An Efficient Bottom-Up Distance between Trees

Gabriel Valiente
Technical University of Catalonia
Department of Software
E-08034 Barcelona, Spain
valiente@lsi.upc.es

Abstract

A new bottom-up distance measure for labeled trees, which is based on the largest common forest of the trees and has the threefold advantage of independence of particular edit costs, low complexity, and coverage of ordered and unordered trees, is introduced and related in this paper with other distance measures published in the literature. Algorithms for computing the bottom-up distance in time linear in the number of nodes are given in full detail.

Key words design and analysis of algorithms, combinatorial problems, graph algorithms, pattern matching, tree pattern matching, tree isomorphism, subtree isomorphism, edit distance, metric space, largest common forest

1 Introduction

Trees count among the most common combinatorial structures in computer science, with a great variety of application areas. The problem of comparing and matching trees finds application in areas as diverse as compiler design [1], term rewriting [22], graph transformation [17], symbolic computation [11], information retrieval [15], image processing [18], pattern recognition [16], signal processing [5], molecular biology [20], chemistry [24], and many others.

Related to the problem of comparing trees is the tree pattern matching problem, which is analogous to the problem of string pattern matching. A string pattern matcher is a program that takes as input a pattern and a text and produces as output the locations in the text at which the pattern appears as a substring. In tree pattern matching, the pattern and the text are rooted trees, and the pattern matching problem consists of finding all subtrees of the text that are isomorphic to the pattern or, more in general, all subtrees of the text that are isomorphic to a subtree of the pattern. Since these common subtrees between the pattern and the text indicate the extent to which pattern and text coincide, tree pattern matching is also useful for comparing trees and, as a matter of fact, several distance measures for comparing trees have been proposed [14, 19, 25, 26, 28, 30] which essentially differ on the underlying notion of subtree.

In this paper, a new bottom-up distance measure between rooted, labeled trees, which is based on the largest common forest of the trees, is introduced and related with other distance measures published in the literature: edit distance [25], top-down distance [19, 30], alignment distance [14], and isolated-subtree distance [26]. The bottom-up distance, which actually is a particular case of the isolated-subtree distance, is shown to coincide with the top-down distance only for isomorphic trees, extending thus the hierarchy among tree edit distances developed in [29].

An advantage of the bottom-up distance is that no particular tree edit operations together with their costs need to be defined. Another advantage is the low complexity: it can be computed in time linear in the size of the trees, on rooted, labeled, ordered trees of unbounded degree. A further advantage is the fact that the bottom-up distance has the same low complexity for unordered trees.

The following notation will be used in the rest of this paper. For a rooted ordered tree T, let n denote the number of nodes of T, let $t[i]$ denote the node of T whose position in the preorder traversal of T is i, with $1 \leqslant i \leqslant n$, and let $par(i)$ denote the preorder number of the parent of node $t[i]$. Let also $T[i]$ denote the subtree of T rooted at node $t[i]$. Node $t[i]$ is said to be to the left of node $t[j]$ if $i < j$. The rightmost node of $T[i]$ is, then, the node of $T[i]$ with largest preorder number. Recall that an *ordered tree* is a rooted tree that has been embedded in the plane, that is, such that the relative order of the children is fixed for each node. A *forest* is a set of zero or more disjoint rooted trees. Recall also that two trees T_1 and T_2 are isomorphic, denoted by $T_1 \cong T_2$, if they either are both empty or have identical root label and the subtrees rooted at corresponding children of their roots are isomorphic.

Only ordered trees will be considered in the rest of the paper, in order to ease the comparison of the bottom-up distance with previous distance measures. However, the notation, algorithms, and results about the bottom-up distance extend in a straightforward way to unordered trees, and the algorithms are given for ordered trees and for unordered trees as well.

The rest of the paper is organized as follows. Previous distance measures between ordered trees are reviewed in Section 2. In Section 3, the new bottom-up measure between ordered trees is introduced and its relationship with the other distance measures is established. Algorithms for computing the bottom-up distance are given in Section 4 in

full detail. Finally, Section 5 presents some conclusions.

2 Distance Measures between Trees

Previous distance measures for comparing trees are essentially the generalization to trees of the (weighted) edit distance between strings [2, 6, 13, 23, 32]. The distance between two trees is given by the shortest or the least-cost sequence of elementary edit operations (insertion, substitution, and deletion of labeled nodes) that allow to transform one tree into the other. Despite their original definition in terms of elementary edit operations, distance measures between trees can also be stated in terms of mappings.

2.1 Edit Operations, Edit Distance, and Mappings

A *mapping* establishes a one-to-one correspondence between the nodes of two ordered trees which preserves the order of siblings and ancestors. Mappings were introduced in [25] in order to describe how a sequence of edit operations transforms a tree into another one.

Definition 1. *A mapping from a tree T_1 to a tree T_2 is a set M of ordered pairs of integers (i, j), $1 \leqslant i \leqslant n_1$, $1 \leqslant j \leqslant n_2$, satisfying the following conditions, for all $(i_1, j_1), (i_2, j_2) \in M$:*

- *$i_1 = i_2$ if, and only if, $j_1 = j_2$;*
- *$t_1[i_1]$ is to the left of $t_1[i_2]$ if, and only if, $t_2[j_1]$ is to the left of $t_2[j_2]$;*
- *$t_1[i_1]$ is an ancestor of $t_1[i_2]$ if, and only if, $t_2[j_1]$ is an ancestor of $t_2[j_2]$.*

As a matter of fact, a mapping from a tree T_1 to a tree T_2 describes the edit operations that allow to transform T_1 into T_2. A node $t_1[i]$ with no pair $(i, j) \in M$ is deleted from T_1, a pair $(i, j) \in M$ indicates the substitution of node $t_1[i]$ by node $t_2[j]$, and a node $t_2[j]$ with no pair $(i, j) \in M$ is inserted into T_2.

Let M be a mapping from a tree T_1 to a tree T_2, let S be the set of pairs $(i, j) \in M$ with $t_1[i]$ and $t_2[j]$ having different label, let D be the set of nodes $t_1[i]$ with no pair $(i, j) \in M$, and let I be the set of nodes $t_2[j]$ with no pair $(i, j) \in M$. The cost of mapping M is given by $|S|p+|I|q+|D|r$, where p is the cost of a non-identical substitution, q is the cost of an insertion, r is the cost of a deletion, and the cost of identical substitutions is 0. It is usual to assume unit cost [21, 29], where the cost of identical substitutions is 0 and the cost of all other edit operations (deletions, non-identical substitutions, and insertions) is 1.

Example 1. *In the mapping M from tree T_1 to tree T_2 of Figure 1, nodes $t_1[1], t_1[7], t_1[8], t_1[9], t_1[10]$ are mapped to nodes $t_2[1], t_2[5], t_2[6], t_2[7], t_2[8]$, respectively, nodes $t_1[2], t_1[3], t_1[4], t_1[5], t_1[6]$ are deleted from T_1, and nodes $t_2[2], t_2[3], t_2[4]$ are inserted into T_2. Since there are $n_1 - |M| = 10 - 5 = 5$ node deletions, no non-identical substitutions, and $n_2 - |M| = 8 - 5 = 3$ node insertions, the cost of the mapping is $3q + 5r$. Under the assumption of unit cost, the cost of mapping M is 8.*

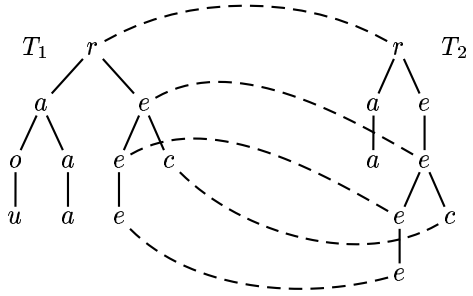


Figure 1. Sample mapping.

The *edit distance* between ordered trees was introduced in [25], and an algorithm was given in [31] to compute the distance between two ordered trees T_1 and T_2 in $O(n_1 n_2 \log n_1 \log n_2)$ time. The edit distance problem for unordered trees was shown to be NP-complete in [33].

Definition 2. *The edit distance from tree T_1 to tree T_2 is the cost of a least-cost mapping between T_1 and T_2.*

2.2 Alignment Distance

Other distance measures between trees impose further conditions on mappings. The *alignment distance* between ordered trees was introduced in [14], where an algorithm was given to compute the distance between two ordered trees T_1 and T_2 in $O(n_1 n_2 \log n_1 \log n_2)$ time, and where the alignment problem between unordered trees was shown to be MAX SNP-hard.

Definition 3. *A mapping is an alignment if it can be extended to an isomorphism between the underlying unlabeled trees after inserting a least number of nodes into the two trees. The alignment distance from tree T_1 to tree T_2 is the cost of a least-cost alignment mapping between T_1 and T_2.*

2.3 Isolated-Subtree Distance

The *isolated-subtree distance* between ordered trees was introduced in [26], where an algorithm was given to compute the distance between two ordered trees T_1 and T_2 in $O(n_1 n_2)$ time. Isolated-subtree mappings always map disjoint subtrees to disjoint subtrees.

Definition 4. *A mapping M from a tree T_1 to a tree T_2 is isolated-subtree if it satisfies the following condition, for all $(i_1, j_1), (i_2, j_2) \in M$:*

- *the rightmost node of $T_1[i_1]$ is to the left of $t_1[i_2]$ if, and only if, the rightmost node of $T_2[j_1]$ is to the left of $t_2[j_2]$.*

The isolated-subtree distance from tree T_1 to tree T_2 is the cost of a least-cost isolated-subtree mapping between T_1 and T_2.

2.4 Top-Down Distance

The *top-down distance* between ordered trees was introduced in [19], and an algorithm was given in [30] to compute the distance between two trees T_1 and T_2 in $O(n_1 n_2)$ time. In a top-down mapping, the parents of nodes in the mapping are also in the mapping.

Definition 5. *A mapping M from a tree T_1 to a tree T_2 is top-down if it satisfies the following condition, for all i, j such that $t_1[i]$ and $t_2[j]$ are not the root of T_1 and T_2, respectively:*

- *if $(i, j) \in M$ then $(\text{par}(i), \text{par}(j)) \in M$.*

The top-down distance from tree T_1 to tree T_2 is the cost of a least-cost top-down mapping between T_1 and T_2.

Example 2. *The mapping of Figure 2 between the trees of Example 1 is an alignment mapping and also an isolated-subtree mapping and a top-down mapping.*

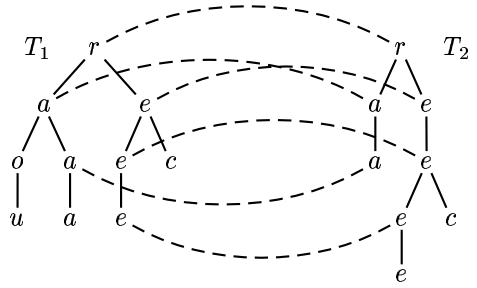


Figure 2. Sample alignment mapping which is also isolated-subtree and top-down.

2.5 Relationship between Distance Measures

These distance measures were related to each other in [29], establishing thus a hierarchy among tree edit distances.

Lemma 1 (Wang, Zhang, 2001). *Assume that the cost of identical substitutions is 0 and the cost of all other edit operations is 1.*

1. *Let T_1 and T_2 be two trees and let M be an isolated-subtree mapping from T_1 to T_2. Then M is also an alignment mapping from T_1 to T_2.*
2. *Let T_1 and T_2 be two trees and let M be a top-down mapping from T_1 to T_2. Then M is also an isolated-subtree mapping from T_1 to T_2.*

Notice that every alignment is, by definition, a mapping. This yields a hierarchy among distance measures [29], where top-down mappings are isolated-subtree mappings which, in turn, are alignments which, in turn, are mappings.

In the next section, the bottom-up distance is introduced and its relationship with previous tree edit distances is established, completing thereby the previous hierarchy.

3 A Bottom-Up Distance between Trees

The *bottom-up distance* between rooted trees was introduced in [28], where an algorithm was given to compute the distance between two trees T_1 and T_2, either ordered or unordered, in expected $O(n_1 + n_2)$ time. The bottom-up distance between two non-empty rooted trees T_1 and T_2 is equal to $1 - f / \max(n_1, n_2)$, where f is the size of a largest common forest of T_1 and T_2. The algorithm for computing the bottom-up distance is based on a simple and efficient bottom-up algorithm for finding all common rooted subtrees in a forest in expected $O(n_1 + n_2)$ time [27], which is improved in Section 4 to take worst-case $O(n_1 + n_2)$ time.

A new definition of bottom-up distance is given next which is based on mappings. A bottom-up mapping is an isolated-subtree mapping in which the children of nodes in the mapping are also in the mapping.

Definition 6. *An isolated-subtree mapping M from a tree T_1 to a tree T_2 is bottom-up if it satisfies the following condition:*

- *if $(i, j) \in M$ then $(i_1, j_1), \ldots, (i_k, j_k) \in M$*

where $t_1[i_1], \ldots, t_1[i_k]$ are the children of node $t_1[i]$ and $t_2[j_1], \ldots, t_2[j_k]$ are the children of node $t_2[j]$. The bottom-up distance from tree T_1 to tree T_2 is the cost of a least-cost bottom-up mapping between T_1 and T_2.

Example 3. *The mapping of Figure 3 between the trees of Examples 1 and 2 is the bottom-up mapping of largest size between the two trees.*

Under the assumption of unit cost [21, 29], the bottom-up distance between two trees clearly corresponds to the largest bottom-up mapping between the trees. As a matter

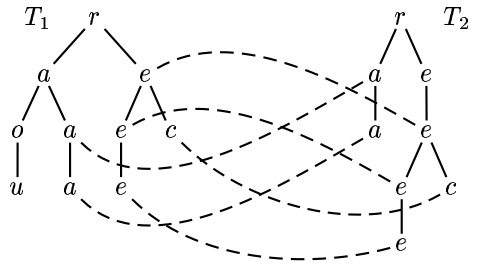


Figure 3. Sample bottom-up mapping.

of fact, it suffices that the cost of identical substitutions be less than the cost of the other edit operations for a largest bottom-up mapping to correspond to the least-cost bottom-up mapping. Similar relationships are also known to hold for the edit distance between graphs [3, 4, 9].

Now, the bottom-up distance turns out to coincide with the top-down distance only for isomorphic trees.

Lemma 2. *Let T_1 and T_2 be two trees and let M be a bottom-up mapping from T_1 to T_2. If M is also a top-down mapping from T_1 to T_2, then $T_1 \cong T_2$.*

Proof. Let T_1 and T_2 be two trees, and let M be a mapping from T_1 to T_2 which is both top-down and bottom-up. Let also $(i, j) \in M$. Since M is a top-down mapping, all nodes in the path from i to the root of T_1 are in M, and since M is also a bottom-up mapping, all nodes of the subtree of T_1 rooted at the root of T_1, that is, all nodes of T_1, are in M. Therefore, $T_1 \cong T_2$. □

There are, however, isolated-subtree mappings which are neither top-down nor bottom-up, as shown by the following example.

Example 4. *The mapping of Figure 4 between the trees of Examples 1 and 2 is an isolated-subtree mapping which is neither a top-down mapping nor a bottom-up mapping.*

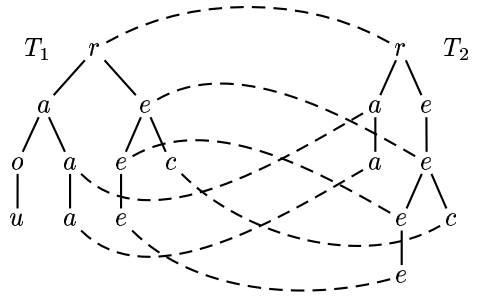


Figure 4. Sample isolated-subtree mapping which is neither top-down nor bottom-up.

4 Computing the Bottom-Up Distance

The bottom-up distance between two trees corresponds to the largest bottom-up mapping between the trees, which in turn corresponds to the largest common forest between the two trees. The latter is because bottom-up mappings are isolated-subtree mappings, and thus correspond to common forests. Therefore, the bottom-up distance between two rooted trees T_1 and T_2 can be computed in linear time by the following steps:

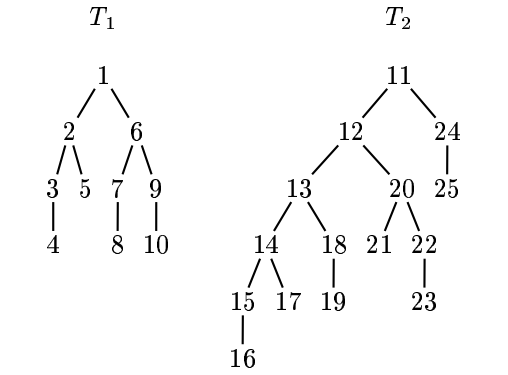
1. Obtain a compacted directed acyclic graph representation G of the forest F consisting of the disjoint union of T_1 and T_2, together with a correspondence K between the nodes of T_1 and T_2 and the nodes of G.
2. Extract a mapping M from T_1 to T_2, according to graph G and node correspondence K.
3. Compute the bottom-up distance between T_1 and T_2, according to mapping M.

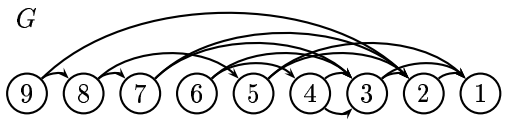
Now, given a forest consisting of the disjoint union of two rooted trees T_1 and T_2, the extended subtree isomorphism problem (finding all subtrees of T_2 that are isomorphic to each subtree of T_1) is equivalent to the extension to forests of the common subexpression problem: represent a rooted tree in a maximally compact form as a directed acyclic graph, where common (isomorphic) subtrees are factored and shared. A solution to any of these problems yield also a largest common forest between T_1 and T_2.

Definition 7. *Let F be a forest. The compacted representation of F is a directed acyclic graph G such that a node $[v]$ of G is an equivalence class of nodes of F, where two nodes u and v are equivalent if, and only if, the subtree of F rooted at u and the subtree of F rooted at v are isomorphic, and there is a directed edge from node $[u]$ to node $[v]$ in G if, and only if, there exist nodes u and v in some rooted tree T of F such that v is a child of u in T.*

Example 5. *Graph G of Figure 5 is the compacted directed acyclic graph representation of T_1 and T_2. The correspondence K between nodes of T_1 and T_2 and nodes of G is given explicitely. Graph G is presented with nodes arranged in bottom-up order of non-decreasing height, from right to left.*

The common subexpression problem was introduced in [8], where a rather complex, linear time algorithm for ordered labeled trees was given. The idea from [10] that a procedure for dynamically maintaining a global table of unique identifiers allows the compacted representation of a rooted tree to be determined in expected linear time, was exploited in [27] to give a simple algorithm for extended





T_1	G	T_1	G	T_2	G	T_2	G	T_2	G
1	6	6	4	11	9	16	1	21	1
2	3	7	2	12	8	17	1	22	2
3	2	8	1	13	7	18	2	23	1
4	1	9	2	14	3	19	1	24	2
5	1	10	1	15	2	20	5	25	1

Figure 5. Compacted directed acyclic graph representation of two rooted trees.

subtree isomorphism that also takes expected linear time, and is here improved to take worst-case linear time by manipulating the rooted trees in the given forest and comparing them with the graph representation of the forest by means of a node correspondence, instead of computing isomorphism codes. More complex algorithms for extended subtree isomorphism are known for ordered, labeled trees [12] and for unordered, unlabeled trees [7].

4.1 Computing the Compacted Directed Acyclic Graph Representation

The algorithm given in Figure 6 computes in time linear in the number of nodes the compacted directed acyclic graph representation G of a forest F consisting of the disjoint union of two rooted trees T_1 and T_2, partitioning thus the set of rooted subtrees of T_1 and T_2 into isomorphism equivalence classes. The resulting correspondence between nodes of the trees and nodes of the graph is returned as parameter K, a map of nodes of T_1 and T_2 to nodes of G.

The algorithm is based on a bottom-up traversal of the trees, that is, a traversal of T_1 and T_2 by order of non-decreasing height. The nodes of T_1 and T_2 are dequeued in bottom-up order, and mapped to a corresponding node in the compacted directed acyclic graph representation G right upon dequeuing them.

Now, because of the bottom-up order, the children of a node of T_1 or T_2 being considered will already have been mapped to nodes in G, which allows to easily check whether the node must be mapped to some node in G corresponding to nodes in T_1 or T_2 of the same height as the node being considered, or it must be mapped to a new node in G. As a matter of fact, notice that a node v in the forest F can be mapped to a node w in the graph G being built, only if the children of node v in F correspond one-to-one to the children of node w in G. Since graph G is built during a bottom-up traversal of forest F, such a correspondence means that the children of node v in F will have been mapped one-to-one to the children of node w in G, and the correspondence can be tested in time linear in the degree of nodes v in F or w in G. Furthermore, since node v in F can be mapped to node w in G only if v and w have the same height, and graph G is being built in order of non-decreasing height, only those nodes w which were last added to G need to be considered as candidates which node v could be mapped to. Alternative algorithms [7, 8, 12], require radix sorting of the set of all nodes of G corresponding to nodes of F of same height as node v, in order to obtain a linear time algorithm.

Notice that, since some ordering among the nodes of a graph is fixed by any particular representation of the graph, and given that in the algorithm in Figure 6 only those nodes w which were last added to graph G need to be considered as candidates which node v could be mapped to, there is no need in line 21 to iterate over all nodes of G and it suffices to iterate over those nodes which where last added to G, that is, to iterate over all nodes of G in reverse (under the particular representation used in the implementation) order.

Regarding time complexity, each of the n_1 nodes in T_1 and n_2 nodes in T_2 is enqueued and dequeued only once and for each dequeued node, only those nodes in G (if any) whose corresponding nodes in T_1 or T_2 have the same height as the dequeued node are considered as candidates which the dequeued node could be mapped to. For each such candidate node, a list of the nodes in G which the children of the node in T_1 or T_2 have been mapped to is compared with a list of the children of the candidate node, whereas these lists are both built according to a same particular ordering of nodes and edges fixed by the representation of G. Therefore, the nodes in G corresponding to all of the nodes in T_1 and T_2 are found or added in $O(n_1 + n_2)$ time.

Dealing with unordered trees just requires sorting the list V of nodes in G which the children of node v in F have been mapped to, and the list W of children of node w in G, right before the $V = W$ equality test. Sorting these lists of nodes can be made in time linear in the length of the lists (the degree of node v in F, same as the outdegree of node

```
    procedure compact (T1, T2: tree, G: graph, K: map)
 1:   let F be the disjoint union of T1 and T2
 2:   let L be an empty map of node labels to nodes of G
 3:   for all leaf labels ℓ in F do
 4:     add a new node z to G
 5:     label[z] ← ℓ
 6:     L[ℓ] ← z
 7:   end for
 8:   let Q be an empty queue of nodes of F
 9:   for all nodes v in F do
10:     children[v] ← outdegree[v]
11:     if children[v] = 0 then
12:       enqueue node v into Q
13:     end if
14:   end for
15:   repeat
16:     dequeue node v from Q
17:     if outdegree[v] = 0 then
18:       K[v] ← L[label[v]]
19:     else
20:       found ← false
21:       for all nodes w in G in reverse order do
22:         if height[v] ≠ height[w] or outdegree[v] ≠
            outdegree[w] or label[v] ≠ label[w] then
23:           break
24:         end if
25:         V ← {K[u] | u is a child of node v}
26:         W ← {K[u] | u is a child of node w}
27:         if V = W then
28:           K[v] ← w
29:           found ← true
30:           break
31:         end if
32:       end for
33:       if not found then
34:         add a new node w to G
35:         K[v] ← w
36:         label[w] ← label[v]
37:         height[w] ← height[v]
38:         for all children u of node v do
39:           add a new arc in G from w to K[u]
40:         end for
41:       end if
42:     end if
43:     if node v is not the root of a tree in F then
44:       children[parent[v]] ← children[parent[v]] − 1
45:       if children[parent[v]] = 0 then
46:         enqueue node parent[v] into Q
47:       end if
48:     end if
49:   until the queue Q is empty
    end procedure
```

Figure 6. Computing the compacted directed acyclic graph representation of a forest, consisting of the disjoint union of two rooted trees.

w in G) using bucket sort, for instance on the node number in the actual representation of G.

4.2 Computing the Bottom-Up Mapping

Given the compacted directed acyclic graph representation G of the disjoint union of two non-empty rooted trees T_1 and T_2, together with a map K of nodes of T_1 and T_2 to nodes of G, a bottom-up mapping from T_1 to T_2 corresponding to a largest common forest between T_1 and T_2 can be obtained by just collecting an unmapped isomorphic subtree in T_2 for each unmapped subtree of T_1 during a preorder traversal of T_1. Although any unmapped isomorphic subtree in T_2 would do in the case of unordered trees, for ordered trees the leftmost one, that is, the one whose root has the least preorder number has to be taken.

The algorithm given in Figure 7 computes a bottom-up mapping from a rooted tree T_1 to a rooted tree T_2, given the compacted directed acyclic graph representation G of the disjoint union of T_1 and T_2, together with the corresponding map K of nodes of T_1 and T_2 to nodes of G. The resulting bottom-up mapping is returned as parameter M, a map of nodes of T_1 to nodes of T_2.

Correctness of the algorithm follows from the fact that every *largest* unmapped subtree of T_1 isomorphic to a subtree of T_2 will be found during a level-order traversal of T_1, upon which the whole subtree of T_1 is mapped to the isomorphic subtree of T_2 found. Such a mapping is made in lines 14–17 during a simultaneous preorder traversal of the two subtrees, although any kind of symultaneous traversal could be used as long as the traversal is of the same kind for both subtrees.

Example 6. *The largest common forest between rooted trees T_1 and T_2 from Example 5, as computed by the bottom-up mapping algorithm, is indicated by encircled nodes in Figure 8.*

4.3 Computing the Bottom-Up Distance

Finally, the bottom-up distance between two rooted trees can be readily computed from a bottom-up mapping between them. The (straightforward) algorithm given in

```
procedure mapping (T1, T2: tree, G: graph, K: map, M: map)
    set M to an empty map of nodes
    let M^-1 be an empty map of nodes
    for all nodes v of T1 in level-order do
      if M[v] is undefined then
        w ← t2[n2]
        for all nodes u of T2 with K[u] = K[v] do
          if M^-1[u] is undefined then
            if preorder[u] < preorder[w] then
              w ← u
            end if
          end if
        end for
        if K[v] = K[w] then
          for all nodes s of T1[v] and t of T2[w] in pre-
          order do
            M[s] ← t
            M^-1[t] ← s
          end for
        end if
      end if
    end for
  end procedure
```

Figure 7. Computing a bottom-up mapping between two rooted trees.

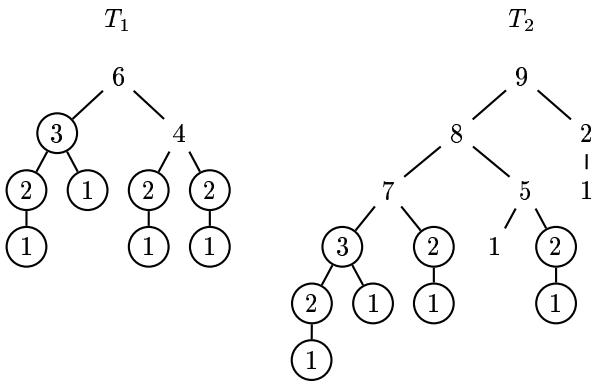


Figure 8. A largest common forest between two rooted trees.

```
1:  function distance (T1, T2: tree, p, q, r: real)
2:    let G be an empty directed graph
3:    let K be a map of nodes of T1 and T2 to nodes of G
4:    let M be a map of nodes of T1 to nodes of T2
5:    compact(T1, T2, G, K)
6:    mapping(T1, T2, G, K, M)
7:    d ← |T1| − |M|
8:    s ← |{(v, w) ∈ M | label[v] ≠ label[w]}|
9:    i ← |T2| − |M|
10:   return s · p + i · q + d · r
11: end function
```

Figure 9. Computing the bottom-up distance between two rooted trees.

Figure 9 computes the bottom-up distance between two non-empty rooted trees T_1 and T_2. A call of the form $distance(T_1, T_2, 1, 1, 1)$ computes the bottom-up distance between T_1 and T_2 under the assumption of unit cost.

5 Conclusions

A new bottom-up distance measure between rooted, labeled trees, which is based on the largest common forest of the trees, is introduced in this paper and compared with previous distance measures published in the literature: edit distance, top-down distance, alignment distance, and isolated-subtree distance. The bottom-up distance, which is a particular case of the isolated-subtree distance, is shown to coincide with the top-down distance only for isomorphic trees, extending thus the known hierarchy among tree edit distances. The diagram in Fig. 10 illustrates the inclusion relationships among the mappings underlying the different distance measures.

The main advantage of the bottom-up distance is the low complexity: it can be computed in time linear in the size of the trees, on rooted, labeled, ordered trees of unbounded degree, and rather simple algorithms are given for computing the measure. Besides, the bottom-up distance has the same low complexity for unordered trees. These facts turn it into a suitable measure for the direct comparison of labeled trees, as well as a fast filter for searching in a metric space of trees.

References

[1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[2] J.-I. Aoe, editor. *Computer Algorithms: String Pattern Matching Strategies*. IEEE Computer Science Press, 1994.

[3] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recogn. Lett.*, 18(8):689–694, 1997.

[4] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recogn. Lett.*, 19(3–4):255–259, 1998.

[5] Y. C. Cheng and S. Y. Lu. Waveform correlation by tree matching. *IEEE Trans. Pattern Anal. Machine Intell.*, 7(3):299–305, 1985.

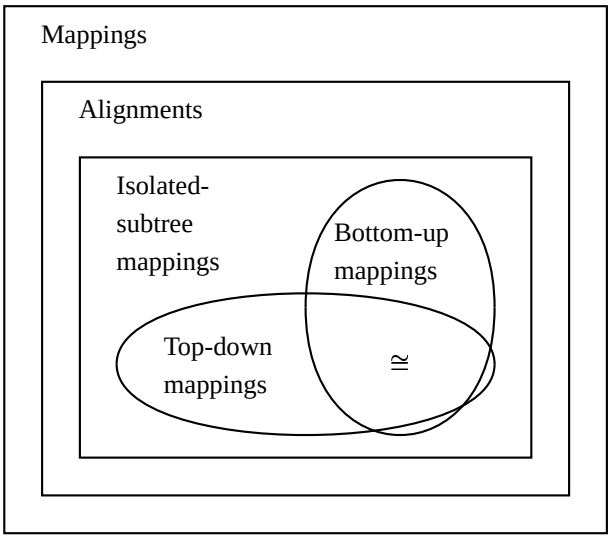


Figure 10. Relationship among tree distance measures.

[6] M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, 1994.

[7] Y. Dinitz, A. Itai, and M. Rodeh. On an algorithm of Zemlyachenko for subtree isomorphism. *Inform. Process. Lett.*, 70(3):141–146, 1999.

[8] P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpression problem. *J. ACM*, 27(4):758–771, 1980.

[9] M.-L. Fernández and G. Valiente. A graph distance measure combining maximum common subgraph and minimum common supergraph. *Pattern Recogn. Lett.*, 22(6–7):753–758, 2001.

[10] P. Flajolet, P. Sipala, and J.-M. Steyaert. Analytic variations on the common subexpression problem. In *Automata, Languages, and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 220–234. Springer-Verlag, 1990.

[11] P. Flajolet and J.-M. Steyaert. A complexity calculus for recursive tree algorithms. *Math. Syst. Theory*, 19(4):301–331, 1987.

[12] R. Grossi. On finding common subtrees. *Theor. Comput. Sci.*, 108(2):345–356, 1993.

[13] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[14] T. Jiang, L. Wang, and K. Zhang. Alignment of trees—an alternative to tree edit. *Theor. Comput. Sci.*, 143(1):137–148, 1995.

[15] P. Kilpeläinen and H. Mannila. Retrieval from hierarchical texts by partial patterns. In *Proc. 16th Annual ACM Conf. Research and Development in Information Retrieval*, pages 214–222, 1993.

[16] B. Moayer and K. S. Fu. A tree system approach for fingerprint pattern recognition. *IEEE Trans. Pattern Anal. Machine Intell.*, 8(3):376–387, 1986.

[17] G. Rozenberg, editor. *Foundations*, volume 1 of *Handbook of Graph Grammars and Computing by Graph Transformation*. World Scientific, 1997.

[18] H. Samet. Distance transform for images represented by quadtrees. *IEEE Trans. Pattern Anal. Machine Intell.*, 4(3):298–303, 1982.

[19] S. M. Selkow. The tree-to-tree editing problem. *Inform. Process. Lett.*, 6(6):184–186, 1977.

[20] B. A. Shapiro and K. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Comput. Appl. Biosci.*, 6(4):309–318, 1990.

[21] D. Shasha and K. Zhang. Fast algorithms for the unit cost editing distance between trees. *J. Algorithms*, 11(4):581–621, 1990.

[22] R. Sleep, R. Plasmeijer, and M. van Eekelen, editors. *Term Graph Rewriting:Theory and Practice*. John Wiley & Sons, 1993.

[23] G. A. Stephen. *String Searching Algorithms*. World Scientific Press, 1994.

[24] R. E. Stobaugh. Chemical substructure searching. *J. Chem. Inf. Comp. Sci.*, 25(3):271–275, 1985.

[25] K.-C. Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.

[26] E. Tanaka and K. Tanaka. The tree-to-tree editing problem. *Int. J. Pattern Recogn. and Artif. Intell.*, 2(2):221–240, 1988.

[27] G. Valiente. Simple and efficient subtree isomorphism. Technical Report LSI-00-72-R, Technical University of Catalonia, Department of Software, 2000.

[28] G. Valiente. Simple and efficient tree comparison. Technical Report LSI-01-1-R, Technical University of Catalonia, Department of Software, 2001.

[29] J. T.-L. Wang and K. Zhang. Finding similar consensus between trees: An algorithm and a distance hierarchy. *Pattern Recogn.*, 34(1):127–137, 2001.

[30] W. Yang. Identifying syntactic differences between two programs. *Software—Practice and Experience*, 21(7):739–755, 1991.

[31] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.

[32] K. Zhang and D. Shasha. Tree pattern matching. In A. Apostolico and Z. Galil, editors, *Pattern Matching Algorithms*, chapter 11. Oxford University Press, 1997.

[33] K. Zhang, R. Statman, and D. Shasha. On the editing distance between unordered labeled trees. *Inform. Process. Lett.*, 42(3):133–139, 1992.