WEIGHTED PARTONOMY-TAXONOMY TREES WITH LOCAL SIMILARITY MEASURES FOR SEMANTIC BUYER-SELLER MATCH-MAKING

Lu Yang[1], Marcel Ball[1], Virendrakumar C. Bhavsar[1], Harold Boley[2]

[1] Faculty of Computer Science, University of New Brunswick, Fredericton, New Brunswick, Canada
[2] Institute for Information Technology e-Business, National Research Council, Canada, Fredericton, New Brunswick, Canada

{o6c11, marcel.ball, bhavsar}@unb.ca, harold.boley@nrc-cnrc.gc.ca

Abstract

A semantically enhanced weighted tree similarity algorithm for buyer-seller match-making is presented. First, our earlier global, structural similarity measure over (product) partonomy trees is enriched by taxonomic semantics: Inner nodes can be labeled by classes whose partial subsumption order is represented as a taxonomy tree that is used for similarity computation. In particular, the similarity of any two classes can be defined via the weighted length of the shortest path connecting them in the taxonomy. Instead of a taxonomy separate from the partonomy, we encode the taxonomy tree into the partonomy tree (as a "Classification" branch) in a way that allows us to directly reuse our partonomy similarity algorithm and to permit weighted (or 'fuzzy') taxonomic subsumption with no added effort. Second, leaf nodes can be typed and each type is associated with a local, special-purpose similarity measure realising the semantics to be invoked when computing the similarity of any two of its instances. We illustrate local similarity measures with e-Business types such as "Currency", "Address", "Date", and "Price". For example, the similarity measure on "Date"-typed leaf node labels translates various notations for date instances into a normal form from which it linearly maps any two to their similarity value. Finally, previous adjustment functions, which prevent similarity degradation for our arbitrarily wide and deep trees, are enhanced by smoother functions that evenly compensate intermediate similarity values.

Key Words/Phrases

global similarity measure, local similarity measure, arc-labeled and arc-weighted trees, taxonomic class similarity, taxonomy tree, Weighted Object-Oriented RuleML, adjustment function, arithmetic mean

1. Introduction

We have proposed earlier a weighted-tree similarity algorithm for multi-agent systems in e-Business environments [Bhavsar et al. 2004]. In a multi-agent system, buyer and seller agents seek matching by exchanging descriptions of products (e.g. key words/phrases) carried by them. One of our motivations is to remove the disadvantage of the flat representation that cannot describe complex relationship of product attributes. Therefore, we have proposed node-labeled, arc-labeled and arc-weighted trees to represent hierarchically structured product attributes. Thus, not only node labels but also arc labels can embody semantic information. Furthermore, the arc weights of our trees express the importance of arcs (product attributes). For the uniform representation and exchange of product trees we use a weighted extension of Object-Oriented RuleML [Boley 2003] to serialize them.

Previous tree similarity (distance) algorithms mostly dealt with trees that have node labels only [Liu and Geiger 1999; Lu 1979], whether they were ordered [Wang et al. 1998] or unordered [Shasha et al. 1994]. The Hamming Distance was also used in some approaches [Schindler et al. 2002] to compute the distance of node-labeled trees after deciding if a path exists for each pair of nodes. Due to our unique representation for product description, we have developed a different weighted-tree similarity algorithm.

In e-Learning environments, buyers and sellers are learners and learning object providers, respectively. Our algorithm has been applied to the eduSource project [Boley et al. 2005]. One goal of this project is to search procurable learning objects for learners. The search results for a learner are represented as a percentage-ranked list of learning objects according to their similarity values with the learner's query.

In our previous algorithm, similarity measures on both inner node labels and leaf node labels involve exact string matching that results in binary similarity values. We improved the exact string matching by allowing permutation of strings. For example, "Java Programming" and "Programming Java" are considered as identical node labels. However, inner node labels can be taxonomically divided into different classes based on their semantics and leaf node labels can be categorized as different types. Therefore, similarity measures on inner node labels and leaf node labels should be different.

In this paper we present three enhancements on our previous algorithm:

(a) We improve the inner node similarity measures by computing their taxonomic class similarities.
(b) For local similarity measures, as an example, a similarity measure on "Date"-typed leaf nodes that linearly maps two dates into a similarity value is given.
(c) Our improved adjustment functions approach limits more smoothly and compensate intermediate similarity values more evenly.

Our earlier global, structural similarity measure over (product) partonomy trees is enriched by taxonomic semantics: inner nodes can be labeled by classes whose partial subsumption order is represented as a taxonomy tree that is used for similarity computation. In particular, the similarity of any two classes can be defined via the weighted length of the shortest path connecting them in the taxonomy. The taxonomic class similarity of inner node labels also falls into the real interval [0.0, 1.0] where 0.0 and 1.0 indicate zero and total class matching, respectively.

The use of an extra taxonomy tree to compute class similarity might result in our partonomy similarity algorithm visiting taxonomy tree many times to compute the semantic similarity for each pair of inner node labels. Therefore, instead of a taxonomy separate from the partonomy, we encode the taxonomy tree into the partonomy tree (as a "Classification" branch) in a way that allows us to directly reuse our partonomy similarity algorithm and permits weighted (or 'fuzzy') taxonomic subsumption with no added effort. An application of this is our Teclantic portal (http://teclantic.cs.unb.ca) which aims to match projects according to project profiles represented as trees. A project profile contains both taxonomic and non-taxonomic descriptions. Both are embedded in a project partonomy tree as a taxonomic subtree and non-taxonomic subtrees describing project attributes (such as start date, end date, group size and so on). Our partonomy-tree similarity algorithm then finds matching projects for a given project.

Local similarity measures aim to compute similarity of leaf nodes. Leaf nodes can be typed and each type associated with a local, special-purpose similarity measure realising the semantics to be invoked when computing the similarity of any two of its instances. We illustrate local similarity measures with e-Business types such as "Currency", "Address", "Date", and "Price". For example, the similarity measure on "Date"-typed leaf node labels translates various notations for date instances into a normal form from which it linearly maps any two to their similarity value.

In order to prevent the similarity degradation during the bottom-up similarity computation, we provide various adjustment functions to increase the intermediate similarity values. This paper provides smoother adjustment functions, which evenly compensate intermediate similarity values.

For each pair of identical arc labels, we average their weights for computing the tree similarity. Arithmetic, geometric and harmonic means are possible weight average functions. However, based on their mathematical properties and our case studies, we have found that arithmetic mean is the one that generates more reasonable similarity results.

This paper is organized as follows. A brief review of our tree representation and tree similarity algorithm is presented in the following section. In Section 3, we discuss weight average functions and the improvements on adjustment functions. Section 4 presents our improvement on class similarity of inner nodes based on their taxonomic semantics and the encoding of taxonomy tree into partonomy tree. This section also presents a local similarity measure for "Date"-typed leaf nodes. Finally, concluding remarks are given in Section 5.

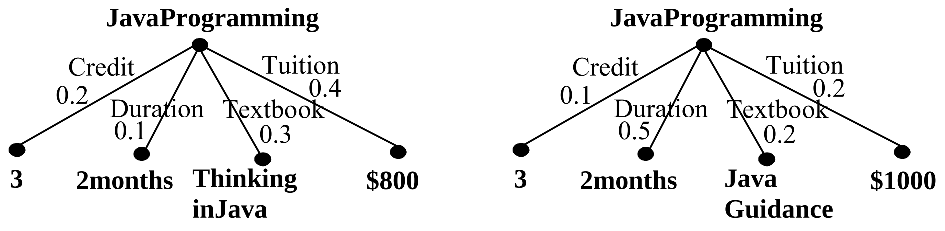
2. Background

In this section, we briefly review our tree representation for buyers as well as sellers and tree similarity algorithm for buyer-seller matching [Haddawy et al. 2004; Chavez and Maes 1996].

2.1. Tree Representation

Key words/phrases are commonly used to describe product advertising and requesting from sellers and buyers. However, we use node-labeled, arc-labeled and arc-weighted tree to represent the product descriptions because plain text is very limit to describe hierarchical relationships of product attributes. To simplify the algorithm, we assume our trees are kept in a normalized form: the arcs will always be labeled in lexicographic left-to-right order. The arc weights on the same level of any subtree are required to add up to 1.

Two flat example trees of learner and course provider that describe the course "JavaProgramming" are illustrated in Figure 1 (a) and (b). The learner and course provider describe their preferences by assigning different weights to different arc labels (course attributes). Thus, they specify which attributes are more or less important to them.



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

Figure 1. Learner and course trees.

Capturing these characteristics of our trees, Weighted Object-Oriented RuleML, a RuleML version for OO modelling [Boley 2003], is employed for serialization. So, the tree in Figure 1 (b) is serialized as shown in Figure 2.

```
<Cterm>
  <Ctor>JavaProgramming</Ctor>
  <slot weight="0.1">
     <Ind>Credit</Ind>
     <Ind>3</Ind>
  </slot>
  <slot weight="0.5">
     <Ind>Duration</Ind>
     <Ind>2months</Ind>
  </slot>
  <slot weight="0.2">
     <Ind>Textbook</Ind>
     <Ind>JavaGuidance</Ind>
  </slot>
  <slot weight="0.2">
     <Ind>Tuition</Ind>
     <Ind>$1000</Ind>
  </slot>
</Cterm>
```

Figure 2. Tree serialization in Weighted OO RuleML.

In Figure 2, the complex term (Cterm) element serializes the entire tree and the "Ctor" type leads to its root-node label, "JavaProgramming". Each multiple-valued "slot" role tag contains two pairs of "Ind" tags. Values between “Ind” tags correspond to arc labels and node labels underneath. The arc weights are represented by the “weight” attribute in “slot” tag.

2.2. Algorithm

In this subsection, we review the three main functions, treesim[N, A](t, t'), treemap[N, A](l, l') and treeplicity (i, t), of our previously proposed algorithm [Bhavsar et al. 2004]. The main function treesim[N, A](t, t') calls the ‘workhorse’ function treemap[N, A](l, l'), which co-recursively calls treesim; treemap also calls treeplicity(i, t) in some cases. The parameter “N” that serves as a node-equality fraction, which is a ‘bonus’ value from [0.0, 1.0], is added to the complementary fraction (1-N) of this subtree comparison (in this paper, the value of N is assumed to be 0.1). The functional parameter “A” specifies an adjustment function to prevent similarity degradation with depth deepening.

Generally speaking, our algorithm traverses input trees top-down (root-to-leaf) and then computes their similarity bottom-up. If two non-empty (sub)trees have identical root node labels, their similarity will be computed via treemap by a recursive top-down (root-to-leaf) traversal through the subtrees, t_i and t'_i, that are accessible on each level via identical arc labels l_i. The treesim recursion is terminated by two (sub)trees t and t' (root-to-leaf) that are leaf nodes or empty trees, in which case their similarity is 1.0 if their node labels are identical and 0.0 otherwise. Every tree is divided into some subtrees. So, the top-down traversal and bottom-up computation is recursively employed for every pair of subtrees.

In general, the arcs can carry arbitrary weights, w_i and w'_i from [0.0, 1.0]. For a pair of identical arc labels l_i and l'_i, their weights are averaged using the arithmetic mean, $(w_i + w'_i)/2$, and the recursively obtained similarity s_i of (sub)trees t_i and t'_i is multiplied by the averaged weight. Finally, on each level the sum of all such weighted similarities, $s_i(w_i + w'_i)/2$, is divided by the sum of all averaged weights.

However, during the computation of tree similarity, the intermediate s_i will become smaller and smaller because it always multiplies numbers between [0.0, 1.0]. Therefore, the final similarity value might be too small for two quite similar trees. In order to compensate similarity degradation for nested trees, an adjustment function A can be applied to s_i and we assume $A(s_i) \geq s_i$.

The tree similarity of trees t_1 and t_2, denoted as $S(t_1, t_2)$, is formally defined as follows when weights on the same level of both trees add up to 1.

$$S(t_1, t_2) = \sum (A(s_i)(w_i + w'_i)/2) \tag{1}$$

When a subtree in t_1 is missing in t_2 (or vice versa), function treeplicity is called to compute the simplicity of the single missing subtree. Intuitively, the simpler the single subtree in t_1, the larger its similarity to the corresponding empty tree in t_2. So, we use the simplicity as a contribution to the similarity of t_1 and t_2. When calling treeplicity with a depth degradation index i and a single tree t as inputs, our simplicity measure is defined recursively to map an arbitrary single tree t to a value from [0.0,1.0], decreasing with both the tree breadth and depth. The recursion process terminates when t is a leaf node or an empty tree. For a non-empty (sub)tree, simplicity will be computed by a recursive top-down traversal through its subtrees. Basically, the simplicity value of t is the sum of the simplicity values of its subtrees multiplied with arc weights from [0.0, 1.0], a subtree depth degradation factor ≤ 0.5, and a subtree breadth degradation factor from (0.0, 1.0].

For any subtree t_i underneath an arc l_i, we multiply the arc weight of l_i with the recursive simplicity of t_i. To enforce smaller simplicity for wider trees, the reciprocal of the tree breadth is used on every level as the breadth degradation factor. On each level of deepening, the depth degradation index i is multiplied with a global depth degradation factor treeplideg ≤ 0.5 (= 0.5 will always be assumed here), and the result will be the new value of i in the recursion.

However, this algorithm only takes into account the global similarity measure. Within the global similarity measure, for each pair of inner node labels, the exact string matching which leads to 0.0 or 1.0 similarity does not semantically embed their taxonomy class similarity. For local similarity measure which computes leaf node similarity, this algorithm does not handle different types of nodes using different similarity measures, but still the exact string matching. Adjustment functions of this algorithm do not provide good curves for compensating the similarity decreasing. Other weight combination functions such as geometric and harmonic means could potentially replace the arithmetic mean employed currently. The discussions and improvements about these issues are given in Sections 3 and 4.

3. Kernel Algorithm Revisited

We revisit here our kernel algorithm by discussing the weight average functions and improvement on adjustment functions. Note that our algorithm treats buyer and seller trees symmetrically as necessary to obtain a classical metric. In section 4.3 we will show that some seemingly asymmetric buyer/seller tree attributes can be made symmetric. Mathematically, for two given positive real numbers, arithmetic mean of them is always greater than the results generated from geometric and harmonic means. From the point of view of compensating similarity degradation, arithmetic mean is the most appropriate one. Furthermore, geometric and harmonic means overlook the overlapping interests of buyers and sellers in some cases. By our case studies, arithmetic mean always generates more reasonable similarity values.

The adjustment function is improved for the purpose of preventing similarity degradation with depth deepening during the similarity computation. We provide smoother adjustment functions that evenly compensate the intermediate similarity values.

3.1. Weight Averaging Functions

As mentioned in Section 2.2, for every pair of identical arc labels, we use arithmetic mean to average their arc weights during the similarity computation. Two possible options are geometric and harmonic means. Due to their different mathematical properties, different similarity values are produced. In this subsection, we discuss them from both mathematical and practical point of view.

Given a set of positive real numbers $\{x_1, x_2, \ldots, x_n\}$, the arithmetic, geometric and harmonic means of these numbers are defined as

$$A(x_1, x_2, \ldots, x_n) = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2}$$

$$G(x_1, x_2, \ldots, x_n) = \left(\prod_{i=1}^{n} x_i\right)^{1/n} \tag{3}$$

$$H(x_1, x_2, \ldots, x_n) = n \Big/ \sum_{i=1}^{n} \frac{1}{x_i} \tag{4}$$

Since arc weights are combined pair by pair in our algorithm, we represent weights of a pair as w_i and w_i'. Furthermore, we denote the arithmetic, geometric and harmonic means of a pair of arc weights with identical arc label *l* by *AM*(*l*), *GM*(*l*) and *HM*(*l*).

$$AM(l) = \frac{1}{2}(w_i + w_i') \tag{5}$$

$$GM(l) = (w_i \times w_i')^{1/2} \tag{6}$$

$$HM(l) = \frac{2w_i w_i'}{w_i + w_i'} \tag{7}$$

Equations (5), (6) and (7) satisfy

$$AM(l) \geq GM(l) \tag{8}$$

$$HM(l) = \frac{(GM(l))^2}{AM(l)} \tag{9}$$

Note that

$$AM(l) \geq GM(l) \geq HM(l) \tag{10}$$

Using geometric and harmonic means leads to two new similarity measures

$$GS\,(t_1, t_2) = \sum\,(A(s_i)\;(w_i \times w_i')^{1/2}) \tag{11}$$

$$HS\,(t_1, t_2) = \sum\,(A(s_i)\;(\frac{2w_i w_i'}{w_i + w_i'})) \tag{12}$$

From the point of view of compensating the degradation of similarity computation, arithmetic mean is preferred because it provides higher similarity value than the other two means according to equation (10). However, example below shows that geometric and harmonic means are more reasonable.

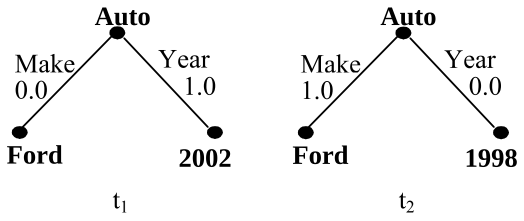


Figure 3. Trees with opposite extreme weights.

In this subsection, for the ease of computation, we use $A(s_i) = s_i$ for similarity computation. We assume user1 and user2 correspond to trees t_1 and t_2 in Figure 3, respectively. User1 puts all the weight on 2002 so that he really does not care the make of the automobile. It seems that an automobile which is not 2002 is of no interest to him. However, user2 puts all the weight on the fact that it must be a Ford and the year 1998 is different from 2002 specified by user1. Intuitively, the car user1 has is of no interest to user2. Table 1 shows the combined weights after applying the three weight averaging functions.

Weight Averaging Functions	Averaged Values
AM(Make)	0.5
AM(Year)	0.5
GM(Make)	0.0
GM(Year)	0.0
HM(Make)	0.0
HM(Year)	0.0

Table 1. Averaged weights for trees in Figure 3.

Using Equations (1), (11) and (12), we obtain $S(t_1, t_2) = 0.5$, $GS(t_1, t_2) = 0.0$ and $HS(t_1, t_2) = 0.0$. It seems that we get "correct" results from geometric and harmonic means because for totally different interests they result in similarity 0.0.

However, the attributes (arc labels) of products are independent. When we compare the subtrees stretching from "Make" arcs, we should not take other arcs into account. Trees t_1 and t_2 in Figure 4 show the two "Make" subtrees picked out from Figure 3.

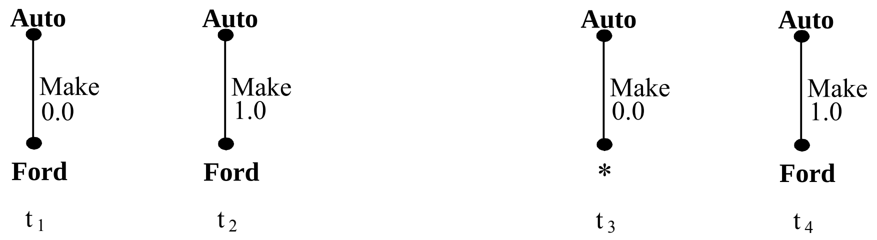


Figure 4. "Make" subtrees from Figure 3.

Tree t_3 is generated by replacing the node label "Ford" in tree t_1 with "*" which represents a "Don't Care" value of the arc "Make". In our algorithm, the similarity of "Don't Care" (sub)tree and any other (sub)tree is always 1.0. Tree t_4 is identical to t_2. The "Don't Care" node in tree t_3 implies that user1 accepts any make of automobile. Therefore, the "Ford" node in tree t_4 perfectly matches the "Don't Care" node in t_3. In tree t_1, although user1 puts no emphasis on the "Make" of the automobile which indicates that "Make" is not at all important to him, he still prefers a "Ford" car which is identical to user2's preference. The "Ford - Ford" comparison indicates more specifically of their identical interests on "Make" than the "Don't Care - Ford" comparison. Thus, the geometric and harmonic means which lead to zero similarity are not reasonable.

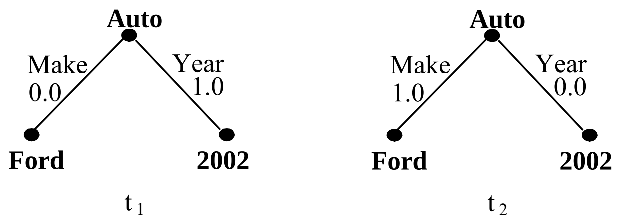


Figure 5. Trees with opposite weights but identical node labels.

Trees t_1 and t_2 in Figure 5 have totally identical node labels which mean that user1 and user2 have the same interests. Although these two trees have opposite extreme arc weights, user1 and user2 have complementary but compatible interests. Their similarity should be 1.0. However, we only get $GS(t_1, t_2) = 0.6$ and $HS(t_1, t_2) = 0.36$ that are too low for representing users' totally identical interests. Using arithmetic mean, $S(t_1, t_2) = 1.0$. Based on the discussion in this subsection, we choose the arithmetic mean for weight averaging.

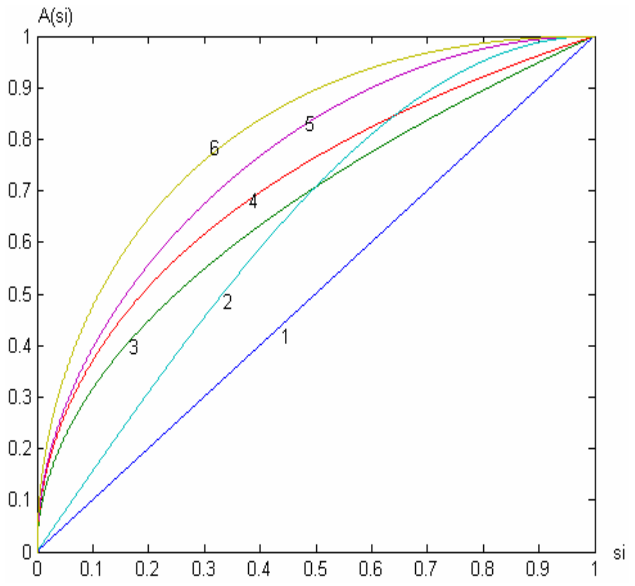
3.2. Adjustment Function

The adjustment function A, which is monotonically increasing, satisfies $A(s_i) \geq s_i$ for compensating similarity decrease during the bottom-up similarity computation. Because $A(s_i)$ is the adjusted value of s_i to continue the similarity computation, it also falls into the real interval [0.0, 1.0].

In Figure 6, function 1 (identity function) is given as $A(s_i) = s_i$. The function $A(s_i) = \sqrt[n]{s_i}$ is a good candidate for obtaining $A(s_i)$ that is greater than s_i. Using the square root function shown as function 3 we cannot get even adjustment because its plot becomes flatter and flatter when s_i approaches 1. Similarly, when s_i approaching 0, another function $A(s_i) = \sin\frac{\pi}{2}s_i$ represented by function 2 also has such a linear-like characteristic. In order to obtain a plot that approaches limits smoothly, we combine $\sqrt{s_i}$ and $\sin\frac{\pi}{2}s_i$ and obtain two functions $A(s_i) = \sqrt{\sin\frac{\pi}{2}s_i}$ and $A(s_i) = \sin(\frac{\pi}{2}\sqrt{s_i})$.

Another function that could be employed is $A(s_i) = \sqrt{\log_2(s_i+1)}$. The reason that we combine the square root function with logarithmic function is that logarithmic function itself does not provide significant increment according to our experiments. We do not combine $\sqrt[3]{s_i}$ or $\sqrt[4]{s_i}$ with other functions because they result in too high similarity values. All plots intersect at two points with coordinates (0, 0) and (1, 1). Therefore, except function 2, all other functions have relationship $\sin(\frac{\pi}{2}\sqrt{s_i}) \geq \sqrt{\sin\frac{\pi}{2}s_i} \geq \sqrt{\log_2(s_i+1)} \geq \sqrt{s_i} \geq s_i$. The plot of function $\sin\frac{\pi}{2}s_i$ has two more intersections with the plots of $\sqrt{s_i}$ and $\sqrt{\log_2(s_i+1)}$.

We allow users to select adjustment functions to get higher or lower similarity values, however we recommend A(s_i) = $\sin(\frac{\pi}{2}\sqrt{s_i})$ and A(s_i) = $\sqrt{\sin\frac{\pi}{2}s_i}$.



1: A(s_i) $= s_i$

2: A(s_i) $= \sin\frac{\pi}{2}s_i$

3: A(s_i) $= \sqrt{s_i}$

4: A(s_i) $= \sqrt{\log_2(s_i+1)}$

5: A(s_i) $= \sqrt{\sin\frac{\pi}{2}s_i}$

6: A(s_i) $= \sin(\frac{\pi}{2}\sqrt{s_i})$

Figure 6. Adjustment functions.

4. Global Similarity and Local Similarity

In order to improve the binary similarity value of exact string matching of node labels in our previous algorithm, we implemented the permutation of strings. Once two node labels have overlapping strings, their similarity is above 0.0. However, both exact string matching and permutation of strings do not take into account the taxonomic semantics of nodes labels and they handle inner nodes and leaf nodes in the same way.

We enrich our earlier global, structural similarity measure over (product) partonomy trees by taxonomic semantics. Inner nodes can be located as classes in a taxonomy tree which represents the weighted hierarchical relationship of them. Thus, the semantic similarity of two inner node labels is transformed into the similarity of two classes in the taxonomy tree. The similarity of any two classes can be defined via the weighted length of the shortest path connecting them in the taxonomy.

Since the taxonomy is separate from the partonomy, our partonomy similarity algorithm might visit taxonomy tree for the semantic similarity computation of each pair of inner node labels. Therefore, we encode the taxonomy tree into the partonomy tree (as a “Classification” branch) in a way that allows us to directly reuse our partonomy similarity algorithm and permits weighted (or 'fuzzy') taxonomic subsumption with no added effort.

Leaf node labels can be divided into different types, such as “Currency”, “Address”, “Date” and “Price”. We improve the local similarity measure on “Date”-typed leaf nodes that linearly maps two arbitrary dates into a similarity value.

4.1. Taxonomic Class Similarity

The taxonomic class similarity stands for the similarity of semantics of two inner node labels. The value of taxonomic class similarity also falls into the real interval [0.0, 1.0] where 0.0 and 1.0 indicates totally different and identical class matching, respectively. As long as a pair of node labels has overlapping semantics, the value of its taxonomic class similarity should be greater than 0.0. For example, as we mentioned, "Java Programming" and "C++ Programming" can be in the same class "Object-Oriented Programming" and they should have a non-zero class similarity. However, although "Prolog Programming" is located in a difference class "Logic Programming", it still has non-zero but smaller class similarity with "Java Programming" and "C++ Programming" because all of them are in the same class "Programming".

For the ease of explanation, we limit our discussion to a small set of the ACM Computing Classification System (http://www.acm.org/class/1998/ccs98.txt). According to its classification of "Programming Techniques", we create a taxonomy tree shown in Figure 7.

Given the taxonomy tree of "Programming Techniques", we find the taxonomic similarity of two classes by looking into the tree and measure the path product between the classes in the taxonomy. If one class is the direct parent of the other, a fuzzy subsumption value between them can be assigned by machine learning algorithms or a human expert (at present, we are developing techniques for automatic generation of fuzzy subsumption from HTML documents [Singh 2005]). Otherwise, the taxonomic similarity of them is the product of similarity values along the shortest path between the two classes. For taxonomy trees, note that we do not limit the sum of the similarity at the same level of a subtree to a fixed number (such as 1.0).

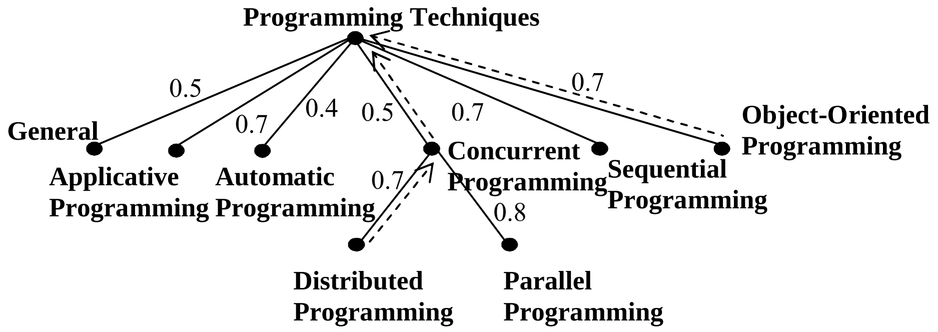


Figure 7. Taxonomy tree of "Programming Techniques".

Suppose the two trees t_1 and t_2 in Figure 8 represent the courses requested and offered by a learner and a course provider, respectively. The two root node labels "Distributed Programming" and "Object-Oriented Programming" are two classes located in different subtrees in the taxonomy tree of Figure 7. Therefore, in order to compute the taxonomic similarity of them, we compute the path product from node "Distributed Programming" to "Object-Oriented Programming". The shortest path is shown by dashed lines with arrows. After multiplying all similarity values on this path, we get their taxonomic class similarity $0.7 \times 0.5 \times 0.7 = 0.245$.

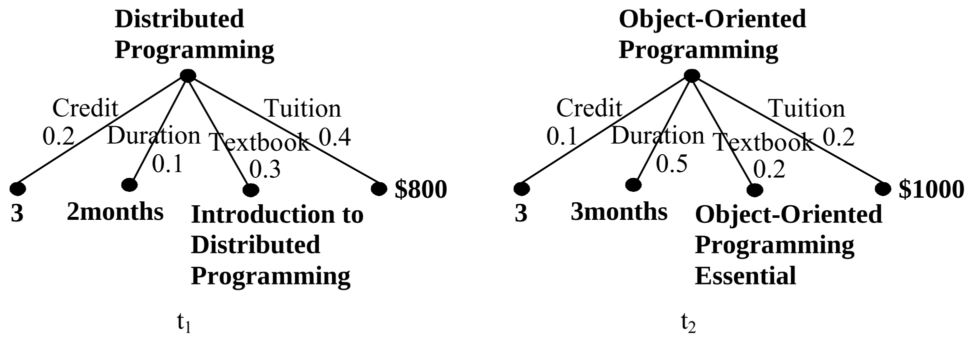


Figure 8. Trees of a learner and a course provider.

4.2. Encoding Taxonomy Tree into Partonomy Tree

The disadvantage of taxonomic class similarity measure is that there is an extra taxonomy tree that is separate from partonomy trees. Thus, during the similarity computation of two trees, partonomy similarity algorithm might look into the taxonomy tree quite a few times to compute the taxonomic class similarity for each pair of inner nodes. Therefore, we encode the taxonomy tree into our partonomy tree (as a "Classification" branch) in a way that allows us to directly reuse our partonomy similarity algorithm and to permit weighted (or 'fuzzy') taxonomic subsumption with no added effort.

We use the taxonomy tree in Figure 7 as an example. Since the taxonomy tree is encoded into the partonomy tree, it must be arc-labeled, arc-weighted. Figure 9 shows a modification of the tree in Figure 7. Classes are represented as arc labels. Each class is assigned an arc weight by user. All arc weights at the same level of a subtree sum up to one. All node labels except the root node label are changed into "Don't Care".

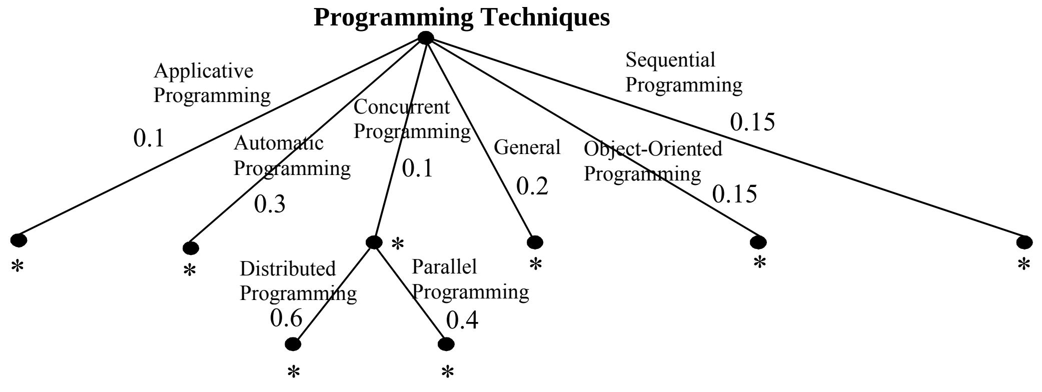


Figure 9. Taxonomy tree of "Programming Techniques" for encoding.

In Figure 10 two example course trees are presented, each with a sub-section of the ACM taxonomy from Figure 9 embedded under the "Classification" branch. This taxonomic sub-tree represents the areas from the ACM classifications that are related to the materials of the course, with the weights representing the relative significance of those areas. With our taxonomy trees embedded into our regular user similarity trees the taxonomic descriptions of the courses also contribute to the similarity computation like other non-taxonomy subtrees such as "Duration", "Tuition" and so on. The Teclantic project uses this technique to allow users to classify their projects using a taxonomy of research and development areas.

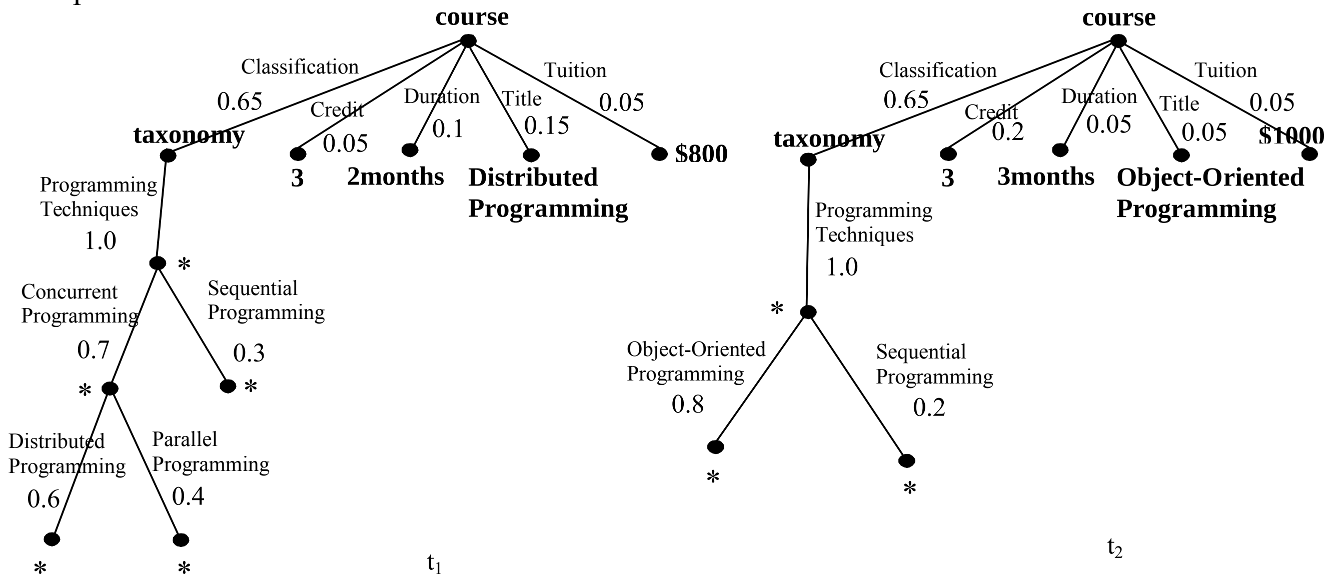


Figure 10. Two course trees with encoded taxonomy subtrees.

4.3. Local Similarity

Another enhancement of our algorithm is the addition of local similarity measures ─ the similarity of leaf node labels. In our previous algorithm, similarity of leaf nodes is also obtained from exact string matching (which results in binary result) or permutation of strings. However, different types of leaf node labels need different types of similarity measures.

Using "Price"-typed leaf node labels as an example, the similarity of two such nodes should conform to our intuitions. As we have seen in Figure 8, if a learner wants to buy a course for $800, a $1000 offer from a course provider does not lead to a successful transaction. However, both of them will be happy if the course provider asks $800 and the learner would be willing to buy it for $1000. It seems that the buyer and seller tree similarity under the "tuition" attribute is asymmetric. However, we can transform this asymmetry into symmetry. Generally, buyers offer the maximum price they can accept and sellers provide the minimum price they are pleased with. We define the price ranges of them by two intervals [0, Max] for buyers and [Min, ∞] for sellers. Intuitively, only if Min ≤ Max the transaction will be successful. If this condition holds, the similarity of them should be a number above 0.0. Otherwise, it is 0.0. Thus, Figure 8 could be changed by having "≤$800" and "≥$1000" as the leaf node labels under the "tuition" attribute in t_1 and t_2, respectively.

Here, we give an example of a local similarity measure on "Date"-typed leaf nodes.

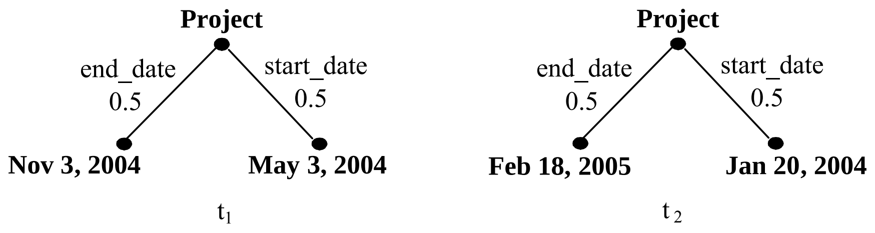


Figure 11. "date" subtrees of projects.

We define a comparison for "Date"-typed node labels. Figure 11 shows two trees that describe the start dates and end dates of two projects. They are segments of the trees we use in our Teclantic project. The corresponding WOO RuleML representation of tree t_1 that describes the dates and date comparison handler is shown in Figure 12.

```
<Cterm>
    <Ctor>Project</Ctor>
    <slot weight="0.5">
        <Ind >end_date</Ind>
        <Ind handler="date">Nov 3, 2004</Ind>
    </slot>
    <slot weight="0.5">
        <Ind >start_date</Ind>
        <Ind handler="date">May 3, 2004</Ind>
    </slot>
</Cterm>
```

Figure 12. WOO RuleML representation of tree t_1 in Figure 11.

The "handler" attribute of the "Ind" tag tells our algorithm that a special local similarity comparison should be conducted. In this case the "date" comparison. "Date"-typed node labels can be easily transformed into integer values thus their difference can be computed. If we use d_1 and d_2 to denote the integer values of dates date1 and date2, the similarity of date1 and date2, $DS(d_1, d_2)$, can be computed by the following formula.

$$DS(d_1, d_2) = \begin{cases} 0.0 & \text{if } |d_1 - d_2| \geq 365, \\ 1 - \dfrac{|d_1 - d_2|}{365} & \text{otherwise.} \end{cases} \qquad (13)$$

If $|d_1 - d_2|$ is equal to 0, the date similarity is 1.0. If $|d_1 - d_2|$ is equal to or greater than 365, the date similarity is assumed as 0.0 for the purpose of this illustration. Other values of $|d_1 - d_2|$ are mapped linearly between 0.0 and 1.0. Using this date similarity measure and our tree similarity algorithm, the similarity of trees t_1 and t_2 in Figure 11 is 0.74.

5. Conclusion

In our previous tree similarity algorithm, both the inner node and leaf node comparisons are exact string matching which produces binary results that cannot indicate a continuous range of semantic similarity of nodes. Although we implemented the permutation of strings for node label comparisons, they do not match node labels semantically. Previous adjustment functions do not adjust intermediate similarity values evenly.

The enhanced tree similarity algorithm proposed in this paper improves the semantic similarity of inner node labels by computing their taxonomic class similarity and the leaf node similarity by applying different local similarity measures to different types of nodes. We also optimize the adjustment functions and analyze three possible functions for weight averaging.

Taxonomy tree is employed to compute the similarity of semantics of inner nodes. Inner nodes are classes whose partial subsumption order is represented as a taxonomy tree that is used for similarity computation. The class similarity of two inner node labels is computed by finding two corresponding classes in the taxonomy tree and computing the product of the similarity along the shortest path between these two classes. In order to avoid combinatorial explosion when finding a common ancestor for two classes, our taxonomic class similarity measure does not currently allow multiple inheritance.

The extra computation of class similarity in the taxonomy tree can be removed by encoding the taxonomy tree into partonomy tree as subtree. Thus, the overall tree similarity computed by our partonomy similarity algorithm contains contributions from not only the taxonomy subtrees but also the non-taxonomy subtrees.

Local similarity measures on leaf nodes are computed by employing special similarity measures suited for node types. Our future work includes the development of additional local similarity measures for leaf nodes.

The proposed adjustment functions evenly adjust the intermediate similarity values and approach limits smoothly. We have selected arithmetic mean for averaging arc weights because, compared to geometric and harmonic means, it not only compensates the similarity degradation but also leads to more reasonable similarity values.

At present, we are investigating the properties of our similarity measure, including its tree simplicity module, as well as their parameters (e.g. various adjustment functions) and limitations [Yang et al. 2005].

Acknowledgements

We thank Zhongwei Sun and Qian Jia for their preparatory work on taxonomic class similarity, as well as Daniel Lemire for discussion on weight averaging functions and Michael Richter, for many discussions, especially about CBR and global versus local similarities. We also thank the NSERC for its support through discovery grants of Virendra C. Bhavsar and Harold Boley.

Duration

erences

Bhavsar, V. C., H. Boley and L. Yang, A weighted-tree similarity algorithm for multi-agent systems in e-business environments. *Computational Intelligence*, 2004, 20(4):584-602.

Boley, H., Functional-Logic integration via minimal reciprocal extensions. *In* Theoretical Computer Science 212, Elsevier, 1999, 77-99.

Boley, H., Object-Oriented RuleML: User-level roles, URI-grounded clauses and order-sorted terms, Springer-Verlag, Heidelberg, LNCS-2876, 2003, 1-16.

Boley, H., V. C. Bhavsar, D. Hirtle, A. Singh, Z. Sun and L. Yang, A match-making system for learners and learning Objects. *Learning & Leading with Technology,* International Society for Technology in Education, Eugene, OR, 2005 (to appear).

Chavez, A. and P. Maes, Kasbah: An agent marketplace for buying and selling goods. *In* Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, London, 1996, 75-90.

Haddawy, P., C. Cheng, N. Rujikeadkumjorn and K. Dhananaiyapergse, Balanced matching of buyers and sellers in e-Marketplaces: the barter trade exchange model. *In* Proceedings of International Conference on E-Commerce (ICEC04), Delft, Netherlands, Oct 2004.

Liu, T. and D. Geiger, Approximate tree matching and shape similarity. *In* Proceedings of the Seventh International Conference on Computer Vision, Kerkyra, 1999, 456-462.

Lu, S., A tree-to-tree distance and its application to cluster analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1979, *PAMI*-1(2):219-224.

Schindler, B., F. Rothlauf and H. J. Pesch, Evolution strategies, network random keys, and the one-max tree problem. *In* Applications of Evolutionary Computing: EvoWorkshops, *Edited by* Stefano Cagnoni, Jens Gottlieb, Emma Hart, Martin Middendorf and Gunther R. Raidl. Springer, Vol. 2279 of LNCS, 2002, 143-152.

Shasha, D., J. Wang and K. Zhang, Exact and approximate algorithm for unordered tree matching. *IEEE Transactions on Systems, Man and Cybernetics*, 1994, 24(4):668-678.

Singh, A., Weighted tree metadata extraction. Master Thesis (in preparation), University of New Brunswick, Fredericton, Canada, 2005.

Wang, J., B. A. Shapiro, D. Shasha, K. Zhang, and K. M. Currey, An algorithm for finding the largest approximately common substructures of two trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1998, 20:889-895.

Yang, L., B. K. Sarker, V. C. Bhavsar and H. Boley, A weighted-tree simplicity algorithm for similarity matching of partial product descriptions (submitted for publication), 2005.