Pramodh Krishna D and Venu Gopal Rao K *EURASIP Journal on Information Security* (2016) 2016:12
DOI 10.1186/s13635-016-0035-2

EURASIP Journal on Information Security

RESEARCH Open Access

Generalized weighted tree similarity algorithms for taxonomy trees

Pramodh Krishna D.[1*] and Venu Gopal Rao K.[2]

Abstract

Taxonomy trees are used in machine learning, information retrieval, bioinformatics, and multi-agent systems for matching as well as matchmaking in e-business, e-marketplaces, and e-learning. A weighted tree similarity algorithm has been developed earlier which combines matching and missing values between two taxonomy trees. It is shown in this paper that this algorithm has some limitations when the same sub-tree appears at different positions in a pair of trees. In this paper, we introduce a generalized formula to combine matching and missing values. Subsequently, two generalized weighted tree similarity algorithms are proposed. The first algorithm calculates matching and missing values between two taxonomy trees separately and combines them globally. The second algorithm calculates matching and missing values at each level of the two trees and combines them at every level recursively which preserves the structural information between the two trees. The proposed algorithms efficiently use the missing value in similarity computation in order to distinguish among taxonomy trees that have the same matching value but with different miss trees at different positions. A set of synthetic weighted binary trees is generated and computational experiments are carried out that demonstrate the effects of arc weights, matching as well as missing values in a pair of trees.

Keywords: Taxonomy trees, Similarity measures, Weighted tree similarity, Generalized weighted tree similarity, Match-miss-measure

1 Introduction

Semantic concept similarity methods and tree similarity techniques have wide range of applications in machine learning [1], e-business [2], e-learning [3], information retrieval [1], case-based reasoning (CBR) [4], image processing and analysis [5], and bioinformatics [5]. In particular, they have many applications in the area of computational linguistics and artificial intelligence such as word sense disambiguation [6], detection and correction of word spelling errors (malapropisms) [7], text segmentation [8], and multi-agent system in e-business and e-learning [2].

Taxonomy is a hierarchical structure that represents relationships among concepts using a tree or graph. Concepts are generalized or specialized depending on the relationships among concepts in the hierarchy. Each concept might have many sub-concepts which are called specialization of the concept and they are represented as children of a concept. A group of concepts might be represented with a single concept which is called generalization of the concepts and it is represented as a parent of the concepts. Concept matching in taxononomy trees is an important task among several semantic concept similarity methods such as schema matching, element matching, and concept matching [9].

A lexical taxonomy is assumed to be structured in a tree-like hierarchy with the concepts represented at every node of the tree. There are many semantic concept similarity methods [10] which can be categorized into two groups: (i) edge counting-based (or dictionary/thesaurus-based) methods and (ii) information theory-based (or corpus-based) methods. Edge counting-based similarity methods [11], e.g., minimum number of edges separating concepts c_1 and c_2, use a metric for measuring the conceptual distance of c_1 and c_2; these methods are useful for specific applications with highly constrained taxonomies. Information theory-based methods [12] extract

*Correspondence: pramodhkrishna.d@gmail.com
[1] CSE, Priyadarshini College of Engineering and Technology, Nellore, India
Full list of author information is available at the end of the article

maximum information content of the concept by finding the negative log likelihood of its probability. Hybrid method such as combination of multiple information sources non-linearly [10] is used to improve semantic similarity. Similarly, tree-based similarity techniques are used in information retrieval. That is, unlike vector space model, interest trees are used to discover groups of users with similar interests in social networking services [13]. Tree-based similarities such as semantic search in digital library which combines Latent Semantic Analyses (LSA) and weighted trees to find similarity between the book's weighted tree and user query weighted tree [14]. Similarily, text retrieval that uses dictionary-based semantic tree to weight the words between query and document is used to find semantic similarities [15].

In literature, tree similarity techniques consider node-labeled trees, whether they are ordered [16, 17] or unordered [18]. These techniques include operations like insertion, deletion and node label substitution [19] with costs defined to transform one tree to another to enable the computation of a distance (which is complementary to, or inverse of, similarity). For local tree matching [20], operations such as merge, cut, and merge-and-cut, with associated operation costs, are also defined to find the best approximate match and matching cost. The Hamming distance [21] is also used in some approaches [22] to compute a tree distance. These techniques require three or four edit operations which ignore the similarity among single nodes and are extremely complicated. Hence, Wang et al. [23] introduced a mapping-based tree similarity algorithm captures node similarity using a dynamic programming scheme and omits inserting and deleting operations.

Tree similarity techniques, with preferences of concepts as weights to the corresponding edges in tree representation named as a weighted tree, have been introduced in a multi-agent system. These techniques are used for matchmaking in e-business environments [2, 24], e-marketplaces [25, 26], P2P e-marketplaces [27, 28], and RNA secondary structure comparison [29]. It allows user preferences to be specified as arc-weights wherein an arc-weight represents importance of a concept. The algorithm combines matching and missing tree values to find the similarity between two taxonomy trees. The missing tree value could be used to rank (or distinguish) the similarities of two or more trees that have the same matching value, but, different missing values. The missing value is calculated using a simplicity function which returns a value between 0 and 1 depending on the structure of a missing tree and associated arc weights. However, this method has some drawbacks such as the way it combines missing and matching tree value in similarity computation. It performs the summation of the simplicity value of a missing tree which is a sibling tree and performs multiplication of the simplicity value of the missing tree which is a child tree in similarity computation. Fuzzy concepts are introduced and a non-symmetric fuzzy similarity is developed in [30] which assumes identical tree structures and uses only the matching value.

In this paper, we propose two generalized weighted tree similarity algorithms. We introduce a generalized formula to combine the matching and missing values in order to find the similarity between two taxonomy trees. It also determines the importance of the matching and missing value. The first generalized algorithm calculates matching and missing information between two taxonomy trees separately and combines them globally. The second generalized algorithm calculates matching and missing values at each level of the two trees and combines them at every level recursively, preserving the structural information present in the trees while calculating similarity. The proposed algorithms efficiently use the missing value in order to distinguish among taxonomy trees that have the same matching value, but, have different missing values. A set of synthetic weighted binary trees is generated and computational experiments are carried out. Our results illustrate the effects of arc weights and matching as well as missing values between two trees.

The paper is organized as follows. Section 2 briefly explains notations and definitions, followed by the weighted tree similarity algorithm. Section 3 describes drawbacks of the existing tree similarity algorithm with an illustrative example followed by a proposed generalized formula to combine matching and missing values. Section 4 proposes the generalized weighted tree similarity algorithms. Section 5 describes the synthetic trees and presents experimental results. Finally, Section 6 concludes the paper.

2 Weighted tree similarity algorithm (wT-similarity)

In this section, we briefly introduce notations and definitions followed by the existing weighted tree similarity algorithm [2]. The original algorithm is presented using Relfun, whereas we give a more generic specification.

2.1 Notation and definitions

Let T and $T^{'}$ be the roots of two given trees. Let the root node of tree T have m arcs which are represented as $l_1, l_2, \ldots, l_m$ and the sub-tree associated with arc l_i is represented with the subroot node T_i, for $i = 1$ to m. Let the root node of tree $T^{'}$ have n arcs which are represented as $l^{'}_1, l^{'}_2, \ldots, l^{'}_n$ and the sub-tree associated with arc $l^{'}_j$ is represented with the subroot node $T^{'}_j$, for $j = 1$ to n. Let the weight of arc l_i be represented as $w(l_i)$

and the weight of arc $l_j^{'}$ be represented as $w(l_j^{'})$. Let the node label be represented as *Node-label(T)*. The similarity between two trees can be calculated by comparing node labels of the two trees recursively (see Section 2.2 for more details).

Definition 1. ***Normalized arc-labeled, arc-weighted tree [31].*** *A normalized arc-labeled, arc-weighted tree is a 5-tuple* $T = (V, E, L_V, L_E, L_W)$ *consisting of a set of nodes (vertices)V, a set of arcs (edges) E, a set of node labels* L_V*, a set of arc labels* L_E *and a set of arc weights* $L_W = [0, 1]$ *having a fan-out-weight-normalized* $(n \rightarrow 1, n \geq 1)$ *mapping from the elements in E to the elements in* L_W*, i.e. the weights of every fan-out add up to 1.0.*

2.2 Weighted tree similarity algorithm

This subsection describes the existing weighted tree similarity algorithm proposed by Bhavsar et al. [2]. The algorithm carries out a recursive top-down traversal comparison of two given normalized arc-labeled, arc-weighted trees (see Definition 1) which computes a similarity value between $[0, 1]$. If the two trees and their respective concepts (i.e., entire structures) are the same, then the similarity is equal to 1, otherwise if the root concepts of two trees are different the similarity equal to 0. For all other cases the similarity value is between 0 and 1.

Initially, the weighted tree similarity algorithm compares the labels of the root node of two given trees (see Algorithm 1). If the labels are different, then it returns the similarity as zero. Otherwise, if the labels are same, then it works as follows. If the two nodes are leaf nodes, then the weighted tree similarity algorithm returns 1 (shown as case 1 in Fig. 1). If the root node of one of the trees is also a leaf and of the other tree is a non-leaf, then the algorithm computes similarity as the sum of weighted simplicity values of the sub-tree associated with each arc of the non-leaf node (see cases 3 and 4 in Fig. 1). Suppose l_i arc is associated with the sub-tree $\mathcal{T}_i$, then the weighted simplicity value is calculated as $w(l_i)/2$ multiplied with the simplicity value of the sub-tree $\mathcal{T}_i$. The simplicity algorithm (given in Section 2.3) computes contribution of the missing tree in the similarity computation. Now, consider case 2 in Fig. 1. If the root nodes of two trees are non-leaf nodes, then the algorithm compares the arc-labels of the root nodes of these two trees and computes similarity as follows:

1. For each pair of identical arcs, the average weight of these arcs is computed and then it is multiplied with the weighted tree similarity of their sub-trees recursively. Finally, the similarities computed for each pair of identical arcs are added.
2. If an arc is present in one tree and it is not present in the second tree, then the similarity is computed as the weighted simplicity values of the missing tree (because the missing tree value should contribute to the similarity computation).

Algorithm 1 wT-similarity($\mathcal{T}$, $\mathcal{T}^{'}$)

```
match = 0, λvalue= 0, 0 < ϵ <= 1;
if Node-label(T) = Node-label(T') then
  {Case.1 in Fig. 1}
  if T & T' are leaf nodes then
    Return 1;
  else
    {Case.2 in Fig. 1}
    if T & T' are non-leaf nodes then
      for each pair of identical arc labels l_i ∈ T and l'_j ∈ T' do
        match = match + (w(l_i) + w(l'_j))/2 * wT-similarity(T_i, T'_j);
      end for
      for each arc l_i which is present in T and it does not present in T' do
        λvalue= λvalue + w(l_i)/2 * λ(1.0, T_i);
      end for
      for each arc l'_j which is present in T' and it does not present in T do
        λvalue = λvalue + w(l'_j/2) * λ(1.0, T'_j);
      end for
      Return (ϵ + (1 − ϵ) * (match + λvalue));
    else
      {Case.3 in Fig. 1}
      if T is a non-leaf & T' is a leaf then
        for each arc l_i ∈ T do
          λvalue = λvalue + w(l'_j)/2 * λ(1.0, T_i);
        end for
        Return (ϵ + (1 − ϵ) * (λvalue));
      else
        {Case.4 in Fig. 1}
        if T is a leaf & T' is a non-leaf then
          for each arc l'_j ∈ T' do
            λvalue = λvalue + w(l_i)/2 * λ(1.0, T'_j);
          end for
          Return (ϵ + (1 − ϵ) * (λvalue));
        end if
      end if
    end if
  end if
else
  Return 0;
end if
```

Note that the weighted tree similarity algorithm computes two parts of the similarity. If the two node labels are identical, then the first part defines a small similarity value contribution due to identical node labels: $0 < \epsilon < 0.5$.

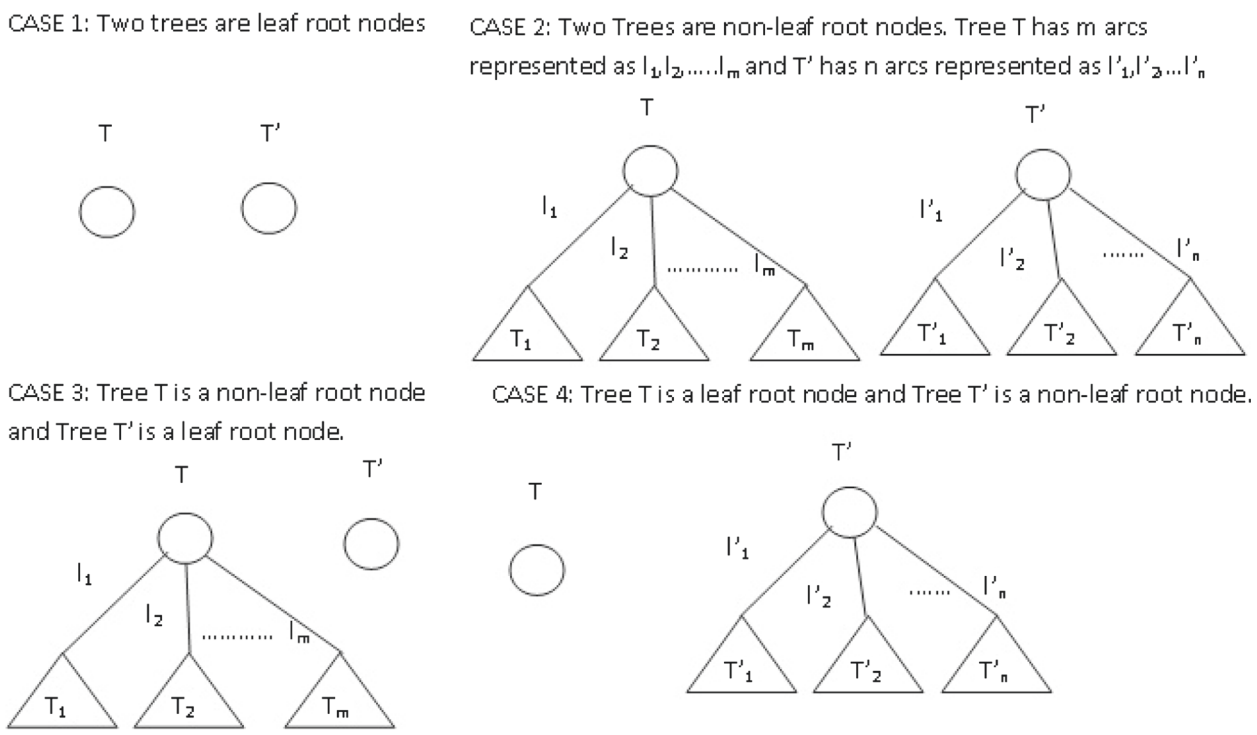

Fig. 1 Weighted tree similarity technique

The similarity of the sub-trees of these two nodes (computed recursively) is multiplied with $1-\epsilon$ and this forms the second part and it is added to the first part. This idea avoids giving zero similarity in the recursive procedure if the root node labels of two trees are same, but all children labels are different; in this case, it guarantees a small ϵ value due to identical node labels.

The similarity in the above approach is computed by recursive top-down (root-leaf) traversal of all corresponding nodes of the two trees as given in Algorithm 1 and followed by the simplicity algorithm explained in Section 2.3.

2.3 Simplicity algorithm for missing trees

In this section, simplicity algorithm [31] is explained. If any missing tree is present in one of the two trees for which the similarity is evaluated, then the missing tree contribution is given by simplicity algorithm. It is introduced by Lu Yang et al. in [31] where the simplicity algorithm depends on the structure of the missing tree.

The simplicity algorithm has two parameters: one simplicity factor (r) which initializes the missing tree contribution in similarity computation and the other parameter is the missing tree $\mathcal{T}$. This function returns a value between $[0, r]$. If the tree has only one missing node, then its simplicity value is r. If the tree grows vertically, then the depth degradation factor d is used to decrease simplicity value of a tree which is taken as 0.5. If the tree grows horizontally, then the weighted average of all its child sub-trees simplicity function value is used. If a tree horizontally and vertically grows infinitely, then its simplicity value approaches to zero. The algorithm for simplicity function is given in Algorithm 2. The recursive simplicity function is given below:

$$\lambda(r,\mathcal{T}) = \begin{cases} r & \text{if } \mathcal{T} \text{ is a leaf,} \\ \frac{1}{m}\sum_{j=1}^{m} w(l_j) * \lambda(r.d, \mathcal{T}_j) & \text{otherwise} \end{cases} \tag{1}$$

$\lambda(r,\mathcal{T})$ is a simplicity function

r- is the simplicity factor
d-is depth degradation factor (which is equal to 0.5)
m- m arcs for a tree $\mathcal{T}$ specified as l_j, for $j = 1$ to m.
T_j- is the subtree attached to arc l_j
$w(l_i)$- is the weight of arc l_i, for $j = 1$ to m.

3 Limitations of the weighted tree similarity algorithm

This section first describes an example to explain the limitations of the existing weighted tree similarity algorithm. This section also introduces a generalized formula to combine the matching and missing values of two taxonomy trees. The generalized formula combines both of these values as harmonic mean between them which follows *F-measure* in statistical theory.

Algorithm 2 $\lambda(r, \mathcal{T})$

$d = 0.5$, sum = 0;
if $\mathcal{T}$ is a leaf node **then**
  Return (r);
else
  { let the tree $\mathcal{T}$ has m arcs and each arc has $weight(l_i)$ and subtree $\mathcal{T}_i$, for $i = 1$ to m.}
  for each arc l_i of root node $\mathcal{T}$ **do**
    sum = sum + weight(l_i) * $\lambda(r * d, \mathcal{T}_i)$;
  end for
  sum = $\frac{1}{m} \times$ sum;
  Return sum;
end if

3.1 Limitations

Figure 2 represents three trees in which tree *T1* and tree *T2* differ a missing sub-tree at node *NA* whereas tree *T1* and tree *T3* differ a missing sub-tree at node *Head*. The missing sub-tree is same in both cases but present as a child tree at tree *T2* and present as a sibling tree at tree *T3* when compared to tree *T1*. The existing weighted tree similarity algorithm calculates the similarity between tree *T1* and tree *T2* as $\frac{(1.0+1.0)}{2} \times \frac{1.0}{2} \times 0.5 = 0.25$ and the similarity between tree *T1* and tree *T3* as $\frac{(1.0+0.5)}{2} + \frac{0.5}{2} \times 0.5 = 0.875$. The matching value is 1.0 between tree *T1* and tree *T2* whereas 0.75 between tree *T1* and tree *T3*. However, the simplicity of the missing sub-tree is 0.5 in both the cases which decreases similarity value in one case and increases similarity value in other case when compared with matching value. These two cases likely to have same similarities since same matching and missing values.

3.2 A generalized measure

We introduce a generalized formula to combine the matching and missing values between two taxonomy trees which is given in Eq. 2. Let the matching value be *match* and missing value be λ*value*. The generalized formula uses the missing value always less than or equal to matching value such that it gives a resultant similarity which is less than or equal to the matching value between two taxonomy trees . It means that the missing value reduces the similarity when compared to the matching value after combining them using the generalized formula. If there is no missing tree, then the missing value is equal to matching value such that for any β value the combined formula is equal to matching value only (see Eq. 2). Let the combined similarity value be τ

$$\tau = \frac{(\beta^2 + 1) * \text{match} * \lambda\text{value}}{\beta^2 * \text{match} + \lambda\text{value}} \tag{2}$$

If there is any missing tree, then the missing value is less than the matching value such that the combined formula reduces its similarity when compared to matching value. The parameter β gives the importance between matching and missing values. If $\beta < 1$, then matching value has more importance; and if $\beta > 1$, then the missing value has more importance. If $\beta = 1$, then the combined formula gives a harmonic mean between missing and matching values. The combined formula can be used to combine the matching and missing values globally once or at each level of the two taxonomy trees which is proposed in Section 4.

4 Generalized weighted tree similarity algorithm

This section proposes two generalized weighted tree similarity algorithms such as (i) a weighted tree similarity algorithm that uses generalized formula once globally called as generalized global weighted tree similarity algorithm (GG-W-tree similarity algorithm) and (ii) a weighted tree similarity algorithm that uses generalized formula at each level of two trees to preserve the structural information between them called as generalized level-wise weighted tree similarity algorithm (GLW-W-tree similarity algorithm).

4.1 GG-W-tree similarity algorithm

This subsection explains a generalized algorithm that combines matching and missing values globally. It consists of *Match* and *Miss* algorithms. The *Match* algorithm computes the matching value between two taxonomy trees

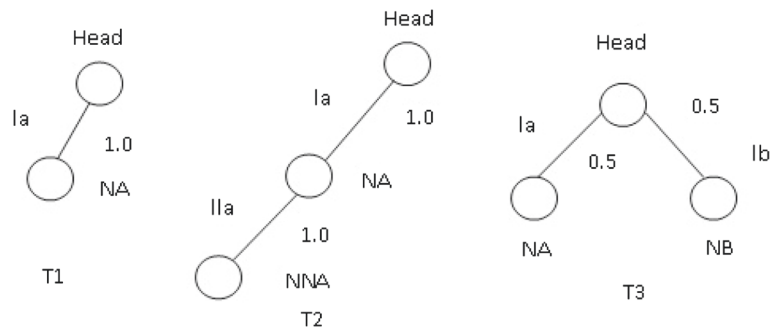

Fig. 2 Three trees to illustrate limitations of the weighted tree similarity algorithm

by traversing from top to bottom and computes from bottom to top recursively. However, *Miss* algorithm computes missing value as average simplicity values of missing trees between two taxonomy trees. Finally, the matching and missing values are combined globally using the generalized formula.

Algorithm 3 Match($\mathcal{T}$, $\mathcal{T}'$)

```
match = 0, 0 < ϵ <= 0.5;
if Node-label(T) = Node-label(T') then
  {Case 1, Case 3 and Case 4 in Fig. 1}
  if (T & T' are leaf nodes) or (T is a non-leaf & T' is
  a leaf) or (T is a leaf & T' is a non-leaf) then
    Return 1;
  else
    {Case 2 in Fig. 1}
    if T & T' are non-leaf nodes then
      for each pair of identical arc labels l_i ∈ T and
      l'_j ∈ T' do
        match = match + (w(l_i) + w(l'_j))/2 * Match(T_i,
        T'_j);
      end for
      Return (ϵ + (1 − ϵ) * (match));
    end if
  end if
else
  Return 0;
end if
```

Match algorithm returns 1.0 for case 1, case 3, and case 4 in Fig. 1 if the node labels are the same. For case 2, it computes the average arc weights of two identical arcs each taken from one tree and multiplies with the *Match* algorithm at their sub-trees recursively. It sums the matching value at every pair of such identical arcs. Note that it gives a small similarity as node equality (ϵ) to avoid zero matching value because of multiplication of recursive algorithm in case if all sub-trees matching value is zero. *Match* algorithm is given in Algorithm 3.

Miss algorithm computes the missing value using simplicity algorithm explained in Section 2.3. Initially, *Miss* algorithm has a parameter called *match* which is matching value between two taxonomy trees. If there is no missing value then it returns *match* value, otherwise the missing value is always less than *match* value. *Miss* algorithm has a counter (*count*) to represent the number of missing trees. Let the missing value be $\lambda value$.

If the two given trees have non-leaf root nodes, then the *Miss* algorithm finds identical arcs where each arc from one tree and recursively calls *Miss* algorithm at their respective sub-trees to calculate any missing value present at their descendants. If any arc which is present in one tree and is absent in other tree, then the missing value is calculated as arc weight multiplied with the simplicity value at its respective sub-tree. Such a pair of trees is shown as case 2 in Fig. 1. If one tree is leaf and other tree is non-leaf, then *Miss* algorithm computes simplicity value at non-leaf node, which are shown as case 3 and case 4 in Fig. 1. *Miss* algorithm sums all such missing value and the counter is updated. Finally, the algorithm returns the average missing value (see Algorithm 4).

GG-wT-similarity algorithm computes matching and missing values separately by using Algorithm 3 and Algorithm 4. Finally, it combines the matching and missing values using the generalized formula given in Eq. 2 and the GG-wT-similarity algorithm is given in Algorithm 5.

Algorithm 4 Miss($\mathcal{T}$, $\mathcal{T}'$, match)

```
λvalue = 0, count = 0, 0 < ϵ <= 0.5;
if Node-label(T) = Node-label(T') then
  {Case 2 in Fig. 1}
  if T & T' are non-leaf nodes then
    for each pair of identical arc labels l_i ∈ T and l'_j ∈
    T' do
      Miss(T_i, T'_j);
    end for
    for each arc l_i which is present in T and it does not
    present in T' do
      λvalue = λvalue + w(l_i) * λ(match, T_i);
      count = count + 1;
    end for
    for each arc l'_j which is present in T' and it does
    not present in T do
      λvalue = λvalue + w(l'_j) * λ(match, T'_j);
      count = count + 1;
    end for
  else
    {Case 3 in Fig. 1}
    if T is a non-leaf & T' is a leaf then
      λvalue = λvalue + λ(match, T);
      count = count + 1;
    else
      {Case 4 in Fig. 1}
      if T is a leaf & T' is a non-leaf then
        λvalue = λvalue + λ(match, T'|);
        count = count + 1;
      end if
    end if
  end if
  return (λvalue/count);
else
  return match;
end if
```

Algorithm 5 GG-wT-similarity($\mathcal{T}$, $\mathcal{T}'$, β)

match = Match($\mathcal{T}$, $\mathcal{T}'$);
$\lambda value$ = Miss($\mathcal{T}$, $\mathcal{T}'$);
if $match \neq 0$ & $\lambda value \neq 0$ **then**
  return $(\frac{(\beta^2+1)*match*\lambda value}{\beta^2*match+\lambda value})$;
else
  return match;
end if

4.2 GLW-wT-similarity algorithm

This subsection describes a generalized level-wise weighted tree similarity algorithm (GLW-wT-similarity algorithm) which combines matching and missing values at each level of two trees to preserve the structural information.

This algorithm computes matching and missing information at each level of the two taxonomy trees by traversing from top to bottom and computes from bottom to top recursively. It computes matching and average missing information at each level and combines them using combined formula (see Eq. 2) and returns it to its previous level matching information. Similarly, it recursively computes from bottom to top and returns similarity between two taxonomy trees at the root level.

Initially, the algorithm finds identical arcs from their root nodes where the first arc taken form the first tree and the second arc taken from the second tree and it goes to bottom level recursively where there is no such identical arcs. This becomes one of the three cases such as case 1, case 3, and case 4 in Fig. 1. For case 3 and case 4, the matching information is taken as 1.0 since two node labels are the same and missing information is computed at non-leaf tree node using the simplicity algorithm and returns the combined value to its previous level matching information. Case 1 returns 1.0 to its previous level matching information. Case 2 in Fig. 1 recursively combines the sum of the matching values of the identical arcs and average missing value at each level and returns it to its previous level matching information. Similarly, the GLW-W-tree similarity algorithm computes recursively from bottom till it reaches the root nodes of the two taxonomy trees and returns the similarity between them. The algorithm for GLW-W-tree-similarity is described in Algorithm 6. The following section describes the experimental results for the proposed algorithms for various parameters and various weights.

Algorithm 6 GLW-wT-similarity($\mathcal{T}$, $\mathcal{T}'$, β)

match = 0, $\lambda value$ = 0, $0 < \epsilon <= 0.5$;
if Node-label($\mathcal{T}$) = Node-label($\mathcal{T}'$) **then**
  {Case 1 in Fig. 1}
  if $\mathcal{T}$ & $\mathcal{T}'$ are leaf nodes **then**
    Return 1;
  else
    {Case 2 in Fig. 1}
    if $\mathcal{T}$ & $\mathcal{T}'$ are non-leaf nodes **then**
      for each pair of identical arc labels $l_i \in \mathcal{T}$ and $l_j' \in \mathcal{T}'$ **do**
        match = match + $(w(l_i) + w(l_j'))/2$ * GLW-wT-similarity($\mathcal{T}$, $\mathcal{T}'$, β);
      end for
      match = ϵ + $(1 - \epsilon)$ * match;
      for each arc l_i which is present in $\mathcal{T}$ and it does not present in $\mathcal{T}'$ **do**
        miss = miss + Simplicity(match, $\mathcal{T}$);
        count = count + 1;
      end for
      for each arc l_j' which is present in $\mathcal{T}'$ and it does not present in $\mathcal{T}$ **do**
        miss = miss + Simplicity(match, $\mathcal{T}'$);
        count = count + 1;
      end for
    else
      match = 1.0, count = 1;
      {Case 3 in Fig. 1}
      if $\mathcal{T}$ is a non-leaf & $\mathcal{T}'$ is a leaf **then**
        miss = miss + Simplicity(match, $\mathcal{T}$)
      else
        {Case 4 in Fig. 1}
        if $\mathcal{T}$ is a leaf & $\mathcal{T}'$ is a non-leaf **then**
          miss = miss + Simplicity(match, $\mathcal{T}'$);
        end if
      end if
    end if
  end if
  if $match \neq 0$ & $miss \neq 0$ **then**
    miss = miss/count;
    return $(\frac{(\beta^2+1)*match*miss}{\beta^2*match+miss})$;
  else
    return match;
  end if
else
  Return 0;
end if

5 Experimental results

This section describes experimental results. A set of all possible binary trees upto two levels are generated which starts with a single node tree *T1* and ends with a complete two level binary tree *T25*. These generated trees are shown in Fig. 3, and tree *T7* is taken as a reference tree to calculate the similarity with all trees in the experiments. It is shown in a box with dotted lines in Fig. 3. In Section 5.1,

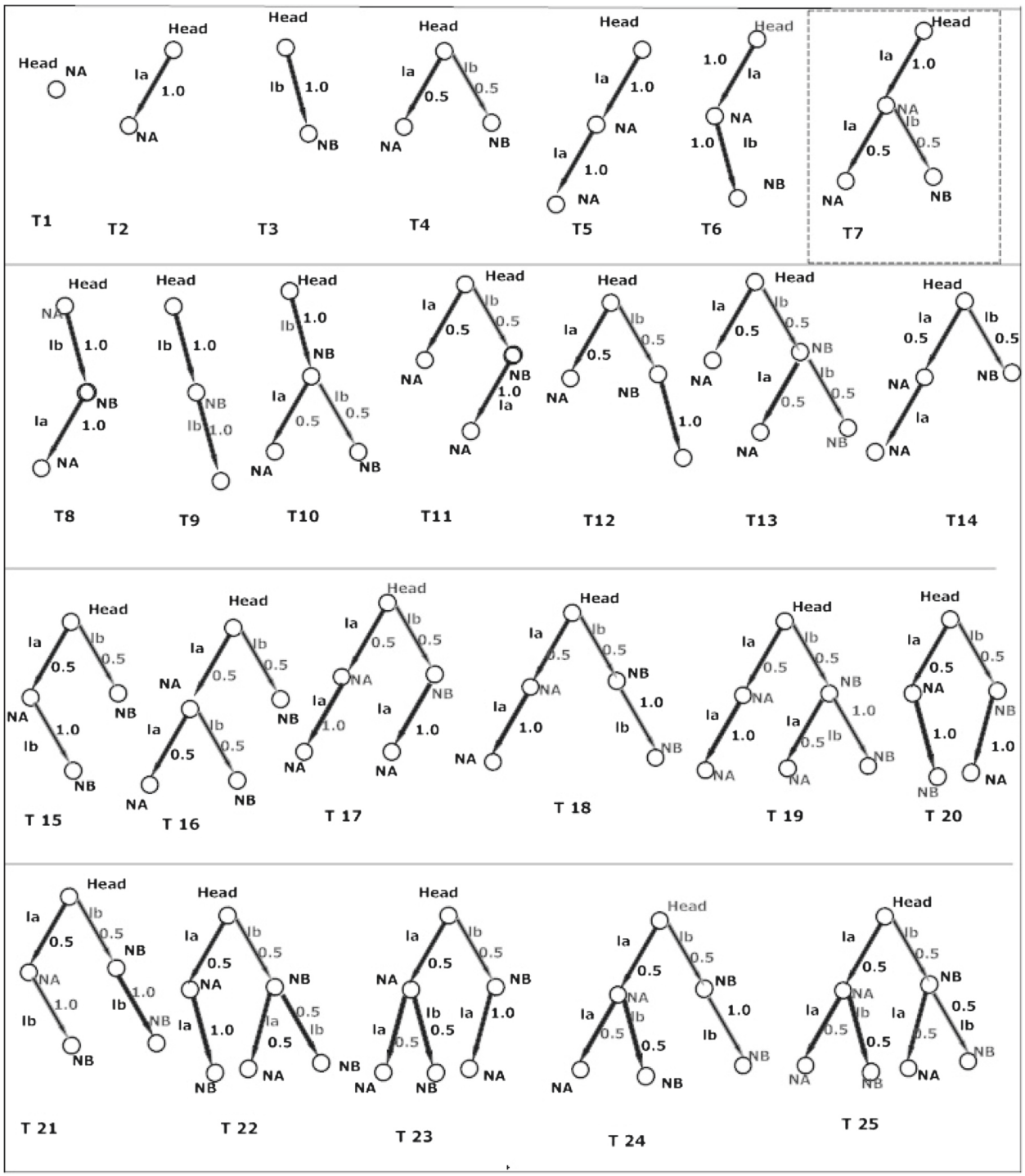

Fig. 3 Trees generated for experiments and the reference tree is represented in a *box* with *dotted lines*

arc weights of the trees are fixed as shown in Fig. 3, experiments are done, and similarity values are plotted for various β and depth degradation values. In Section 5.2, β value and depth degradation values are fixed, experiments are done, and similarity values are plotted for various arc weights of the trees.

5.1 Generalized tree similarity for various parameter values of the combined formula

This subsection provides experimental results for various β values that are used to combine matching and missing information between two trees and two depth degradation values. Two plots are drawn for two depth

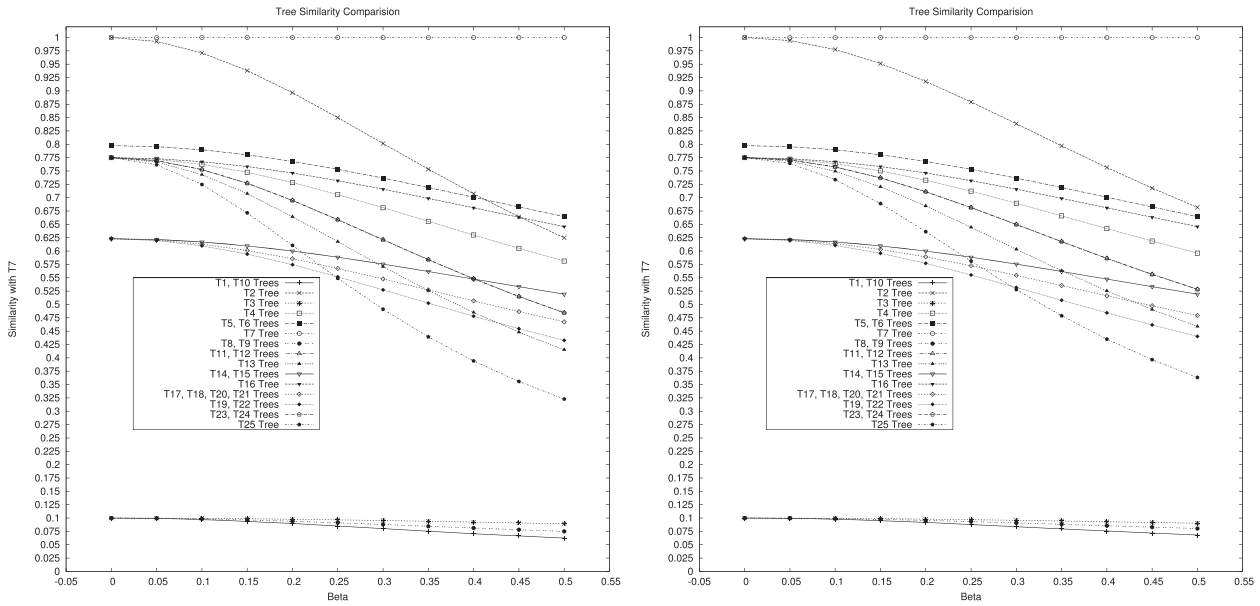

Fig. 4 GG-W-tree similarity for various β values

degradation values of 0.5 and 0.8. Each of the two plots presents the similarity values of all trees represented on y-axis for different β values taken from $\{0.0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5\}$ which are represented on x-axis. There are two such results shown in Figs. 4 and 5 of the proposed generalized tree similarity tree algorithms, that is, GG-W-tree similarity algorithm and GLW-W-tree similarity algorithm, respectively. Some of the trees have the same matching and missing information when compared with tree *T7* which are structurally the same and are grouped in a single representation.

It is clear from the results that the missing information decreases the similarity values when compared to matching information using the proposed combined formula which are controlled by β and depth degradation values. In Fig. 4, although trees *T11, T12* and trees *T23, T24* are structurally different, they have same similarity values when matching and missing information is combined globally using GG-W-tree similarity algorithm whereas

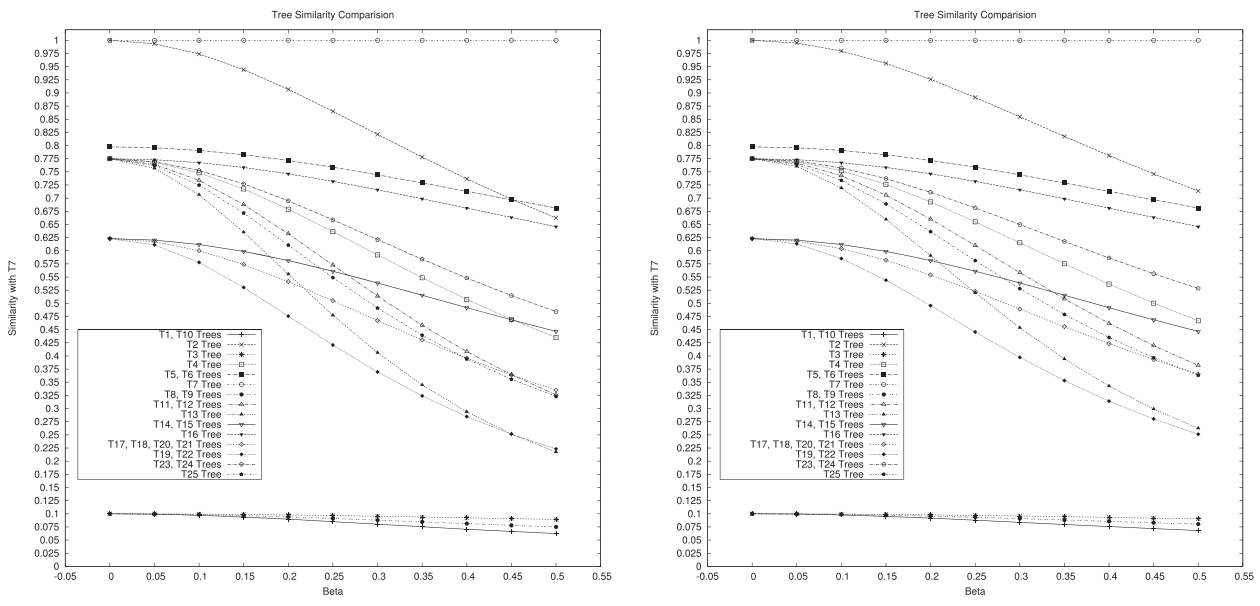

Fig. 5 GLW-W-tree similarity for various β values

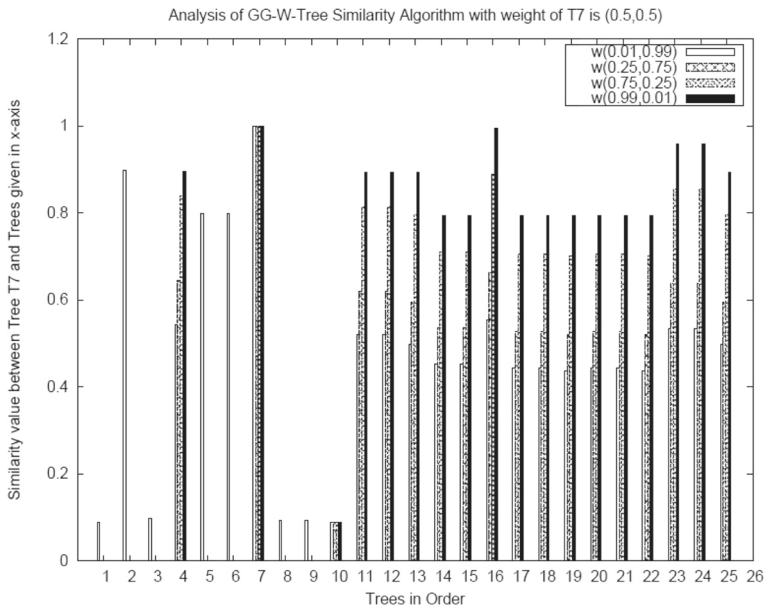

Fig. 6 Similarity values between tree *T7* and the set of trees in Fig. 3 using the GG-W-tree algorithm for various weights

they have different similarity values when they use GLW-W-tree similarity algorithms as shown in Fig. 5 since it preserves structural information. From the results, we have chosen β value as 0.2 and depth degradation value as 0.5.

5.2 Generalized tree similarity for various arc weights

The experimental results of the generalized tree similarity algorithms for various arc weights of the trees are given in Fig. 7. The parameter β is fixed as 0.2 and depth degradation factor is set as 0.5. The arc weights are assigned as $(\epsilon, 1-\epsilon)$ if the node of a tree has two arcs, else the arc weight is 1.0. The set of ϵ values is chosen as $\{0.01, 0.25, 0.75, 0.99\}$. The reference tree is *T7* as given in Fig. 3. The similarity values between tree *T7* and all trees using GG-W-tree similarity algorithm is given in Fig. 6. Similarly, the similarity between tree *T7* and all trees using GLW-W-tree similarity is given in

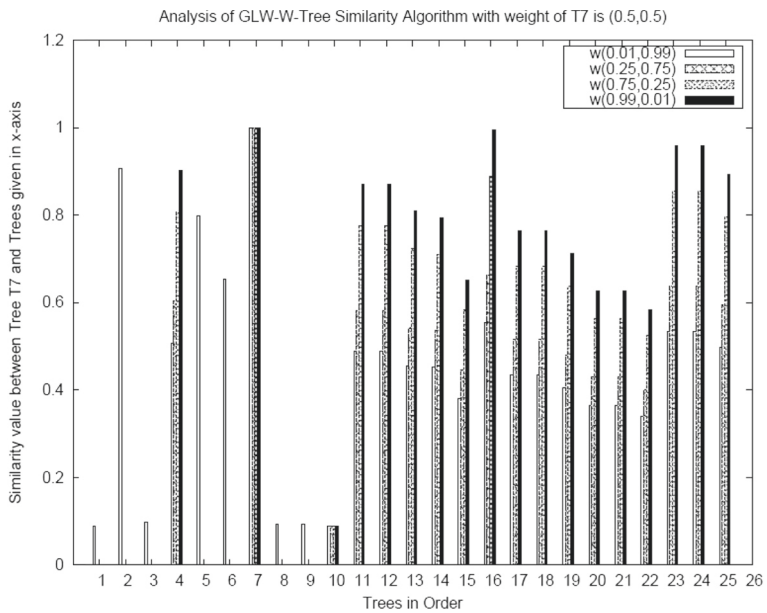

Fig. 7 Similarity values between tree *T7* and the set of trees in Fig. 3 using the GLW-W-tree algorithm for various weights

Fig. 7. It is clear from the results that as the weight of left arc increases from 0.01 to 0.99, then the similarity value increases.

6 Conclusions

In this paper, limitations of the weighted tree similarity algorithm have been identified and a generalized formula to combine matching and missing information between two taxonomy trees are proposed. We presented two generalized weighted tree algorithms. The first algorithm is called GG-W-tree similarity algorithm that combines matching and missing information globally, and the second algorithm is called GLW-W-tree similarity algorithm that combines matching and missing information at each of level of the two trees inorder to preserve structural information. Experiments are conducted and similarity values of the proposed two algorithms are plotted for various parameters of the combined formula and various weights of the trees. Thus, the proposed methods are generalized compared to the weighted tree similarity method for matchmaking environments.

Competing interests

I declare that I have no significant competing financial, professional, or personal interests that might have influenced the performance or presentation of the work described in this manuscript.

Authors' contributions

DPK proposed the new algorithms for similarity search in taxonomy trees. He is involved in the implementation of the code and development of the synthetic data set and developed the plots for comparison of results. He mainly contributed in writing several sections of the paper such as proposed algorithms and experimental results section. VGRK is involved in writing the introduction part. He is involved in the technical discussions and contributed in the proposed algorithms partially. He supported in writing the literature review and revising the manuscript. Both authors read and approved the final manuscript.

Author details

[1]CSE, Priyadarshini College of Engineering and Technology, Nellore, India. [2]CSE, GNITS, Hyderabad, India.

Received: 24 January 2016 Accepted: 4 May 2016
Published online: 03 June 2016

References

1. H Nottelmann, U Straccia, Information retrieval and machine learning for probabilistic schema matching. Inf. Process. Manag. **43**, 552–576 (2007)
2. VC Bhavsar, H Boley, L Yang, A weighted-tree similarity algorithm for multi-agent systems in e-business environments. Comput. Intell. **20**(4), 584–602 (2004)
3. H Boley, VC Bhavsar, D Hirtle, A Singh, Z Sun, L Yang, A match-making system for learners and learning objects. Int. J Interactive Technol. Smart Educ. **2**(3), 171–178 (2005)
4. JA Iyer, P Bhattacharyya, Using semantic information to improve case retrieval in case-based reasoning systems in International Conference on the Convergence of Knowledge, Culture, Language and Information Technologies, December 2–6, 2003, Alexandria, Egypt, 1–6
5. VN Kamat, Inductive learning with the evolving tree transformation system, Doctoral thesis, Faculty of Computer Science. Fredericton, Canada, University of New Brunswick (1996)
6. P Resnik, Semantic similarity in a taxonomy: an information based measure and its application to problems of ambiguity in natural language. J Artif. Intell. Res. **11**, 95–130 (1999)
7. A Budanitsky, G Hirst, in *Workshop on WordNet and Other Lexical Resources*. Semantic distance in WordNet: an experimental, application-oriented evaluation of five measures, vol. 2, (2001)
8. H Kozima, Computing lexical cohesion as a tool for text analysis, doctoral thesis, computer science and information maths, University of Electo-Comm (1994)
9. L Yang, A survey on semantic concept similarity algorithms and an approach of concept relative-depth scaling, CS6999 Directed Studies Course (Semantic Matching Algorithm) Report (2005)
10. Y Li, ZA Bandar, D McLean, An approach for measuring semantic similarity between words using multiple information sources. IEEE Trans Knowl. Data Eng. **15**(4), 871–882 (2003)
11. R Rada, H Mili, E Bichnell, M Blettner, Development and application of a metric on semantic nets. IEEE Trans. Knowl. Data Eng. **9**(1), 17–30 (1989)
12. P Resnik, Using information content to evaluate semantic similarity in a taxonomy. arXiv preprint cmp-lg/9511007, 448–453 (1995)
13. L Zhang, et al., in *Computer and Information Technology (CIT), 2012 IEEE 12th International Conference on*. Discovering similar user models based on interest tree (IEEE, 2012), pp. 1046–1050
14. U Saadah, R Sarno, UL Yuhana, in *The Proceedings of The 7th ICTS, Bali*. Latent semantic analysis and weighted tree similarity for semantic search in digital library, (2013), pp. 159–164
15. G Huang, X Zhang, in *E-Product E-Service and E-Entertainment (ICEEE), 2010 International Conference on*. Text retrieval based on semantic relationship (IEEE, 2010), pp. 1–4
16. J Wang, BA Shapiro, D Shasha, K Zhang, KM Curry, An algorithm for finding the largest approximately common substructures of two trees. IEEE Trans. Pattern Anal. Mach. Intell. **20**, 889–895 (1998)
17. D Shasha, J Wang, K Zhang, Treediff: approximate tree matcher for ordered trees (2001)
18. D Shasha, J Wang, K Zhang, Exact and approximate algorithm for unordered tree matching. IEEE Trans. Syst. Man Cybernet. **24**(4), 668–678 (1994)
19. S Lu, A tree-to-tree distance and its application to cluster analysis. IEEE Trans. Pattern Analy. Mach. Intell. **1**(2), 219–224 (1979)
20. T Liu, D Geiger, in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*. Approximate tree matching and shape similarity, vol. 1 (IEEE, 1999), pp. 456–462
21. RW Hamming, in *Coding and information theory*. Error-correcting codes, 2nd Edition (Prentice-Hall, Inc., 1986)
22. B Schindler, F Rothlauf, HJ Pesch, in *Applications of Evolutionary Computing*. Evolution strategies, network random keys, and the one-max tree problem (Springer Berlin Heidelberg, 2002), pp. 143–152
23. J Wang, H Liu, H Wang, A mapping-based tree similarity algorithm and its application to ontology alignment. Knowl. Based Syst. **56**, 97–107 (2014)
24. L Yang, M Ball, H Boley, VC Bhavsar, Weighted partonomy-taxonomy trees with local similarity measures for semantic buyer-seller match-making. J. Business Technol. **1**(1), 42–52 (2005)
25. L Yang, BK Sarker, VC Bhavsar, H Boley, Range similarity and satisfaction measures for buyers and sellers in e-marketplaces. J. Intell. Syst. **17**(1), 247–266 (2008)
26. M Joshi, VC Bhavsar, H Boley, in *Proceedings of the 11th International Conference on Electronic Commerce ICEC 2009*. A knowledge representation model for matchmaking systems in e-marketplaces, (2009), pp. 362–365
27. M Joshi, VC Bhavsar, H Boley, in *Multi-disciplinary Trends in Artificial Intelligence*. Compromise matching in P2P e-marketplaces: concept, algorithm and use case (Springer, 2011), pp. 384–394
28. M Joshi, VC Bhavsar, H Boley, in *Proceedings of the 12th International Conference on Electronic Commerce: Roadmap for the Future of Electronic Business*. Matchmaking in p2p e-marketplaces: soft constraints and compromise matching (ACM, 2010), pp. 148–154
29. J Jin, et al., in *High-Performance Computing in Asia-Pacific Region, 2005. Proceedings. Eighth International Conference on*. Towards a weighted-tree similarity algorithm for RNA secondary structure comparison (IEEE, 2005), pp. 639–644

30. H Kebriaei, VJ Majd, A new agent matching scheme using an ordered fuzzy similarity measure and game theory. Comput. Intell. **24**(2), 108–121 (2008)
31. L Yang, BK Sarker, VC Bhavsar, H Boley, in *14th International Conference on Intelligent and Adaptive Systems and Software Engineering (IASSE-2005), Proceedings of The International Society for Computers and Their Applications (ISCA)*. A weighted-tree simplicity algorithm for similarity matching of partial product descriptions, (2005), pp. 55–60