Journal of Universal Computer Science, vol. 15, no. 9 (2009), 1766-1781
submitted: 15/8/08, accepted: 25/4/09, appeared: 1/5/09 © J.UCS

A Tree Similarity Measuring Method and its Application to Ontology Comparison

Yunjiao Xue
(University of Western Ontario, London, Ontario, Canada
yxue24@uwo.ca)

Chun Wang
(Concordia University, Montreal, Quebec, Canada
cwang@ciise.concordia.ca)

Hamada H. Ghenniwa
(University of Western Ontario, London, Ontario, Canada
hghenniwa@eng.uwo.ca)

Weiming Shen
(University of Western Ontario, London, Ontario, Canada
wshen@uwo.ca)

Abstract: Classical tree similarity measuring approaches focus on the structural and geometrical characteristics of the trees. The degree of similarity between two trees is measured by the minimal cost of editing sequences that convert one tree into the other one from pure structural perspective. Differently, when the trees are created to represent concept structures in a knowledge context (known as concept trees), the tree nodes represent concepts, not merely abstract elements occupying specific positions. Therefore, measuring similarity of such trees requires a more comprehensive method which takes the position, significance of the concepts (represented by the tree nodes), and conceptual similarity among the concepts from different trees into consideration. This paper extends the classical tree similarity measuring method to introduce tree transformation operations which transform one concept tree to another one. We propose definitions for the costs of the operations based on the position, importance of each concept within a concept structure, and similarity between individual concepts from different concept structures in a knowledge context. The method for computing the transformation costs and measuring similarity between different trees is presented. We apply the proposed method to ontology comparison where different ontologies for the same domain are represented as trees and their similarity is required to be measured. We show that the proposed method can facilitate the initiation of ontology integration and ontology trust evaluation.

Keywords: Tree, Similarity Measuring, Transformation Operation, Transformation Cost, Ontology Integration, Ontology Comparison
Categories: M.1

1 Introduction

Ontologies have been recognized as a fundamental infrastructure for advanced approaches to knowledge management [Arroyo, 07]. An ontology specifies a conceptualization of a domain in terms of concepts and their relationships [Guarino,

97]. Ontology can create an agreed-upon vocabulary for sharing knowledge, exchanging information, and eliminating ambiguity. An ontology can be viewed as a concept structure which contains concepts identified within a domain and relationships associating the concepts. It plays a very important role in many areas such as Collaborative Design [Shen, 01].

In practice, it is not common that a shared ontology is provided for a specific domain and widely accepted. Contrarily, usually different communities may build their own ontologies that are heterogeneous in terms of structure and semantics, even the ontologies commit to the same conceptualization of that domain. Such heterogeneity is also known as conceptualization mismatch [Visser 97], which is a difference in the way a domain is interpreted (conceptualized), resulting in different ontological concepts or different relationships between those concepts due to different interests. For instance, in the domain of collaborative design, various design tools and standards are employed and their ontologies are often heterogeneous, which may hinder integration of the designs.

Ontology integration [Pinto, 99] is developed to address such heterogeneities. Ontology integration refers to building a larger and more complete ontology reusing available ontologies from ontology libraries which serves at a higher level than that of the source ontologies. It is a very complex process as a part of the ontology development lifecycle [Pinto, 04].

In some cases the ontologies need to be compared to support initialization of the ontology integration process and ontology trust evaluation, e.g., finding one ontology among a group of ontologies that is more similar to many of others and taking is as the foundation of the integration.

An ontology can be viewed as a concept structure representing some domain knowledge [Sanin, 07], and one commonly used form is a tree structure. Much of the research on comparing trees uses editing cost from one tree to another to measure the similarity of two trees [Guegan, 06]. The classical methods focus on the structural and geometrical characteristics of the trees, mainly considering the number of nodes affected by the tree editing operations [Allali, 04 and Guda, 02]. However, in a knowledge context where the trees are used to represent the concept structures, in addition to the structural characteristics of the trees, more attention must be paid to the concepts represented by the internal tree nodes. Therefore, besides the number of edited nodes, the positions and conceptual similarities of the affected nodes also have to be considered.

The similarity of two individual concepts can be relatively easily estimated by domain experts. As an example, based on common sense, concepts "*People*" and "*Human*" are often regarded as referring to the same meaning, i.e. their similarity degree is 1. On the other hand, concept "*Faculty*" is not always exactly referring to the same thing as "*Professor*" in the university domain. Roughly speaking, a similarity degree can be assigned to these two concepts, say, 0.9, meaning that under around 90% occasions they are describing the same group but not in other cases. Some researches have also proposed various methods of determining conceptual similarity between individual concepts in a knowledge context [Warin, 05 and Han, 00].

Determining the similarity of various structures containing many concepts is another complicated research topic. For instance, given the following three trees in

Figure 1 (which are modelling the concept structures about the university domain and are developed by different people) where relationships between concepts are identical ("*part-of*" in this example) and a list describing the similarities of individual concept pairs (e.g. sim(*People*, *Human*) = 1 and sim(*Faculty*, *Professor*) = 0.9) which can be provided by domain experts, how can we determine the extent that they are similar to each other and which two are more similar?

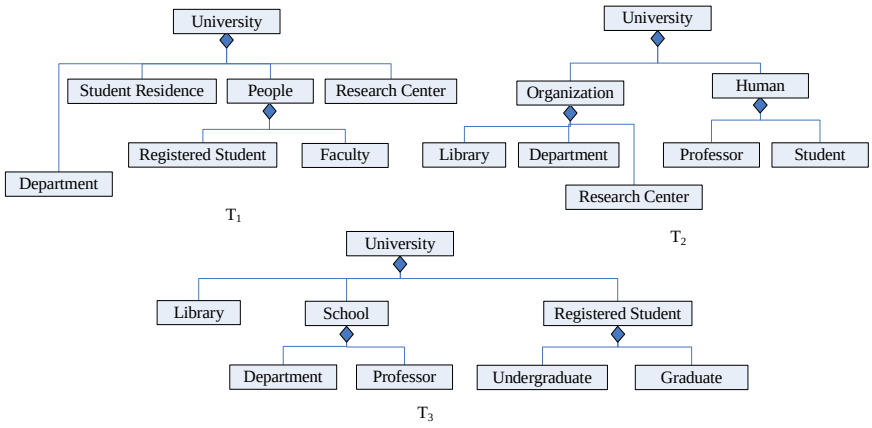

Figure 1: An example of multiple concept trees for the same domain.

Our work extends the classical tree editing operations and introduces the tree transformation operations. We propose four types of transformation operations which can transform one concept tree into another, and provide definitions for the cost of each operation considering the number of affected nodes, the scale of the node set, the conceptual significance of affected nodes, and the conceptual similarity of the node pairs (each node representing one concept) in a knowledge context. The degree of tree similarity is measured according to the tree transformation cost. This method can be applied to ontology comparison to support ontology integration in cases where different ontologies for the same domain can be represented as trees.

The rest of this paper is organized as follows. In section 2 we analyze previous work in related research, and then in section 3 we present basic definitions of our work. Section 4 and 5 discuss tree transformation operations and their costs in detail. Section 6 presents the cost computing algorithm. A case study on ontology comparison is discussed in section 7. Finally section 8 concludes the whole paper and proposes our future work.

2 Related Work

Tree is one of the most commonly used combinatorial structures in computer science. Research on comparing tree structures has a long history in many fields. It has been well studied in several diverse areas such as computational biology, structured text databases, image analysis, and compiler optimization [Bille, 03]. In these researches edit cost (or edit distance) from one tree to another one is employed to measure

similarity degree of two trees [Guegan, 06, Guda, 02, Jin 05, and Allali, 04]. However, these researches are mainly focused on finding matches based on the pure structure or geometry perspective without considering the conceptual semantics of the tree nodes in a knowledge context.

Tree patter matching is another one of the often used methods. For example, some researches explored the algorithm of matching pattern discovery in XML query [Yao, 04 and Bruno, 02] whereas they did not focus on the cost of matching. Another domain of using tree pattern matching is compiling where matching cost is defined through tree-rewriting rules and instruction types [Aho, 89].

Maedche et al conducted in-depth research of the similarity between ontologies [Maedche, 02]. In their research context, an ontology has a tree structure that is modeling a concept taxonomy. A method was developed to measure the similarity between ontologies based on the notions of lexicon, reference functions, and semantic cotopy. This method is based on an assumption that the same terms are used in different ontologies for concepts but their relative positions may vary. However, in many real ontologies different terms will be adopted to construct the concept taxonomies, although some of them have similar semantics. In these cases computing taxonomic overlap is not fully applicable and lexical level comparison becomes almost inapplicable. Furthermore, this research did not take the structural characteristics of trees into consideration.

Li et al conducted a similar research on measuring similarity of ontologies (represented as trees) based on tree structure mapping [Li, 06]. They proposed a mapping method that combines the similarity of the inner structure of concepts in different ontologies and the language similarity of concepts. The similarity of concepts is computed from some lexical databases like WordNet [Warin, 05]. However, such a generic semantic similarity calculating algorithm is not perfectly applicable in domain-based concept systems. Furthermore, Li's work did not handle cases of crossing-layer mappings, which is common in tree mapping where similar terms may be placed in various layers within the trees.

Summarizing, to the best of our knowledge, no research has been fully done to measure the similarity of trees based on both structure comparing and concept comparing and apply it to ontology comparison.

3 Definition for Concept Tree

Tree comparing has been studied in many researches. These researches are mainly focused on finding matches based on the pure structure or geometry perspective (e.g. [Guda, 02 and Jin, 05]) without considering the conceptual semantics of the tree nodes in a knowledge context.

We extend the traditional definition of trees for the sake of describing concept structures. The formal definition is given below:

Definition 1: Concept Tree. An (unordered and labelled) Concept Tree is a six-tuple $T = (V, E, L^V, \text{root}(T), D, M)$ where V is a finite set of nodes, E is a set of edges satisfying that $E \subset V \times V$ which implies an irreflexive and antisymmetric relationship between nodes, L^V is a set of lexicons (terms) for concepts used as node labels,

root(T)$\in V$ is the root of the tree, D is the domain of discourse, and M is an injective mapping from V to L^V, M: $V \rightarrow L^V$ ensuring that each node has a unique label. For convenience, we simply call each term in L^V a concept with an agreement of their semantics. A mapping from a node v to a label l is simply written as a tuple (v, l) $\in$ M.

A concept tree is acyclic and directed. If (u, v)$\in E$, we call u a parent of v and v a child of u, denoted as u = parent(v) or v = child(u). The set of all children of node u is denoted as $C(u)$. For two nodes u_1, $u_2 \in V$, if (u_1, u_2) $\in$ E^* holds, then we call u_1 an ancestor of u_2 and u_2 a descendant of u_1. The set of all descendants of node u is named $D(u)$.

The following conditions are satisfied by any concept tree:

(1) The root node does not have parent node.

(2) Any node in V other than the root has one and only one parent node.

(3) For each non-root node u in V, there exists (root(T), u)$\in E^*$, where E^* is transitive closure of E, meaning that no any node is isolated from others.

(4) There is a unique directed path composed by a sequence of elements in E from the root to each of the other elements in V.

Definition 2: Conceptual Similarity Measure. A conceptual similarity measure $S_{L^{V_1},L^{V_2}}$ is a set of mappings from two lexicon sets L^{V1}, L^{V2} used in different concept trees to the set of real numbers R, $S_{L^{V_1},L^{V_2}}$: $L^{V1} \times L^{V2} \rightarrow R$, in which each mapping denotes the conceptual similarity between two concepts represented by these two lexicons. R has a range of (0, 1]. $S_{L^{V_1},L^{V_2}}$ is semantically reflexive and symmetric, i.e. for $l_1 \in L^{V1}$ and $l_2 \in L^{V2}$ we have $S_{L^{V_1},L^{V_2}}(l_1, l_1) = 1$ and $S_{L^{V_1},L^{V_2}}(l_1, l_2) = S_{L^{V_1},L^{V_2}}(l_2, l_1)$. For convenience, we simply use $w = s(l_1, l_2)$ to refer to the number value of conceptual similarity between two concepts from two trees T_1 and T_2. Intuitively, the larger w is, the closer the two concepts are and w = 1 means two concepts are actually identical (the terms used to denote the concepts are synonymous).

Conceptual similarity between two concepts can be given by domain experts or calculated based on some linguistic analysis methods. For instance, Mitra et al use a linguistic matcher to assign a similarity score to a pair of similar concepts [Mitra, 02]. As an example, given the strings "*Department of Defense*" and "*Defense Ministry*", the match function returns match(*Defense*, *Defense*) = 1.0 and match(*Department*, *Ministry*) = 0.4, then it calculates the similarity between the two strings as: s("*Department of Defense*", "*Defense Ministry*") = (1 + 0.4)/2 = 0.7.

For $l_1 \in L^{V1}$ and $l_2 \in L^{V2}$, if there is no definition for l_1 and l_2 in the measure, we view l_1 and l_2 as totally different (disjoint) concepts. Such a concept pair will not be considered when two concept trees are being compared.

4 Tree Transformation Operations and Transformation Cost

Tree transformation operations can map one tree T into another one, T', as are defined below.

4.1 Deleting node *v* (denoted as delete(*v*))

If $v \neq$ root(T), then $V' = V - \{v\}$, $E' = E - \{(u, v) \mid u = \text{parent}(v)\} - \{(v, v_c) \mid v_c \in C(v)\} + \{(u, v_c) \mid u = \text{parent}(v) \wedge v_c \in C(v)\}$, $L^{V'} = L^V - \{M(v)\}$, and $M' = M - \{(v, M(v))\}$.

It must be noted that when deleting one node, besides eliminating that node from the tree we still need to make its children nodes new direct children nodes of its parent node, which is different from deleting a sub-tree.

If v = root(T), the result of deleting is a forest $\{T[v_c] \mid v_c \in C(v)\}$. In a concept tree the root is usually a very general concept like "object", therefore we assume that all trees have a common root concept and restrict that the root is never allowed to be deleted.

The deleting operation is depicted in the following Figure 2:

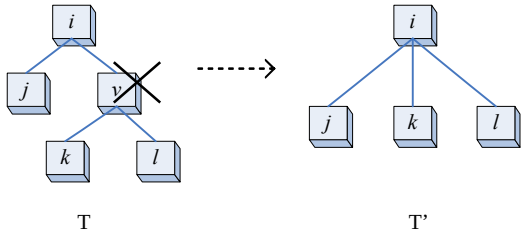

Figure 2: Deleting a node.

4.2 Inserting node *v* under node *u* (denoted as insert$_u$(*v*))

We have $V' = V + \{v\}$, $E' = E + \{(u, v)\} + \{(v, u_c) \mid u_c \in C'(u)\} - \{(u, u_c) \mid u_c \in C'(u)\}$, $L^{V'} = L^V + \{l_v\}$, and $M' = M + \{(v, l_v)\}$, where l_v is the lexicon assigned to the new node v, and $C'(u) \subseteq C(u)$ meaning that some children nodes of u are changed to be children of the new node v. The elements contained in $C'(u)$ is determined by the context when performing the editing operation.

The inserting operation is depicted in Figure 3:

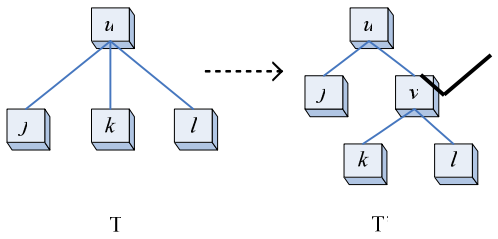

Figure 3: Inserting a node.

4.3 Re-labelling node *v* (denoted as relabel$_{lv \to lv'}$(*v*))

This is a particular operation in labelled tree. Re-labelling of v with label l_v is to assign v a new label l_v', keeping positions of all nodes unchanged. We have $L^{V'} = L^V - \{l_v\} + \{l_v'\}$ and $M' = M - \{(v, l_v)\} + \{(v, l_v')\}$, where l_v' is the new label assigned to v, as is depicted in the following Figure 4.

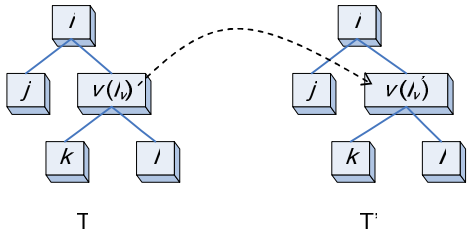

Figure 4: Re-labelling a node.

4.4 Moving node *v* to be under node *u* (denoted as move$_u$(*v*))

This is an extended operation in a knowledge context that is not defined in classical tree editing operation sets. From Figure 5 we see that in the case of pure structured trees (a) and (b) two operations delete(E) and insert$_B$(E) can be performed to convert (a) to (b). However, when mapping a concept tree to another one we cannot simply delete a node and then insert it since the concept represented by the node's label already exists in the tree.

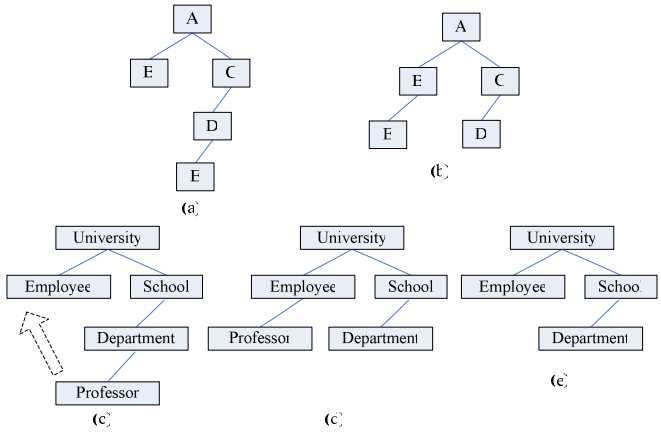

Figure 5: An example of moving operation.

More specifically, in Figure 5 two trees (c) and (d) put the concept "*Professor*" in different positions and by moving node "*Professor*" to be under "*Employee*" we transform (c) to (d), instead of deleting "*Professor*" and then inserting it back (from (c) to (e) and then (e) to (d)).

The moving operation regulates that $V' = V$, $E' = E + \{(u, v)\} + \{(v, u_c) \mid u_c \in C'(u)\} + \{(\text{parent}(v), v_c) \mid v_c \in C(v)\} - \{(\text{parent}(v), v)\} - \{(v, v_c) \mid v_c \in C(v)\} - \{(u, u_c) \mid u_c \in C'(u)\}$, where $C'(u) \subseteq C(u)$ meaning that some children of node u are changed to be children of the node v based on the operation context.

4.5 Definition for Transformation Cost

Definition 3: Transformation Cost. Each transformation operation Op on tree T is mapped to a real number which is defined as the transformation cost of the operation and denoted as $\gamma(Op)$. The transformation cost reflects the extent of change it makes to the tree.

If $OP = \{Op_1, Op_2, \ldots, Op_k\}$ is an transformation sequence, then the transformation cost of the sequence is defined as $\gamma(OP) = \sum_{i=1}^{i=|OP|} \gamma(Op_i)$.

Definition 4: Tree Transformation Cost and Similarity Index. If OP is a transformation sequence mapping a tree T_1 to another tree T_2, then the tree transformation cost from T_1 to T_2 is defined as

$\gamma(T_1 \to T_2) = \min\{\gamma(OP) \mid OP$ is a transformation sequence mapping T_1 to $T_2\}$.

Also, we define similarity index of two trees T_1 and T_2 as

$\gamma(T_1, T_2) = \min\{\gamma(T_1 \to T_2), \gamma(T_2 \to T_1)\}$.

It is a measure representing the extent to which two trees are similar to each other. The higher the tree transformation cost and similarity index is, the less similar the two trees are and vice versa.

5 Computing of Transformation Cost

In a tree transforming process we need to count the total cost of all transformation operations. A tree transforming process that maps tree $T_1 = (V_1, E_1, L^{V1}, \text{root}(T_1), D, M_1)$ into $T_2 = (V_2, E_2, L^{V2}, \text{root}(T_2), D, M_2)$ based on $S_{L^{V_1}, L^{V_2}}$ contains the following tasks:

(1) Compute the set of nodes to be deleted, D, in T_1.

$D = \{u \mid u \in V_1 \wedge M_1(u) \notin L^{V2} \wedge \neg\exists s(M_1(u), l_2) \in S_{L^{V_1}, L^{V_2}} (l_2 \in L^{V2})\}$. That is, the nodes which labels are appearing in T_1 but T_2 and have no conceptual similarity with any labels in T_2 defined (the concepts represented by the nodes in T_1 are totally not contained by T_2).

(2) Compute the set of nodes to be inserted into T_1, I.

$I = \{v \mid v \in V_2 \wedge M_2(v) \notin L^{V1} \wedge \neg\exists s(l_1, M_2(v)) \in S_{L^{V_1}, L^{V_2}} (l_1 \in L^{V1})\}$. That is, the nodes which labels are appearing in T_2 but T_1 and do not have conceptual similarity definition with any labels in T_1 (the concepts represented by the nodes in T_2 are totally not contained by T_1).

(3) Try every possible combination of the deletion and insertion operations and find the minimal cost.

(4) Compute the set of nodes to be moved within T_1 itself, M, and move them.

$M = \{u \mid u \in V_1 \wedge (M_1(u) \in L^{V2} \wedge M_1(\text{parent}(u)) \neq M_2(\text{parent}(M_2^{-1}(M_1(u)))) \wedge \neg\exists s(M_1(\text{parent}(u)), M_2(\text{parent}(M_2^{-1}(M_1(u))))) \in S_{L^{V_1},L^{V_2}}) \vee (\exists s(M_1(u), l_2) \in S_{L^{V_1},L^{V_2}} (l_2 \in L^{V2})) \wedge M_1(\text{parent}(u)) \neq M_2(\text{parent}(M_2^{-1}(l_2))) \wedge \neg\exists s(M_1(\text{parent}(u)), M_2(\text{parent}(M_2^{-1}(l_2)))) \in S_{L^{V_1},L^{V_2}})\}$. That is, the nodes that are appearing in both T_1 and T_2, or which labels have conceptual similarity with labels defined in T_2, but which parents are neither the same nor similar.

(5) After the deleting, inserting, and moving operations are performed on T_1, T_1 now has the same structure with T_2, but still has some nodes with different labels (implying different conceptual semantics). The final task is to compute the set of nodes to be re-labelled, R, and re-label them. $R = \{u \mid u \in V_1 \wedge M_1(u) \notin L^{V2} \wedge \exists s(M_1(u), l_2) \in S_{L^{V_1},L^{V_2}} (l_2 \in L^{V2})\}$. That is, the nodes that are appearing in both T_1 and T_2 with different labels, but the labels have conceptual similarity between them.

Let OP be the editing sequence containing operations in the above tasks, the transforming cost is computed as follows (using pure operation names):

$$\gamma_{T1 \to T2}(OP) = \min\{\sum_{i \in D} \gamma(delete(i)) + \sum_{i \in I} \gamma(insert(i)) + \sum_{i \in M} \gamma(move(i)) + \sum_{i \in R} \gamma(relabel(i))\}$$

The cost of each transformation operation (deleting, inserting, moving, and re-labelling) is a key issue for the measuring. The cost is affected by the level that a node resides in the tree structure, the scale of the node set, the number of descendants of a node, and the similarity of two concepts (labels) attached to two nodes. For example, first, a node at a higher layer contains richer semantics than a lower node does, or, the concept it represents is more significant for the domain than a lower one does. Therefore, when a node u is at a higher layer, the effect to the concept tree of deleting u or inserting a new node under u is larger than that of deleting or inserting a node at a lower layer. Second, the more nodes a tree has, the less the effect will be when one node is deleted or inserted. That is, the larger the concept tree is, the less different it will be if it gets one new concept or loses one old concept. Third, a node with more descendants will cause greater change to the tree structure if it is deleted, or greater change is made if a node gets more descendants after it is inserted. Finally, the more similar the two concepts are, the less the cost will be to change one into the other one.

Based on the research of [Bille, 03 and Kruskal, 99] and above observations, we define the cost for each transformation operation as follows:

- Deleting cost.

$\gamma(delete(v)) = \dfrac{height(T) - depth(v) + 1 + |D(v)|}{|V|}$, where v is a non-root node, height(T) is a function calculating the height of tree T, depth(v) calculates the depth of node v, and $|D(v)|$ is the number of descendants of node v (including its direct children and indirect offspring). Intuitively, depth(root(T)) = 1, and depth(v) > 1 iff v is not the root. If v is a leaf node, D(v) = $\varnothing$ and |D(v)| = 0. When v is a leaf node at the lowest level (height(T) = depth(v)), deleting v will cause the minimal effect to the tree and γ(delete(v)) = 1/|V|. Note that here V refers to the original node set before the deletion.

- Inserting cost.

$\gamma(insert_u(v)) = \frac{height(T) - depth(u) + 1 + |D(v)|}{|V|}$, where $|D(v)|$ is the number of descendants that v gets after it is inserted. Note that here V refers to the original node set before the insertion. When u is at the lowest layer, inserting a new node v under u will result in the minimal cost $\gamma(\text{insert}_u(v)) = 1/|V|$.

- Moving cost.

$\gamma(move_u(v)) = \frac{1}{2}[\gamma(delete(v)) + \gamma(insert_u(v))] \times \frac{|V| - 2}{|V|}$, where $|V|>2$ (the tree has a root and at least two non-root nodes) and $u \neq \text{parent}(v)$. Note that here $\text{insert}_u(v)$ is performed on a tree without node v. In this definition we consider both deleting and inserting operations because the moving operation does generate effects similar to deleting and inserting, although not exactly the same. The factor 1/2 adjusts the cost of operations since the node is not truly deleted and inserted into the tree. Another factor $(|V| - 2)/|V|$ adjusts the cost again to ensure that in an extreme case where v is the only node other than the root, its moving cost should be 0 (actually it cannot be moved) and when the number of nodes in the tree grows, the effect of the moving operation to the tree structure turns weaker.

- Re-labelling cost.

This cost is heavily dependent on the similarity of two labels (concepts). Re-labelling cost is different from deleting cost, inserting cost, or moving cost since the re-labelling operation does not result in change of a tree structure. Kouylekov et al [Kouylekov, 05] proposed a definition for substitution of two similar words w_1, w_2 as $\gamma(\text{insert}(w_2)) \times (1 - \text{sim}(w_1, w_2))$ where $\text{insert}(w_2)$ is the cost of inserting w_2 and $\text{sim}(w_1, w_2)$ is the similarity between w_1 and w_2. This definition does not take the deletion of the original word into consideration, therefore when two words have no conceptual similarity the cost of substitution becomes the cost of insertion, neglecting the implicit deleting operation. In our work we give a more comprehensive definition.

Let the conceptual similarity measure between two labels l_{v1}, l_{v2} which are attached to node v be s, $0 \leq s \leq 1$, we define:

$\gamma(relabel_{l_{v1} \to l_{v2}}(v)) = [\gamma(delete(v)) + \gamma(insert_{parent(v)}(v))] \times (1 - s)$

We analyze two extreme cases: if $s = 1$, then re-labelling will only result in literal replacing without any loss of information, therefore the re-labelling cost is 0; if $s = 0$ (i.e., the two concepts are totally different), the re-labelling operation is equivalent to deleting v and inserting v again, the transformation cost is $\gamma(\text{delete}(v)) + \gamma(\text{insert}_{\text{parent}(v)}(v))$. In other cases, the cost will be between these two boundaries.

6 Cost Computing Algorithm

The cost computing algorithm is composed by a pre-processing phase and a transforming phase, as depicted below. The pre-processing phase finds the nodes that are to be deleted and inserted. In the transforming phase, an exhaustive method is used to try every possible transformation sequence to find the minimal cost.

A. The pre-processing phase.

Input: Tree $\boldsymbol{T_1}$ and $\boldsymbol{T_2}$; Concept similarity measure set $S_{L^{V_1}, L^{V_2}}$

Output: Sets of nodes to be deleted, ***D***, and inserted, ***I***
Algorithm:

```
1)  D = ∅;
2)  for every node u in V1
3)  {
4)    if(not exists any l in L^V2 such that M1(u) = l)
5)      if(not exists any s(M1(u), l) in S_{L^V1, L^V2})
6)        add u into D;
7)  }
8)  I = ∅;
9)  for every node v in V2
10) {
11)   if(not exists any l in L^V1 such that M2(v) = l)
12)     if(not exists any s(l, M2(v)) in S_{L^V1, L^V2})
13)       add v into I;
14) }
15) return D and I;
```

B. The transforming cost computing phase.
Input: Tree T_1 and T_2; ***D, I***; Concept similarity measure set $S_{L^{V_1},L^{V_2}}$

Output: $\gamma(T_1 \rightarrow T_2)$
Algorithm:

```
1)  find all permutations composed by elements in D∪I and store in P;
2)  transformCost = +∞;
3)  for each permutation p in P
4)  {
5)    backup T1 and T2;
6)    editCost = 0;
7)    for each element u in p
8)    {
9)      perform deletion (if u∈D) or insertion (if u∈I) on u if applicable;
10)     editCost = editCost + (γ(delete(u)) or γ(insert(v)));
11)   }
12)   for each u in V1 but not in p
13)   { /* handle the nodes to be moved. */
14)     if(exists l in L^V2 such that M1(u) = l or exists any s(M1(u), l) in S_{L^V1, L^V2})
15)       if(M1 (parent(u)) ≠ M2 (parent(M2^-1(l))) and
16)         not exists any s(M1 (parent(u)), M2 (parent(M2^-1(l))) in S_{L^V1, L^V2})
17)         perform moving on u;
18)         editCost = editCost + γ(move(u));
19)   }
20)   for each u in V1 but not in p
21)   { /* handle the nodes to be re-labelled. */
```

22) if(exists l in L^{V2} such that exists any $s(M_1(u), l)$ in $S_{L^{V1},L^{V2}}$)
23) perform re-labelling on u;
24) *editCost* = *editCost* + γ(relabel(u));
25) }
26) transformCost = min(transformCost, editCost);
27) restore T_1 and T_2;
28) }
29) return transformCost;

In this algorithm a backup operation and a restore operation are included, which are used to setup a common starting point each time a new operation sequence is tried.

The same algorithm can be used to compute the cost of converting T_2 into T_1, therefore the similarity index of T_1 and T_2 can be determined.

Following we give the time complexity analysis of the algorithm: Given two trees T_1 = (V_1, E_1, L_1^{V1}, root(T_1), D, M_1), T_2 = (V_2, E_2, L_2^{V2}, root(T_2), D, M_2), and a conceptual similarity measure $S_{L^{V1},L^{V2}}$, let $|V_1|$ and $|E_1|$ be the number of nodes and edges in T_1, $|V_2|$ and $|E_2|$ be the number of nodes and edges in T_2, so the upper bound of $|S_{L^{V1},L^{V2}}|$ is $|V_1|\times|V_2|$. In the pre-processing phase, the times to search T_1, T_2 as well as $S_{L^{V1},L^{V2}}$ are:

$|V_1|\times|V_2|\times|V_2| + |V_2|\times|V_1|\times|V_1|$

Without loss of generality, we assume that two trees have similar sizes. That is, $|V_1| \approx |V_2| \approx n$. Therefore, we have $|E_1| \approx |E_2| \approx n$-1. The time complexity of the pre-processing phase is $O(n^3)$.

In the cost computing phase, by average half of the nodes in T_1 may be deleted and half of the nodes in T_2 need to be inserted, so the complexity of getting the permutations of $D \cup I$ is $O(n(n + 1)/2) = O(n^2)$. The average times of deleting and inserting nodes are n. When moving the nodes, by average $n/4$ nodes can be moved (half of the untouched nodes), and the time complexity of finding the position to move for each node is $O(n/4 + n/4) = O(n/2)$ (considering both the node itself and its parent node). The time complexity of the relabeling operations is $O(n/4)$. Therefore, the time complexity of the cost computing phase is $O(n^2)\times O(n + n/2\times n/2 + n/2) = O(n^4)$.

To sum up, the time complexity of the algorithm is $O(n^3) + O(n^4) = O(n^4)$. Usually in an ontology the number of concepts is limited and the comparison is often an one-time action, therefore the cost is acceptable although better tree comparison algorithms can be explored to reduce the cost.

7 Application on Ontology Comparison

In the situations where two trees should be compared not only based on their geometrical structures but also the concept structure implicated by their structures, our method can be applied to measure their similarity in a knowledge context. The integration of ontologies, as a type of knowledge integration [Alfirevic, 04], is among such situations.

In ontology integration, we consider the following two tasks that require ontology comparison to be crucial:

(1) Before starting the integration, find from the original candidate ontologies one that is relatively more similar to most of others (meaning that it is possibly a better one) and take it as a foundation to initialize the integration.

(2) Or, after the integration process is finished, compare the integrated result (the global ontology) obtained with the original candidate ontologies to find the best one among them that is the most similar to the integrated result, i.e., to evaluate the trust of the original candidate ontologies.

Both the above two tasks require some way to measure the similarity of different ontologies (composed by concepts and relationships, therefore bearing structural characteristic), other than just the similarity of individual concepts.

In many cases an ontology can be organized into a tree structure where each node represents one concept, semantics of the relationships between concepts are identical (e.g. "*part-of*" or "*is-a*"), and each concept is related to only one parent concept [Cho, 06]. Our method is able to help evaluate the similarities between different ontologies.

Figure 1 shows three simplified ontologies for the university domain. Given that a set of measures describing the similarities of individual concept pairs are defined:

- s(*People*, *Human*) = 1;
- s(*Registered Student*, *Student*) = 1, and
- s(*Faculty*, *Professor*) = 0.9.

One transformation sequence mapping T_1 to T_2 causes the following costs:

(1) γ(delete(*Student Residence*)) = 2/7 = 0.29;

(2) $\gamma(\text{insert}_{University}(Organization)) = 4/6 = 0.67$ (making *Department* a child node of *Organization*);

(3) $\gamma(\text{insert}_{Organization}(Library)) = 2/7 = 0.29$;

(4) $\gamma(\text{move}_{Organization}(Research\ Center)) = (1/2)((1/4) + (2/7))(6/8) = 0.20$;

(5) $\gamma(\text{relabel}_{People \to Human}(People)) = 0$;

(6) $\gamma(\text{relabel}_{Registered\ Student \to Student}(Registered\ Student)) = 0$;

(7) $\gamma(\text{relabel}_{Faculty \to Professor}(Faculty)) = 0.041$.

Finally, the entire tree transformation cost is 1.491. We have to point out that compared with the deleting, inserting, and moving costs, the re-labelling operation has a minimal effect on the tree, therefore, its cost is much smaller than the cost of the other three types of operations.

Usually there are various ways to map one tree into another one with different costs. For instance, in the university case, if both "*Department*" and "*Research Center*" are made child nodes of "*Organization*" when inserting "*Organization*", the inserting cost will be changed to 0.83, and there will be no moving cost. Consequently, the entire matching cost becomes 1.451.

The following Table 1 summarizes the transformation cost and similarity index between the three trees:

Tree Pair	Transformation Cost		Similarity Index
T_1, T_2	$\chi(T_1 \rightarrow T_2)$	$\chi(T_2 \rightarrow T_1)$	$\gamma(T_1, T_2)$
	1.227	1.034	1.034
T_1, T_3	$\chi(T_1 \rightarrow T_3)$	$\chi(T_3 \rightarrow T_1)$	$\gamma(T_1, T_3)$
	2.839	2.614	2.614
T_2, T_3	$\chi(T_2 \rightarrow T_3)$	$\chi(T_3 \rightarrow T_2)$	$\gamma(T_2, T_3)$
	2.128	2.039	2.039

Table 1: Transformation cost and similarity index.

Since T_2 is closer to both T_1 and T_3, it is better to be employed as a foundation for the integration.

On the other hand, if T_2 is the result of the integration based on T_1 and T_3, T_1 can be claimed more trustable since it is closer to the common ontology (T_2) in terms of both structure and knowledge contained in its structure.

8 Conclusion and Future Work

In this work we extend the classical tree editing operation based similarity measuring method to make it more applicable to compare trees that are representing concept structures. We propose definitions for tree transformation operations and transformation costs based on structural characteristics of the concept trees under comparison and the similarity of individual concept pairs represented by the tree nodes. We apply this method to ontology comparison where different ontologies of one domain can be represented as trees and relationships between concepts are identical. By discovering the similarity between ontologies we are able to choose the best one from a set of candidate ontologies and take it as the foundation to initialize the ontology integratio. Also the trust of these ontologies can be evaluated.

In our next step we will extend the tree structure to a graph which can model more complex concepts and relationships. New definitions for graph transformation operation and transformation cost are to be explored. Meanwhile, more types of relationships among concepts have to be considered, which requires further considerations on the semantics of the relationships.

References

[Aho, 89] Aho, A. V., Ganapathi, M., Tjiang, S. W. K.: Code Generation Using Tree Matching and Dynamic Programming, ACM Transactions on Programming Languages and Systems, Vol. 11, Issue 4, October 1989, 491-516.

[Alfirevic, 04] Alfirevic, N., Racic, D.: Knowledge Integration as a Source of Competitive Advantage in Large Croatian Enterprises, Journal of Universal Computer Science, Vol. 10, No. 6 (2004), 712-722.

[Allali, 04] Allali, J., Sagot, M.: Novel Tree Edit Operations for RNA Secondary Structure Comparison, In Proceedings of IWABI 2004, Bergen, Norway, 17-21 September, 2004, 412-425.

[Arroyo, 07] Arroyo, S.: Ontology and Grammar of the SOPHIE Choreography Conceptual Framework – An Ontological Model for Knowledge Management, Journal of Universal Computer Science, Vol. 13, No. 9 (2007), 1157-1183.

[Bille, 03] Bille, P.: Tree Edit Distance, Alignment Distance and Inclusion, Technical Report Series TR-2003-23, ISSN 1660-6100, IT University of Copenhagen, March 2003.

[Bruno, 02] Bruno, N., Srivastava, D., Koudas, N.: Holistic Twig Joins: Optimal XML Pattern Matching, In Proceedings of the 2002 ACM SIGMOD, Madison, Wisconsin, USA, June 4-6, 2002, 310-321.

[Cho, 06] Cho, M., Kim, H., Kim, P.: A New Method for Ontology Merging based on Concept using WordNet, In Proceedings of ICACT 2006, 20-22 February, 2006, Volume 3, 1573-1576.

[Guarino, 97] Guarino, N.: Understanding, Building and Using Ontologies, International Journal of Human-Computer Studies 1997, 46(2/3), 293-310.

[Guegan, 06] Guegan, M., Hernandez, N.: Recognizing Textual Parallelisms with Edit Distance and Similarity Degree, In Proceedings of EACL 2006, The Association for Computer Linguistics, Trento, Italy, April 3-7, 2006.

[Guda, 02] Guda, S., Jagadish, H. V., Koudas, N., Srivastava, D., Yu, T.: Approximate XML Joins, In Proceedings of the 2002 ACM SIGMID, Madison, Wisconsin, USA, June 4-6,2002, 287-298.

[Han, 00] Han, J., Kamber, M.: Data Mining: Concepts and Techniques, The Morgan Kaufmann Series in Data Management Systems, Jim Gray, Series Editor. Morgan Kaufmann Publishers, August 2000.

[Jin, 05] Jin, J., Sarker, B. K., Bhavsar, V. C., Boley, H., Yang, L.: Towards a Weighted-Tree Similarity Algorithm for RNA Secondary Structure Comparison, In Proceedings of HPC Asia 2005, IEEE Computer Society, Beijing, China, November 30-Decemeber 3, 2005, 639-644.

[Kouylekov, 05] Kouylekov, M., Magnini, B.: Recognizing Textual Entailment with Tree Edit Distance Algorithms, In Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment, Southampton, UK, 2005.

[Kruskal, 99] Kruskal, J. B.: An Overview of Sequence Comparison, In David Sankoff and Joseph Kruskal (Editors): Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison, Chapter One, CSLI Publications, 1999.

[Li, 06] Li, S., Hu, H., Hu, X.: An Ontology Mapping Method Based on Tree Structure, In Proceedings of the Second International Conference on Semantics, Knowledge, and Grid (SKD 2006), Guilin, Guangxi, China, 1-3 November, 2006, 87-88.

[Maedche, 02] Maedche, A., Staab, S.: Measuring Similarity between Ontologies, In Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management, Ontologies and the Semantic Web, Lecture Notes in Computer Science, Vol. 2473 (2002), 251-263.

[Mitra, 02] Mitra, P., Wiederhold, G.: Resolving Terminological Heterogeneity in Ontologies, In Proceedings of the ECAI'02 Workshop on Ontologies and Semantic Interoperability CEUR Workshop Proceedings 64: 45-50, Amsterdam, The Netherlands, 2002.

[Pinto, 99] Pinto, S., Gómez-Pérez, A., Martins, J. P.: Some Issues on Ontology Integration, In Proceedings of IJCAI 1999 Workshop on Ontologies and Problem Solving Methods (KRR 5), Stockholm, Sweden, 2 August, 1999.

[Pinto, 04] Pinto, S., Martins, P. J.: Ontologies: How Can They Be Built? Knowledge and Information Systems, 6 (4), 441-464, 2004.

[Sanin, 07] Sanin, C., Szczerbicki, E., Toro, C.: An OWL Ontology of Set of Experience Knowledge Structure, Journal of Universal Computer Science, Vol. 13, No. 2 (2007), 209-223.

[Shen, 01] Shen, W., Norrie, D. H., Barthes, J. A.: Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing, Taylor & Francis, 2001.

[Visser 97] Visser, P. R. S., Jones, D. M., Bench-Capon, T. J. M., Shave, M. J. R.: An Analysis of Ontological Mismatches: Heterogeneity versus Interoperability. In AAAI 1997 Spring Symposium on Ontological Engineering, Stanford, USA, 1997.

[Warin, 05] Warin, M., Oxhammark, H., Volk, M.: Enriching An Ontology with WordNet based on Similarity Measures, In Proceedings of the MEANING-2005 Workshop, Trento, Italy, February, 2005.

[Yao, 04] Yao, J. T., Zhang, M.: A Fast Tree Pattern Matching Algorithm for XML Query, In Proceedings of International Conference on Web Intelligence (WI 2004), 2004, 235-241.