREE(L(i+1)-1,L(j+1)-1)+SUB_TREE(i+1,j+1).

Thus, computations are done recursively until the starting nodes of the paths. This solves the problems in case 3.

Every path in trees hangs sub-trees that root at the sons of the nodes in paths. Thus, trees are decomposed into sub-trees by paths, and sub-trees are to be split recursively. Any path decomposition to a tree such as the heavy path or the leftmost path aims at the valid establishment of mapping pairs. Well-designed path decompositions can reduce the complexity of similarity algorithms by reducing the number of sub-trees. Therefore, the decomposition method should be designed based on the features of tree structures on specific issues.

V. Implementation of an Algorithm of the Similarity of Trees with Complicated Labels

Let Ø be a null tree, S_TREE(T, Ø)=0, S_TREE(Ø, T\`)=0， S_TREE(Ø, Ø)=0 as initial conditions of the following computations.

The nodes in T and T\` are post-order numbered, which are respectively placed in two structure-type arrays T[m] and T\`[n] sequentially by a child-sibling-link. Every element in T[m] and T\`[n] contains four fields: {Code; /* post-order number*/ Data; /*complicated label*/ Child; /*the first child pointer*/ Sibling; /*the left sibling pointer*/ }.

All root nodes of sub-trees hanging in the leftmost paths[5] in T are put in the array P[m\`], Similarly P\`[n\`] is in T\`.

The algorithm for the similarity of trees with complicated labels consists of the following key steps:

```
1) B=S_TREE(L(P[i`]):i, L(P`[j`]):(j-1));
2) C=S_TREE(L(P[i`]):(i-1), L(P`[j`]):j);
3) if L(i)=L(P[i`]) and L(j)=L(P`[j`])
   {E=S_TREE(L(P[i`]):(i-1),L(P`[j`]):(j-1))+ S_NODE(i, j);
   SUB_TREE(i,j)=S_TREE(L(P[i`]):i, L(P`[j`]):j)=MAX{E,B,C}; /*SUB_TREE(i,j) put in array SS[i][j]*/}
4) else
   {F=S_TREE(L(P[i`]):(L(i)-1),L(P`[j`]):(L(j)-1))+ SUB_TREE(i,j);
   S_TREE(L(P[i`]):i,L(P`[j`]):j)=MAX{F,B,C}}
5) S_TREE(T,T`)=S_TREE(P[m],P`[n`]);  /*S_TREE(i,j) put in array TT[i][j]*/
```

The similarity between T and T\` in Figure 3 was computed by the above algorithm. For simplicity, the similarity of the roots of T and T\` S_NODE(8, 7)=1, S_NODE(ø, j)=0, S_NODE(i, ø)=0, S_NODE(i,j)=1-(m+n-i-j)*0.01, P[m\`]={2,4,6,7,8}, P\`[n\`]={2,6,7} is assumed.

The results in array SS[i][j] and TT[i][j] of applying the algorithms to T and T\` are shown in Table Ⅰ.

The complexity of the algorithm is primarily affected by the number of sub-trees hanging in paths and the number of nodes in every sub-tree, which is O(n^4) in the worst case.

VI. Conclusion

Compared with the edit distance, the similarity of the sum of matched nodes is a better way of comparing trees with complicated labels, which underlines the matched nodes and omits the other inserted and deleted nodes. This method conforms to the intuitive thinking of people in comparing things, and is more concise and realistic.

Table Ⅰ Matrix of the similarity of sub-trees

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0.87	0.88	0.89	0.9	0.91	0.92	0.93
2	0.88	1.76	1.76	1.76	1.82	1.83	1.83
3	0.89	1.78	1.78	1.78	2.69	2.7	2.7
4	0.9	1.79	1.82	1.84	2.7	3.64	3.64
5	0.91	1.79	2.72	2.76	2.76	2.76	4.61
6	0.92	1.84	2.72	2.76	3.72	3.73	4.61
7	0.93	1.86	2.72	2.76	3.73	3.74	4.61
8	0.94	1.86	2.82	3.69	3.73	3.74	4.74

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0.87	0.88	0.89	0.90	0.91	0.92	0.93
2	0.88	0.89	0.90	0.91	0.92	0.93	0.94
3	0.89	0.90	0.91	0.92	0.93	0.94	0.95
4	0.90	0.91	1.82	1.84	0.94	1.88	1.90
5	0.91	0.92	2.72	2.76	0.95	1.90	4.61
6	0.92	0.93	0.94	0.95	0.96	0.97	0.98
7	0.93	0.94	0.95	0.96	0.97	0.98	0.99
8	0.94	0.95	2.82	3.69	0.98	1.96	4.74

Acknowledgment

This work was supported by the Shandong Provincial Science and Technology Development Plan (2010G0020807), and the Shandong Provincial Key Laboratory Project.

References

[1] P. Bille. A survey on tree edit distance and related problems [J]. Theoretical computer science, 337: 217–239, 2005.

[2] J. Jansson, Z. Peng. Algorithms for finding a most similar sub-forest[C]. Proceeding of the 17th annual symposium on combinatorial pattern matching (CPM2006), 4009: 377–388, 2006.

[3] R. A. Wagner. The string-to-string correction problem[J]. Journal of the Association for Computing Machinery, 21(1): 168–173, 1974.

[4] K. C. Tai. The tree-to-tree correction problem[J]. Journal of the Association for Computing Machinery, 26(3): 422–433, 1979.

[5] Kaizhong Zhang, Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems[J]. Society for Industrial and Applied Mathematics, 18(6): 1245–1262, 1989.

[6] P. N. Klein. Computing the edit-distance between unrooted ordered trees[J]. ESA'98, LNCS 1461: 91–102, 1998.

[7] S. Dulucq, H. Touzet. Analysis of tree edit distance algorithms[J]. Journal of Discrete Algorithms, 3(2–4): 448–471, 2005.

[8] E. D. Demaine, S. Mozes, B. Rossman, etc. An optimal decomposition algorithm for tree edit distance[J]. ACM Transactions on Algorithms, 6(1): 2:1–2:19, 2009.