International Journal of
Emerging Research in Management &Technology
ISSN: 2278-9359 (Volume-5, Issue-6)

Research Article | **June 2016**

A Tree Similarity Algorithm for Semantic Search in Information Retrieval

[1]Pramodh Krishna D, [2]Dr. K Venugopal Rao
[1] Assist Professor, CSE, Priyadarshini College of Engineering, Nellore, Andhra Pradesh, India
[2] Professor, CSE, GNITS, Hyderabad, India

Abstract:

Ordered, labeled trees are trees in which each node has a label and the left-to-right order of its children (if it has any) is fixed. Although similarity search on textual data has been extensively studied, searching for similar trees is still an open problem due to the high complexity of computing the similarity between trees, especially for large numbers of tress Tree-structured data are becoming ubiquitous nowadays and manipulating them based on similarity is essential for many applications. 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.

Keywords: Similarity measures, weighted tree similarity, e-Learning, similarity measure, buyer-seller matching, arc-labeled trees, arc-weighted trees, Match-miss-measure

I. INTRODUCTION

The use of tree-structured data in modern database applications is attracting the attention of the research community. Typical examples of huge repositories of rooted, ordered and labeled tree-structured data include the secondary structure of RNA in biology or the XML data on the web. In this paper, we study the similarity measure and similarity search on large trees in huge datasets. These problems form the core operation for many database manipulations, such as, approximate join, clustering, k-NN classification, data cleaning, data integration etc. The present study employs a multi-agent architecture similar to Agent-based Community Oriented Routing Network (ACORN) (Marsh et al. 2003) as the foundation for semantic match-making (Sycara et al. 2001) and focuses on its central similarity algorithm for comparing RuleML-like message contents (Boley 2003).

In a multi-agent system such as ACORN (Marsh et al. 2003) a set of key words/phrases with their weights is used for describing the information an agent is carrying or seeking. Product/service advertising and requesting can be realized on top of sets of weighted key words/phrases. However, such a flat representation is limited in that it cannot represent tree-like product/service descriptions. For example, when we want to describe a car, we often provide its color, maker and model. The attribute color is independent of the maker of the car, while the

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. 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 use-ful for specific applications with highly constrained taxonomies. Information theory-based methods [12] extract 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]. Similarity, 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.

Trees must be transformed to an appropriate representation before the computation of their similarity. For a given application, buyer and seller trees must conform to the same standard schema in order to compute similarity. In a marketplace, the first step of a transaction between buyer and seller agents is to provide information that describes their requirements and offers. However, in a hybrid human-computer virtual marketplace, human buyers and sellers have to input that information. Thus, a user interface is needed for human buyers and sellers to input their descriptions. The interface implements a standard schema by generating instance trees only for well-formed input.

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.

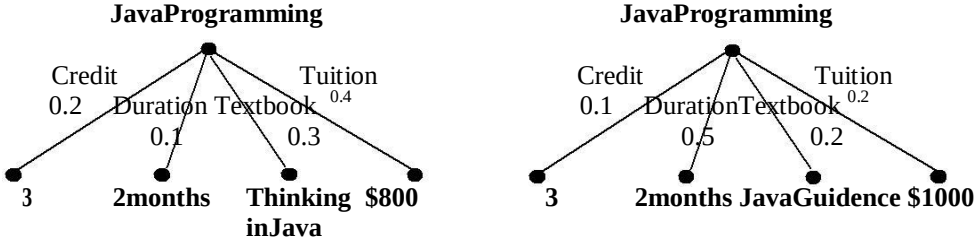
In this paper, we propose two generalized weighted tree similarity algorithms. We introduce a generalized formula to combine the matching and missing values in order to find the similarity between two taxonomy trees. It also determines the importance of the matching and missing value. The first generalized 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. The representation and generation of our node-labeled, arc-labeled, and arc-weighted trees are presented in Section 2. Section 3 briefly explains notations and definitions, followed by the weighted tree similarity algorithm. Many issues that need to be addressed while developing a similarity measure for our trees are discussed in Section 4. This section also presents our algorithm for computing the similarity of trees. Section 5 presents similarity results obtained with the Relfun implementation (included in the Appendix) of our algorithm. Finally concluding remarks are given in Section 6.

II. TREES

2.1 Representation

Various representations of trees and their matching are possible. To simplify the algorithm, we assume our trees are kept in a normalized form: the arcs will always be labeled in lexicographic (alphabetical) left-to-right order. The arc weights on the same level of any sub tree are required to add up to 1. Two flat example trees that describe the course "Java Programming" are illustrated in Figure 4 (a) and (b). To emphasize the difference between arc labels and node labels, node labels will always be bold-faced.



a) Tree of a learner agent. (b) Tree of a course agent.

Figure 1 Leaner and course trees.

Figure 1 (a) represents a tree carried by a learner agent. In this tree, the course this learner searches is "Java Programming". Sub trees stretching out from this root node represent the learner's preferences about this course. For example, this learner gives the arc "Tuition" the highest weight "0.4" relative to other arcs to express that cost will be the most important factor for his/her decision-making. The leaf node "$800" is the amount of money he/she is expecting.

This learner only gives the arc "Duration" (of 2 months) a rather low weight "0.1", which means that he/she does not care much about how long the course will last. The other two sub trees (leaves) are analogous.

In order to be applicable to real world description refinement, we do not limit the complexity, breadth or depth, of any sub tree. So, the trees in Figure 4 could have extra sub trees for the interaction language, prerequisites, etc., as well as a non-leaf "Textbook" sub tree mentioning a website etc.

Capturing these characteristics of our arc-labelled, arc-weighted trees, Weighted Object-Oriented RuleML, a RuleML version for OO modeling (Boley 2003), is employed for serialization in Web-based agent interchange. The XML child-subchild structure reflects the shape of our normalized trees and XML attributes are used to serialize the arc labels and weights. So, the tree in Figure 1 (b) will be serialized as shown in Figure 2 (a).

In Figure 2(a), the complex term (cterm) element serializes the entire tree and the _opc role leads to its root-node label, "Java Programming". Each child element **_slot** is a metarole, where the start tag contains the names and weights of arc labels as XML attributes **name** and **weight**, respectively, and the element content is the role filler serializing a subtree (e.g. a leaf). Consider the first **_slot** metarole of the cterm as an example. The attribute **name** has the value "Credit", describing the credit name of the course "Java Programming". The other attribute **weight**, with the value "0.1", endows the "Credit" branch with its weight. The content between the **_slot** tags is an ind (individual constant) serializing a leaf node labelled "3". Such weights have the interpretation of relative importance. For example, from a course's point of view the "Credit" weight means that the importance of the credit of this course is 0.1 relative to the other sub trees "Duration", "Textbook" and "Tuition", which have importance "0.5", "0.2" and "0.2", respectively.

```
<cterm>
  <_opc><ctor>JavaProgramming</ctor></_opc>
  <_slot name="Credit" weight="0.1"><ind>3</ind></_slot>
  <_slot name="Duration" weight="0.5"><ind>2months</ind></_slot> <_slot
  name="Textbook" weight="0.2"><ind>JavaGuidence</ind></_slot> <_slot
  name="Tuition" weight="0.2"><ind>$1000</ind></_slot>
</cterm>
```

(a) Tree serialization in Weighted OO RuleML.

```
cterm[ -opc[ctor[javaProgramming]], -
       slot[name[Credit],weight[0.1]][ind[3]], -
       slot[name[Duration],weight[0.5]][ind[2months]], -
       slot[name[Textbook],weight[0.2]][ind[javaGuidence]], -
       slot[name[Tuition],weight[0.2]][ind[$1000]]
       ]
```

(b) Tree representation in Relfun.

Figure 2. Symbolic tree representations.

2.2 Generation

It is clear that trees users have in mind must be transformed to the internal representation before computing their similarity. In order to make the similarity values reasonable and comparable, trees cannot be generated arbitrarily by buyers or sellers. For a specific application, e.g., e-Commerce, trees representing the information of buyers and sellers have to conform to the same standard schema. The standard schema for a specific application restricts what node labels and arc labels will be allowed in instance trees.

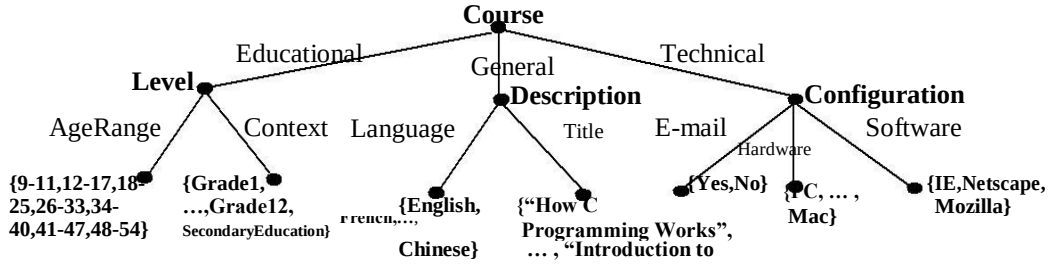


Figure 3. A standard tree schema for e-Learning.

Using e-Learning application as an example, course providers want to advertise their courses, while learners want to search appropriate courses. In order to describe a course, from our eduSource project experience, we need to specify the course name, course level, interaction language, etc. For the specific e-Learning application, Internet-enabled hardware and software are also needed. In our design of a standard schema for both learners and course providers, we tried to address concerns of both sides..

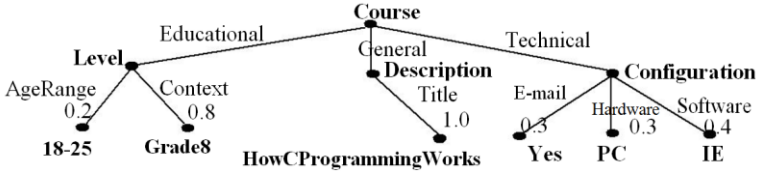


Figure 4. An instance tree generated from the interface in Figure 8.

In the above tree schema, we do not provide any arc weights because they are decided by the course providers and learners. For every leaf node, we give an enumation of potential values forlearners and course providers to select from. For example, for the arc "Language", learners or course providers can select English or the other listed languages as their favorite interaction language. For certain nodes, also 'built-in' types could be employed, for example, boolean or string.

III. SIMILARITY OF TREES

When developing a tree similarity measure, several issues have to be tackled because of the quite general shape of trees, their recursive nature, and their arbitrary sizes. The similarity function maps two such potentially very complex trees to a single real number ranging between 0 and 1. These issues are discussed in the first subsection with examples. The similarity algorithm is outlined in the subsequent subsection and fully listed in the appendix.

3.1Issues

In this subsection, we present six groups of characteristic sample trees that explore relevant issues for developing a similarity measure. Section 5 gives similarity values of these trees using the algorithm given in Section 3.2.

Example 1:

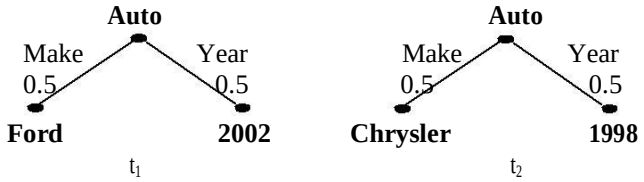
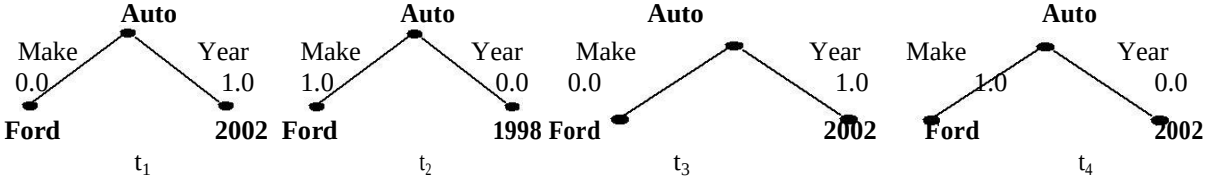


Figure 5. Two trees with mismatching leaves.

In Figure 5, tree t_1 and tree t_2 have the same root node label "Auto". But the node labels of subtrees (leaf nodes) are all different. In this example, although the two leaf nodes have numeric values, we do not carry out any arithmetic operation on these values to find out their closeness. Therefore, the similarity of these trees could be defined as zero. However, we have decided to award a user specifiable similarity 'bonus' reflecting the equality of the root nodes.

Example 2:



(a) Trees with opposite extreme weights. (b) Trees as in (a) but with identical leaves.

Figure 6. Trees with opposite branch weights.

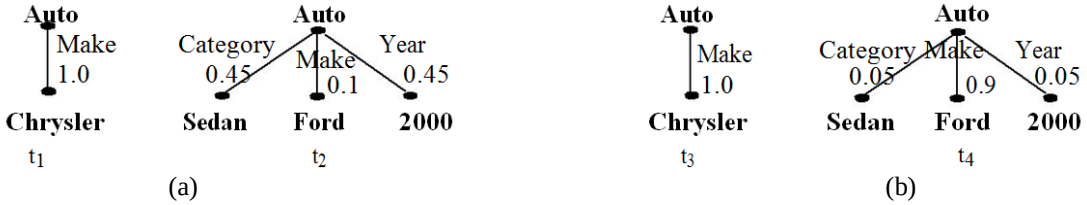
This example can be viewed as a modification of Example1. In Figure 6 (a), tree t_1 and tree t_2 have one identical subtree, the leaf "Ford," and therefore the similarity of these two subtrees could be considered as 1.0. However, we note that the weights of the arcs labeled "Make" are 0.0 versus 1.0. This indicates that the agent of tree t_1 puts no emphasis on the "Make" of the automobile, even though "Ford" is specified. The agent of tree t_2 puts the whole emphasis on the "Make" of the automobile. The averaged weight, using the *arithmetic mean*, of the corresponding branches is (0.0 + 1.0)/2 = 0.5. Our similarity measure for the "Make" branches, then, is defined using a pre-multiplier of value 1.0, because of the same label "Ford," as 1.0*(0.0 + 1.0)/2 = 0.5. We could have chosen to use the *geometric mean*, which would give zero branch similarity. However, we think that the branch similarity should be nonzero for identical subtrees. Since the leaf node labels for the "Year" branches are different we use a pre-multiplier of 0.0, and we obtain 0.0*(1.0 + 0.0)/2 = 0.0. Thus, the weights of the branches do not contribute to the similarity, as stated for Example 1. We consider the similarity of trees with one having an arc weight equal to 0.0 to be larger than the similarity of trees with one having a missing arc. Thus, the similarity $S(t_1, t_2)$ of the entire trees t_1, t_2 is defined as follows:

$S(t_1, t_2) = 1.0*(0.0 + 1.0)/2 + 0.0*(1.0 + 0.0)/2 = 0.5.$

In Figure 6 (b), tree t_3 and tree t_4 are the same as in Figure 10 (a) but have identical leaves. In this case, the trees are exactly the same except for their weights. In an e-Business environment this can be interpreted as follows. While the seller and buyer agents have attached opposite branch weights to reflect their subjective preferences, their autos represented are exactly the same. This implies that the similarity of the two trees should be equal to 1.0. Indeed, we obtain the similarity analogously to the case of (a), as follows:

$S(t_3, t_4) = 1.0*(0.0 + 1.0)/2 + 1.0*(1.0 + 0.0)/2 = 1.0.$

Example 3:



(a) (b)

Figure 7. Tree pairs only differing in arc weights.

Figures 7 (a) and (b) represent two pairs of trees only differing in the weights of the arcs of t_2 and t_4. In Figure 7 (a), t_1 has only one arc with label "Make," which also occurs in tree t_2. But their leaf node labels are different. The situation of Figure 7 (b) is the same as Figure 7 (a) except that the weight of the label "Make" is 0.9 in tree t_4, while it is 0.1 in tree t_2. On cursory look, the similarity of both pairs of trees should be identical because the leaf node differences between each pair of trees are identical. However, we should not overlook the contribution of the weights. In tree t_2, the weight of arc-label "Make" is much smaller than that in tree t_4. Thus, during the computation of similarity, the weight of the arc labeled "Make" should make a different contribution to the similarity: the importance of the "Chrysler-Ford" mismatch in Figure 7 (a) should be much lower than the same mismatch in Figure 7 (b). So, we expect $S(t_1, t_2) > S(t_3, t_4)$.

Example 4:

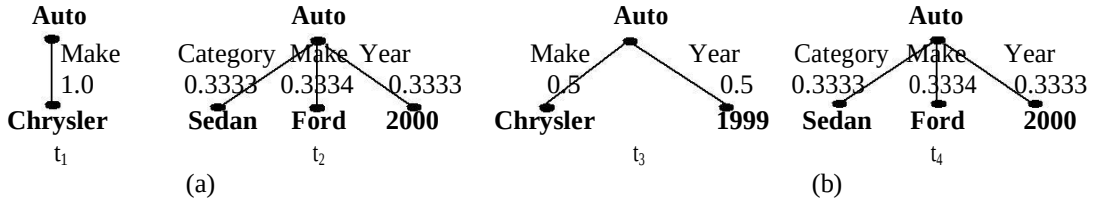


Figure 8. Tree pairs with left-tree refinement.

In Figure 8 (a), tree t_1 only has one arc, while tree t_3 in Figure 8 (b) has two arcs. Tree t_2 and tree t_4 are identical. However, in both pairs of trees, for identical arc labels, the leaf nodes are different. For example, both tree t_3 and tree t_4 have arc label "Year", but their node labels are "1999" and "2000", respectively. Tree t_1 only has one different subtree compared to tree t_2, but tree t_3 has two different subtrees compared to tree t_4. Therefore, $S(t_1, t_2) > S(t_3, t_4)$.

Example 5:

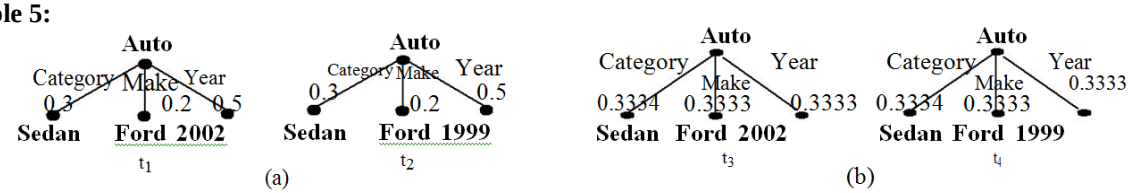


Figure 9. Tree pairs with the same structure.

In Figures 9 (a) and (b), trees t_1 and t_2 and trees t_3 and t_4 have almost the same structure except one pair of node labels "2002" and "1999". So, we are sure that node labels and arc labels cannot make $S(t_1, t_2)$ and $S(t_3, t_4)$ different. But for the arc label "Year", Figure 9 (a) and Figure 9 (b) have different weights which should lead to $S(t_1, t_2) < S(t_3, t_4)$, because of the higher mismatch of weights in Figure 9 (a) compared to Figure 9 (b).

Example 6:

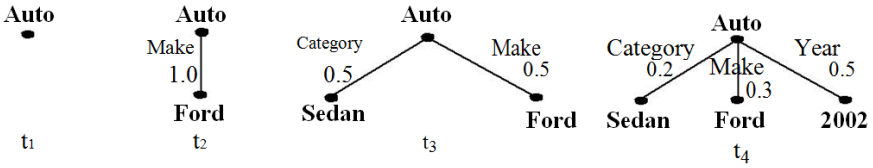


Figure 10. Trees with an increasing number of subtrees.

Figure 10 gives four sample trees with an increasing number of sub trees. Tree t_1 is an empty tree (a labeled tree) while the other trees have one or more branches. Intuitively, when comparing tree t_1 to the other trees, the more complex a tree, the smaller the tree's simplicity (we refer to it as treeplicity), and, consequently, the smaller its similarity with t_1. Thus, we initially expected that $S(t_1, t_2) > S(t_1, t_3) > S(t_1, t_4)$. Clearly, $S(t_2, t_3) > S(t_2, t_4)$. For a discussion of the actual results obtained see Section 5.

3.2 The Algorithm

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 rep-resented 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.

This subsection describes the existing weighted tree similarity algorithm The algorithm carries out a recursive top-down traversal comparison of two given normalized arc-labeled, arc-weighted trees 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 com-pares the labels of the root node of two given trees. 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. 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. Suppose l_i arc is associated with the sub-tree T_i, then the weighted simplicity value is calculated as $w(l_i)/2$ multiplied with the simplicity value of the sub-tree T_i . The simplicity algorithm 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.

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 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 and followed by the simplicity algorithm .

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 algorithm for simplicity function is given below:

Miss Algorithm λ(r, T)

d = 0.5, sum = 0;
if T is a leaf node then
Return (r);
else
{ let the tree T has m arcs and each arc has weight(l_i) and subtree T_i, for i = 1 to m.}
for each arc l_i of root node T do
sum = sum + weight(l_i) * λ(r ∗ d, T_i);
end for
sum=(1/m1)×sum;
Return sum;
end if

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 *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.

λ(r, 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 *T* specified as l_j, for $j = 1$ to *m*.
T_j is the sub tree attached to arc l_j.
$w(l_i)$- is the weight of arc l_i, for $j = 1$ to *m*.

IV. EXPERIMENTAL RESULTS

In this section we consider the examples discussed in Section 4.1 and use our implemented algorithm in Section 4.2 to obtain similarity values. Table 1 summarizes the similairity values obtained for all tree pairs. In our experiments we have set the node-equality parameter N to 0.1.

For Example 1, the complete mismatch of corresponding leaf nodes leaves only the user specified similarity 'bonus' of 0.1, reflecting the equality of the root nodes contributing to the similarity. The similarities of Example 2 (a) and (b) are obtained as 0.5 and 1.0, respectively, as desired (see Section 4.1).

Since the importance of the "Chrysler-Ford" mismatch in Figure 7 (a) of Example 3 should be much lower than the same mismatch in Figure 7 (b), we expected $S(t_1, t_2) > S(t_3, t_4)$. It turns out that $S(t_1, t_2)$ is more than two times bigger than $S(t_3, t_4)$.

For Example 4 we anticipated that $S(t_1, t_2) > S(t_3, t_4)$. Our program gives consistent results. Our results confirm the expected result of $S(t_1, t_2) < S(t_3, t_4)$ for the Example 5.

Example 6 consists of four trees and we first expected $S(t_1, t_2) > S(t_1, t_3) > S(t_1, t_4)$. However, we observe that we obtain $S(t_1, t_2) = S(t_1, t_3) = S(t_1, t_4)$. This can be explained as follows. Our constraint that weights on the same level add up to 1 together with our similarity-decreasing use of the weights imply that more branches in one of the trees do not necessarily make it less similar to the other tree since its weights must also be decreased. For instance, a single 1.0-

weighted branch is worth as much as two 0.5-weighted branches. Thus, the t_1 similarities with t_2, t_3, and t_4, are identical.

Table 1. Experimental results of the examples in Section 4.1.

Example	Tree	Tree	Similarity
Example 1	t_1	t_2	0.1
Example 2	t_1	t_2	(a) 0. 5
	t_3	t_4	(b) 1.0
Example 3	t_1	t_2	0.2823
	t_3	t_4	0.1203
Example 4	t_1	t_2	0.2350
	t_3	t_4	0.1675
Example 5	t_1	t_2	0.55
	t_3	t_4	0.7000
Example 6	t_1	t_2	0.3025
	t_1	t_3	0.3025
	t_1	t_4	0.3025
	t_2	t_3	0.8763
	t_2	t_4	0.8268

In order to prevent rapid depth degradation of the similarity for complex trees, we propose three arc similarity adjusting functions as similarity parameters to ensure that the results are reasonable and intuitive. This is exemplified here with two 2-level trees shown in Figure 15.

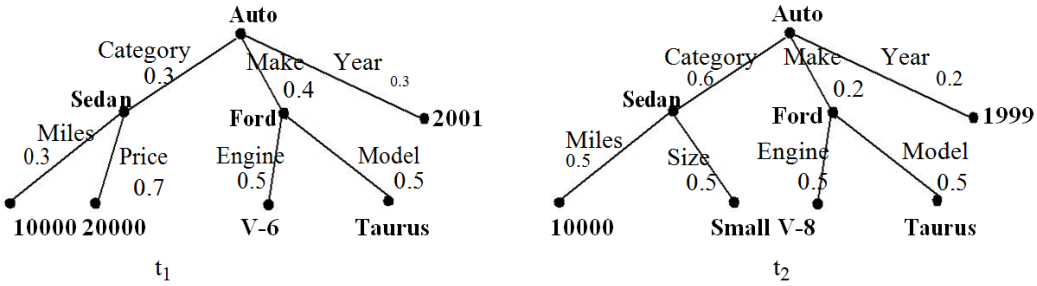


Figure 11. Applying three arc-adjusting functions to a pair of trees.

The square root function satisfies our assumption from Section 4.2 that $A(x) \geq x$ in the closed interval [0,1], and actually achieves $A(x) > x$ in the open interval (0,1). We also conducted experiments with compositions of root functions and logarithmic functions

V. CONCLUSION

The outlined multi-agent system is a novel architecture for agents to represent information and interact with each other. We focus on the metadata information these agents carry, and a similarity measurement over this information. In order to make the interaction between agents more meaningful and fine-grained, we chose trees to represent this information. More precisely, these trees are node-labelled, arc-labelled, and arc-weighted. The characteristics of these kinds of trees led us to use complex terms of Weighted Object-Oriented RuleML and XML attributes within them to represent arc labels and arc weights.

Our tree similarity algorithm, as part of semantic match-making between agents, computes the similarity of subtrees in a recursive way. It can be parameterized by different functions to adjust the similarity of these subtrees. This gives

agents the option to tailor their similarity measure for trees based on their subjective preferences. Other parameter functions could, e.g., be "local" similarity measures for leaf nodes of various types. The appendix gives the full definition of the extensible core algorithm in Relfun. This executable functional specification has proved to be a flexible test bed for our experiments. These experiments have given us meaningful results for e-Commerce/e-Learning environments. The algorithm can also be applied in other environments wherein weighted trees are used, and reimplemented in other languages, as we did in Java.

This multi-agent system has already been adapted for an electric power grid application with power plants as sellers and power distributors as buyers (Sarno et al. 2003). The tree representations and the similarity algorithm described in the current paper could be used unchanged in this new application. This has shown the flexibility of our match-making architecture for e-Business environments.

For expressing buyer queries, we have already generalized our trees to *non-ground trees* containing "Don't Care" leaves that have similarity value 1.0 with arbitrary corresponding seller trees. Furthermore, local similarity measures will be incorporated as a polymorphic functional parameter, based on the semantic leaf comparison of our eduSource e-Learning application (Boley et al. 2004). Finally, in a dynamic marketplace, it would be desirable to allow more flexible agent interactions. In future work we will thus consider user-definable buyer-seller interaction protocols.

REFERENCES

[1] Pramodh Krishna D and Venu Gopal Rao K. "Generalized weighted tree similarity algorithms for taxonomy trees." EURASIP Journal on Information Security 2016.1 (2016): 35.
[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.

