Structure preserving semantic matching

Fausto Giunchiglia[1], Mikalai Yatskevich[1] and Fiona McNeill[2]

[1] Dept. of Information and Communication Technology, University of Trento, 38050 Povo, Trento, Italy
{fausto,mikalai.yatskevich}@dit.unitn.it
[2] School of Informatics, University of Edinburgh, EH8 9LE, Scotland
f.j.mcneill@ed.ac.uk

Abstract The most common matching applications, e.g., ontology matching, focus on the computation of the correspondences holding between the nodes of graph structures (e.g., concepts in two ontologies). However there are applications, such as matching of web service descriptions, where matching may need to compute the correspondences holding between the full graph structures and to preserve certain structural properties of the graphs being considered. The goal of this paper is to provide a new matching operator, that we call *structure preserving match*. This operator takes two graph-like structures and produces a mapping between those nodes of the structures that correspond semantically to each other, (*i*) still preserving a set of structural properties of the graphs being matched, (*ii*) only in the case that the graphs *globally* correspond semantically to each other. We present an exact and an approximate structure matching algorithm. The latter is based on a formal theory of abstraction and builds upon the well known tree edit distance measures. We have implemented the algorithms and applied them to the web service matchmaking scenario. The evaluation results, though preliminary, show the efficiency and effectiveness of our approach.

1 Introduction

We are interested in the problem of location of web services on the basis of the capabilities that they provide. This problem is often referred to as the matchmaking problem; see [12,14,15] for some examples. Most previous solutions employ a single ontology approach, that is, the web services are assumed to be described by the concepts taken from a shared ontology. This allows the reduction of the matchmaking problem to the problem of reasoning within the shared ontology. In contrast to this work, as described in [6,19], we assume that the web services are described using terms from different ontologies and that their behaviour is described using complex terms, actually first order terms. This allows us to provide detailed descriptions of their input and output behaviour. The problem becomes therefore that of matching two web service descriptions (which can be seen as graph structures) and the mapping is considered as successful only if the two graphs are *globally* similar (e.g., $tree_1$ is 0.7 similar to $tree_2$, according to some metric). A further requirement of these applications is that the mapping must preserve certain structural properties of the graphs being considered. In particular, the syntactic types and sorts have to be preserved (e.g., a function symbol must be mapped to a function symbol and a variable must be mapped to a variable). At the same time we would like to enable the matchmaking of the web service descriptions that match only approximately (see [6] for a detailed description). For instance, $get\_wine(Region, Country, Colour, Price, Number\_of\_bottles)$ can be approximately mapped to $get\_wine(Region(Country, Area), Colour, Cost, Year, Quantity)$.

In this paper, we define an operator that we call *structure preserving match*. This operator takes two graph-like structures and produces a mapping between those nodes of the structures that correspond semantically to each other, (*i*) still preserving a set of structural properties of the graphs being matched, (*ii*) only in the case that the graphs *globally* correspond semantically to each other. Notice that this problem significantly differs from the ontology matching problem, as defined for instance in [8], where (*i*) is only partially satisfied and (*ii*) is an issue which is hardly ever dealt with (see [12,23] for some noticeable exceptions). We present an exact and an approximate structure matching algorithm. The former solves the exact structure matching problem. It is designed to succeed on equivalent terms and to fail otherwise. The latter solves an approximate structure matching problem. It is based on the fusion of the ideas derived from the theory of abstraction [7] and tree edit distance algorithms [3,28]. We have implemented the algorithms and evaluated them on the dataset constructed from different versions of the state-of-the-art first order ontologies. The evaluation results, though preliminary, show the efficiency and effectiveness of our approach.

Section 2 introduces a motivating example, Section 3 discusses the exact structure preserving semantic matching. Section 4 defines the abstraction operations and introduces the correspondence between them and tree edit operations. In Section 5 we show how existing tree edit distance algorithms can be exploited for the computation of the global similarity between two web service descriptions. Section 6 is devoted to the approximate structure matching algorithm. The evaluation results are presented in Section 7. Section 8 briefly reviews the related work and concludes the paper.

2 Motivating example

Figure 1 provides an example of exactly matched web service descriptions along with their tree representations (or term trees). Dashed lines stand for the correspondences holding among the nodes of the term trees. In particular, in Figure 1 we have an exact match, namely the first of the services requires the second to return $Cars$ of a given $Brand$, $Year$ and $Colour$ while the other provides $Autos$ of a given $Brand$, $Year$ and $Colour$. Notice that there are no structural differences and that the only difference is in the function names. Where these names differ, their semantic content remains the same (*e.g.*, $Colour$ is semantically identical to $Colour$) and therefore the two descriptions constitute an exact match.

Figure 2 provides an example of an approximate match. In this case a more sophisticated data translation is required. For example, the first web service description requires the fourth argument of get_wine function ($Colour$) to be mapped to the second argument ($Colour$) of get_wine function in the second description. On the other

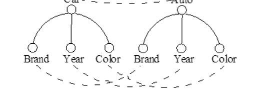

Figure 1: Exactly matched web service descriptions and their tree representations.

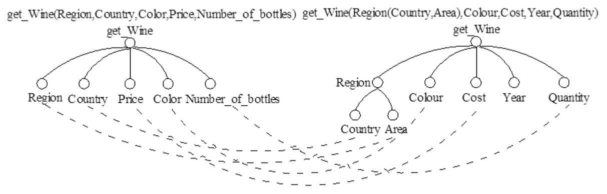
get_Wine(Region,Country,Color,Price,Number_of_bottles)
get_Wine(Region(Country,Area),Colour,Cost,Year,Quantity)
get_Wine
Region
Country
Price
Color
Number_of_bottles
get_Wine
Region
Colour
Cost
Year
Quantity
Country
Area

Figure 2: Approximately matched web service descriptions and their tree representations.

hand, *Region* on the right is defined as a function with two arguments (*Country* and *Area*) while on the left *Region* is an argument of *get_wine*. Thus, *Region* in the first web service description must be passed to the second web service as the value of the *Area* argument of the *Region* function. Moreover, *Year* on the right has no corresponding term on the left.

Therefore, in order to guarantee the successful data translation, we are interested in the correspondences holding among the nodes of the term trees of the given web service descriptions only in the case when the web service descriptions themselves are "similar enough". At the same time the correspondences have to preserve the certain structural properties of the descriptions being matched. In particular we require functions to be mapped to functions and variables to variables. We can see how the context is preserved through this mapping: for example, the two nodes *Colour* are mapped to one another, but this is done in the context that they are both children of nodes *get_wine* that are also mapped to one another. Thus we can tell that *Colour* is likely to mean the same thing in both cases.

3 Exact structure semantic matching

There are two stages in the matching process:

- *Node matching*: solves the semantic heterogeneity problem by considering only labels at nodes and domain specific contextual information of the trees. In our approach we use semantic matching, as extensively described in [8]. Notice that the result of this stage is the set of correspondences holding between the nodes of the trees.
- *Structural tree matching*: exploits the results of the node matching and the structure of the tree to find the correspondences holding between the trees themselves (e.g., $tree_1$ is 0.7 similar to $tree_2$).

The exact structure matching algorithm exploits the results of the node matching algorithm. It is designed to succeed for equivalent terms and to fail otherwise. It expects the trees to have the same depth and the same number of children. More precisely we say that two trees T_1 and T_2 match iff for any node n_{1i} (numbers in subscript refer to the tree and the node in this tree, respectively) in T_1 there is a node n_{2j} in T_2 such that:

- n_{1i} semantically matches n_{2j};
- n_{1i} and n_{2j} reside on the same depth in T_1 and T_2, respectively;
- all ancestors of n_{1i} are semantically matched to the ancestors of n_{2j}.

We do not discuss the exact structure preserving matching any further, since its implementation is straightforward, see [6] for details.

4 Approximate matching via abstraction/refinement operations

In [7], Giunchiglia and Walsh describe their theory of abstraction. We present here the key concepts in order to facilitate the presentation of our approach, which builds upon this work. Giunchiglia and Walsh categorise the various kinds of abstraction operations in a wide-ranging survey. They also introduce a new class of abstractions, called TI-abstractions (where TI means "Theorem Increasing"), which have the fundamental property of maintaining completeness, while losing correctness. In other words, any fact that is true of the original term is also true of the abstract term, but not vice versa. Similarly, if a ground formula is true, so is the abstract formula, but not vice versa. Dually, by taking the inverse of each abstraction operation, we can define a corresponding refinement operation which preserves correctness while losing completeness. The second fundamental property of the abstraction operations is that they provide all and only the possible ways in which two first order terms can be made to differ by manipulations of their signature, still preserving completeness. In other words, this set of abstraction/refinement operations defines all and only the possible ways in which correctness and completeness are maintained when operating on first order terms and atomic formulas. This is the fundamental property which allows us to study and consequently quantify the semantic similarity (distance) between two first order terms. To this extent it is sufficient to determine which abstraction/refinement operations are necessary to convert one term into the other and to assign to each of them a cost that models the "semantic distance" associated to the operation.

Giunchiglia and Walsh's categories are as follows:

Predicate: Two or more predicates are merged, typically to the least general generalisation in the predicate type hierarchy, e.g.,

– $Bottle(X) + Container(X) \mapsto Container(X)$.

We call $Container(X)$ a predicate abstraction of $Bottle(X)$ or $Container(X) \sqsupseteq_{Pd} Bottle(X)$. Conversely we call $Bottle(X)$ a predicate refinement of $Container(X)$ or $Bottle(X) \sqsubseteq_{Pd} Container(X)$.

Domain: Two or more terms are merged, typically by moving the functions (or constants) to the least general generalisation in the domain type hierarchy, e.g.,

– $Daughter(Me) + Child(Me) \mapsto Child(Me)$.

Similarly to the previous item we call $Child(Me)$ a domain abstractions of $Daughter(Me)$ or $Child(Me) \sqsupseteq_D Daughter(Me)$. Conversely we call $Daughter(Me)$ a domain refinements of $Child(Me)$ or $Daughter(Me) \sqsubseteq_D Child(Me)$.

Propositional: One or more arguments are dropped, e.g.,

– $Bottle(A) \mapsto Bottle$.

We call $Bottle$ a propositional abstraction of $Bottle(A)$ or $Bottle \sqsupseteq_P Bottle(A)$. Conversely $Bottle(A)$ is a propositional refinement of $Bottle$ or $Bottle(A) \sqsubseteq_P Bottle$.

Precondition: The precondition of a rule is dropped [1] , e.g.,

– $[Ticket(X) \rightarrow Travel(X)] \mapsto Travel(X)$.

Consider the pair of first order terms ($Bottle\ A$) and ($Container$). In this case there is no abstraction/refinement operation that make them equivalent. However consequent applications of propositional and predicate abstraction operations make the two terms

[1] We do not consider precondition abstraction and refinement in the rest of this paper as we do not want to drop preconditions, because this would endanger the successful matchmaking of web services.

equivalent:

$$(Bottle\ A) \mapsto^{\sqsubseteq_P} (Bottle) \mapsto^{\sqsupseteq_{Pd}} (Container) \qquad (1)$$

In fact the relation holding among the terms is a composition of two refinement operations, namely $(Bottle\ A) \sqsubseteq_P (Bottle)$ and $(Bottle) \sqsubseteq_{Pd} (Container)$. We define an *abstraction mapping element (AME)* as a 5-tuple $\langle ID_{ij}, t_1, t_2, R, sim \rangle$, where ID_{ij} is a unique identifier of the given mapping element; t_1 and t_2 are first order terms; R specifies a relation for the given terms; and *sim* stands for a similarity coefficient in the range [0..1] quantifying the strength of the relation. In particular for the AMEs we allow the semantic relations $\{\equiv, \sqsupseteq, \sqsubseteq\}$, where $\equiv$ stands for equivalence, $\sqsupseteq$ represents an abstraction relation and connects the precondition and the result of a composition of arbitrary numbers of predicate, domain and propositional abstraction operations, and $\sqsubseteq$ represents a refinement relation and connects the precondition and the result of a composition of arbitrary numbers of predicate, domain and propositional refinement operations.

Therefore, the problem of AME computation becomes a problem of minimal cost composition of the abstraction/refinement operations allowed for the given relation R that are necessary to convert one term into the other. In order to solve this problem we propose to represent abstraction/refinement operations as tree edit distance operations applied to the term trees. Calculating the cost of moving between nodes therefore becomes the problem of determining whether these nodes are equivalent, an abstraction or refinement of one another, or none of these relations. Note that this calculation does not in general require specific background knowledge; the semantic matching techniques allow us to calculate this automatically. Naturally, the semantic matching techniques themselves require some kind of background knowledge but this is not specific: currently, our semantic matching techniques use WordNet; see [8] for more details. This allows us to redefine the problem of AME computation into a tree edit distance problem.

In its traditional formulation, the tree edit distance problem considers three operations: (i) vertex deletion, (ii) vertex insertion, and (iii) vertex replacement [25]. Often these operations are presented as rewriting rules:

$$(i)\ \ \upsilon \rightarrow \lambda \qquad (ii)\ \ \lambda \rightarrow \upsilon \qquad (iii)\ \ \upsilon \rightarrow \omega \qquad (2)$$

where υ and ω correspond to the labels of nodes in the trees while λ stands for the special blank symbol. Figure 3 illustrates two applications of delete and replace tree edit operations.

Our proposal is to restrict the formulation of the tree edit distance problem in order to reflect the semantics of the first order terms. In particular we propose to redefine the tree edit distance operations in a way that will allow them to have one-to-one correspondence to the abstraction/refinement operations presented previously in this section. Table 1 illustrates the correspondence between abstraction/refinement and tree edit operations. The first column presents the abstraction/refinement operations. The second column lists corresponding tree edit operations. The third column describes the preconditions of the tree edit operation use. Consider, for example, the first line of Table 1. The predicate abstraction operation applied to first order term t_1 results with term t_2 $(t_1 \sqsupseteq_{Pd} t_2)$. This abstraction operation corresponds to a tree edit replacement operation applied to the term tree of t_1 that replaces the node a with the node $b(a \rightarrow b)$. Moreover the operation can be applied only in the case that (i) label a is a generalisation of label b and (ii) both nodes a and b in the term trees correspond to predicates in the first order terms.

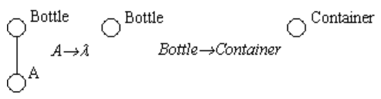

Figure 3: Delete and replace tree edit operations

Table 1: The correspondence between abstraction/refinement operations and tree edit operations.

Abstraction/refinement operation	Tree edit operation	Preconditions of operation
$t_1 \sqsupseteq_{Pd} t_2$	$a \rightarrow b$	$a \sqsupseteq b$; a and b correspond to predicates
$t_1 \sqsupseteq_{D} t_2$	$a \rightarrow b$	$a \sqsupseteq b$; a and b correspond to functions or constants
$t_1 \sqsupseteq_{P} t_2$	$a \rightarrow \lambda$	a corresponds to predicates, functions or constants
$t_1 \sqsubseteq_{Pd} t_2$	$a \rightarrow b$	$a \sqsubseteq b$; a and b correspond to predicates
$t_1 \sqsubseteq_{D} t_2$	$a \rightarrow b$	$a \sqsubseteq b$; a and b correspond to functions or constants
$t_1 \sqsubseteq_{P} t_2$	$a \rightarrow \lambda$	a corresponds to predicates, functions or constants

5 Computing the global similarity between two trees

Our goal now is to compute the similarity between two term trees. In order to perform this we need to compute the minimal cost composition of the abstraction/refinement operations that are necessary to convert one term tree (or first order term) into the other. The starting point is the traditional formulation of the tree edit distance problem.

$$Cost = \sum_{i \in S} n_i * Cost_i \tag{3}$$

The similarity between two trees is thus the minimal possible *Cost* as defined in Eq. 3; that is, the set of operations that transforms one tree into another at minimal cost. In Eq. 3, S stands for the set of the allowed tree edit operations; n_i stands for the number of *i-th* operations necessary to convert one tree into the other and $Cost_i$ defines the cost of the *i-th* operation. Our goal is to define the $Cost_i$ in a way that models the semantic distance between the two trees.

A possible uniform proposal is to assign the same unit cost to all tree edit operations that have counterparts in the theory of abstraction. These are defined in Table 1. Table 2 illustrates the costs of the abstraction/refinement (tree edit) operations, depending on the relation (equivalence, abstraction or refinement) being computed. These costs have to be adjusted depending on what relation is being considered: for example, the cost of applying an abstraction operation is different if we are considering abstraction relations than if we are considering refinement relations. In particular, the tree edit operations corresponding to abstraction/refinement operations that are not allowed by the definition of the given relation have to be prohibited by assigning to them an infinite cost. Notice also that we do not give any preference to a particular type of abstraction/refinement operations. Of course this strategy can be changed to satisfy certain domain specific requirements.

Consider, for example, the first line in Table 2. The cost of the tree edit distance operation that correspond to the propositional abstraction ($t_1 \sqsupseteq_{Pd} t_2$) is equal to 1 when used for the computation of equivalence ($Cost_{\equiv}$) and abstraction ($Cost_{\sqsupseteq}$) relations in AME. It is equal to ∞ when used for the computation of refinement ($Cost_{\sqsubseteq}$) relation.

Table 2: Costs of the abstraction/refinement (tree edit) operations, exploited for computation of equivalence ($Cost_{\equiv}$), abstraction ($Cost_{\sqsubseteq}$) and refinement ($Cost_{\sqsupseteq}$) relations holding among the terms.

Abstraction/refinement (tree edit) operation	$Cost_{\equiv}$	$Cost_{\sqsubseteq}$	$Cost_{\sqsupseteq}$
$t_1 \sqsupseteq_{Pd} t_2$	1	∞	1
$t_1 \sqsupseteq_{D} t_2$	1	∞	1
$t_1 \sqsupseteq_{P} t_2$	1	∞	1
$t_1 \sqsubseteq_{Pd} t_2$	1	1	∞
$t_1 \sqsubseteq_{D} t_2$	1	1	∞
$t_1 \sqsubseteq_{P} t_2$	1	1	∞

Eq. 3 can now be used for the computation of the tree edit distance score. However, when comparing two web service descriptions we are interested in similarity rather than in distance. We exploit the following equation to convert the distance produced by an edit distance algorithm into the similarity score:

$$sim = 1 - \frac{Cost}{max(number\_of\_nodes_1, number\_of\_nodes_2)} \tag{4}$$

where $number\_of\_nodes_1$ and $number\_of\_nodes_2$ stand for the number of nodes in the trees. Note that for the special case of $Cost$ equal to ∞ the similarity score is estimated as 0.

Many existing tree edit distance algorithms allow us to keep track of the nodes to which a replace operation is applied. Therefore, as a result they allow us to obtain not only the minimal tree edit cost but also a minimal cost mapping among the nodes of the trees. According to [25], this minimal cost mapping is (*i*) one-to-one; (*ii*) horizontal-order preserving between sibling nodes; and (*iii*) vertical-order preserving. These criteria are not always preserved in our approach. For example, the mapping depicted in Figure 1 complies to all these requirements while the mapping depicted in Figure 2 violates (*ii*). In particular the third sibling *Price* on the left tree is mapped to the third sibling *Cost* on the right tree while the fourth sibling *Colour* on the right tree is mapped to the second sibling *Colour* on the left tree.

For the tree edit distance operations depicted in Table 1, we propose to keep track of nodes to which the tree edit operations derived from the replace operation are applied. In particular we consider the operations that correspond to predicate and domain abstraction/refinement ($t_1 \sqsupseteq_{Pd}$, $t_1 \sqsubseteq_{Pd}$, $t_1 \sqsupseteq_{D}$, $t_1 \sqsubseteq_{D}$). This allows us to obtain a mapping among the nodes of the term trees with the desired properties (i.e., there is only one-to-one correspondences in the mapping). Moreover it complies to the structure preserving matching requirements that functions are mapped to functions and variables are mapped to variables. This is the case because (*i*) predicate and domain abstraction/ refinement operations do not convert, for example, a function into a variable and (*ii*) the tree edit distance operations, as from Table 1, have a one-to-one correspondence with abstraction/refinement operations.

At the same time, a mapping returned by a tree edit distance algorithm preserves the horizontal order among the sibling nodes, but this is not desirable property for the data translation purposes. This is the case because the correspondences that do not comply to the horizontal order preservation requirements, like the one holding between *Colour* and *Colour* on Figure 2, are not included in the mapping. However, as from Table 1, the tree edit operations corresponding to predicate and domain abstraction/refinement ($t_1 \sqsupseteq_{Pd}$, $t_1 \sqsubseteq_{Pd}$, $t_1 \sqsupseteq_{D}$, $t_1 \sqsubseteq_{D}$) can be applied only to those nodes of the trees whose labels are either generalisations or specialisations of each other, as computed by the node matching algorithm. Therefore, given the mapping produced by the node matching algorithm we can always recognise the cases when the horizontal order between sibling nodes is not preserved and change the ordering of the sibling nodes to make the mapping horizontal order preserving. For example, swapping the nodes *Cost* and *Colour* in the right tree depicted on Figure 2 does not change the meaning of the corresponding term but allows the correspondence holding between *Colour* and *Colour* on Figure 2 to be included in the mapping produced by a tree edit distance algorithm.

We can see that this technique satisfies the two properties mentioned earlier: namely, that the operator finds a mapping (*i*) still preserving a set of structural properties of the graphs being matched, (*ii*) only in the case that the graphs *globally* correspond semantically to each other. If the graphs do not correspond semantically to one another, and the structural properties of the graphs do not match, the similarity score will be very low.

6 The approximate structure matching algorithm

As discussed above, our goal is to find good enough services [9] if perfect services are not available. We start by providing a definition of the approximate structure matching as the basis for the algorithm.

We say that two nodes n_1 and n_2 in trees T_1 and T_2 approximately match iff $c@n_1$ R $c@n_2$ holds given the available background knowledge, where $c@n_1$ and $c@n_2$ are the concepts at nodes of n_1 and n_2, and where $R \in \{\equiv, \sqsubseteq, \sqsupseteq\}$. We say that two trees T_1 and T_2 match iff there is at least one node n_{1i} in T_1 and a node n_{2j} in T_2 such that: (*i*) n_{1i} approximately matches n_{2j} and (*ii*) all ancestors of n_{1i} are approximately matched to the ancestors of n_{2j}.

First the approximate structure matching algorithm estimates the similarity of two terms by application of a tree edit distance algorithm with the tree edit operations and costs modified as described in Sections 4 and 5. The similarity scores are computed for equivalence, abstraction and refinement relations. For each of these cases the tree edit distance operation costs are modified as depicted in Table 2. The relation with the highest similarity score is assumed to hold among the terms. If the similarity score exceeds a given threshold, the mappings connecting the nodes of the term trees, as computed by the tree edit distance algorithm, are returned by the matching routine what allows for further data translation. Algorithm 1 provides pseudo code for the approximate structure matching algorithm.

approximateStructureMatch takes as input the *source* and *target* term trees and a *threshold* value. approximateTreeMatch fills the *result* array (line 3) which stores the mappings holding between the nodes of the trees. An AME *ame* is computed (line 4) by analyzeMismatches. If *ame* stands for equivalence, abstraction or refinement relations (line 5) and if an *approximationScore* exceeds *threshold* (line 6) the mappings calculated by approximateTreeMatch are returned (line 7). analyzeMismatches calculates the aggregate score of tree match quality by exploiting a tree edit distance algorithm as described in Section 5.

Algorithm 1 Pseudo code for approximate structure matching algorithm

```
AME struct of
    Tree of Nodes source;
    Tree of Nodes target;
    String relation;
    double approximationScore;

1.MappingElement[] approximateStructureMatch(Tree of Nodes source, target, double threshold)
2. MappingElement[] result;
3. approximateTreeMatch(source,target,result);
4. AME ame=analyzeMismatches(source,target,result);
5. if (getRelation(ame)=="=") or (getRelation(ame)=="<") or (getRelation(ame)==">")
6.     if (getApproximationScore(ame)>threshold)
7.     return result;
8. return null;
```

7 Evaluation

We have implemented the algorithm described in the previous section. In the implementation we have exploited a modification of simple tree edit distance algorithm from Valiente's work [27]. We have evaluated the matching quality of the algorithms on 132 pairs of first order logic terms. Half of the pairs were composed of the equivalent terms (e.g., *journal(periodical-publication)* and *magazine (periodical-publication)*) while the other half were composed from similar but not equivalent terms (e.g., *web-reference(publication-reference)* and *thesis-reference (publication-reference)*). Te terms were extracted from different versions of the Standard Upper Merged Ontology (SUMO)[2] and the Advanced Knowledge Technology (AKT)[3] ontologies. We extracted all the differences between versions 1.50 and 1.51, and between versions 1.51 and 1.52 of the SUMO ontology and between versions 1, 2.1 and 2.2 of the AKT-portal and AKT-support ontologies[4]. These are both first-order ontologies, so many of these differences mapped well to the potential differences between terms that we are investigating. However, some of them were more complex, such as differences in inference rules, or consisted of ontological objects being added or removed rather than altered, and had no parallel in our work. These pairs of terms were discarded and our tests were run on all remaining differences between these ontologies. Therefore, we have simulated the situation when the service descriptions are defined exploiting the two versions of the same ontology.

In our evaluation we have exploited the commonly accepted measures of matching quality, namely precision, recall, and F-measure. Precision varies in the [0..1] range; the higher the value, the smaller the set of incorrect correspondences (false positives) which have been computed. Precision is a correctness measure. Recall varies in the [0..1] range; the higher the value, the smaller the set of correct correspondences (true positives) which have not found. Recall is a completeness measure. F-measure varies in the [0..1] range. The version computed here is the harmonic mean of precision and recall. It is a global measure of the matching quality, increasing as the matching quality improves. While computing precision and recall we have considered the correspondences holding among first order terms rather than the nodes of the term trees. Thus, for instance, *journal(periodical-publication$_1$)=magazine(periodical-publication$_2$)* was con-

[2] http://ontology.teknowledge.com/

[3] http://www.aktors.org

[4] See http://dream.inf.ed.ac.uk/projects/dor/ for full versions of these ontologies and analysis of their differences.

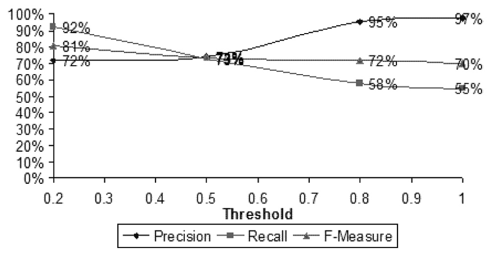

Figure 4: The matching quality measures depending on threshold value for approximate structure matching algorithm.

sidered as single correspondence rather than two correspondences, namely *journal*= *magazine* and *periodical-publication*$_1$=*periodical-publication*$_2$.

Interestingly enough, our exact structure matching algorithm was able to find 36 correct correspondences what stands for 54% of Recall with 100% Precision. All mismatches (or correct correspondences not found by the algorithm) corresponded to structural differences among first order terms which exact structure matching algorithm is unable to capture. The examples of correctly found correspondences are given below:

```
meeting-attendees(has-other-agents-involved) : meeting-attendee(has-other-agents-involved)

r&d-institute(Learning-centred-organization) : r-and-d-institute(Learning-centred-organization)

piece(Pure2,Mixture) : part(Pure2,Mixture)

has-affiliatied-people(Affiliated-person) : has-affililated-person(affiliated-person)
```

The first and second examples illustrate the minor syntactic differences among the terms, while the third and fourth examples illustrate the semantic heterogeneity in the various versions of the ontologies.

Figure 4 presents the matching quality measures depending on the cut-off threshold value for approximate structure preserving matching algorithm. As illustrated in Figure 4, the algorithm demonstrates high matching quality on the wide range of threshold values. In particular, F-Measure values exceed 70% for the given range. Table 3 summarizes the time performance of the matching algorithm. It presents the average time taken by the various steps of the algorithm on 132 term matching tasks. As illustrated in Table 3, Step 1 and 2 of the node matching algorithm significantly slow down the whole process. However these steps correspond to the linguistic preprocessing that can be performed once offline [8]. Given that the terms can be automatically annotated with the linguistic preprocessing results [8] once when changed, the overall runtime is reduced to 4.2 ms, which corresponds roughly to 240 term matching tasks per second.

Table 3: Time performance of approximate structure matching algorithm (average on 132 term matching tasks).

	Node matching: steps 1 and 2 [8]	Node matching: steps 3 and 4 [8]	Structure matching
Time, ms	134.1	3.3	0.9

8 Conclusions and Related work

We have presented an approximate structure matching algorithm that implements the *structure preserving match* operator. We have implemented the algorithm and applied it to the web service matchmaking scenario. The evaluation results, though preliminary, show the efficiency and effectiveness of our approach.

Future work includes further investigations on the cost assignment for the abstraction/refinement operations. In the version of the algorithm presented in the paper, no preference is given to the particular abstraction/refinement operation and all allowed operations are assigned a unit cost. One may argue, for example, that the semantic distance between *cat* and *mammal* is less then the semantic distance between *cat* and *animal*. Therefore, the operation abstracting *cat* to *mammal* should be less costly than the operation abstracting *cat* to *animal*.

The problem of location of web services on the basis of the capabilities that they provide (often referred as the matchmaking problem) has recently received a considerable attention. Most of the approaches to the matchmaking problem so far employed a single ontology approach (i.e., the web services are assumed to be described by the concepts taken from the shared ontology). See [14,15,21] for example. Probably the most similar to ours is the approach taken in METEOR-S [1] and in [20], where the services are assumed to be annotated with the concepts taken from various ontologies. Then the matchmaking problem is solved by the application of the matching algorithm. The algorithm combines the results of atomic matchers that roughly correspond to the element level matchers exploited as part of our algorithm. In contrast to this work, we exploit a more sophisticated matching technique that allows us to utilise the context provided by the first order term.

Many diverse solutions to the ontology matching problem have been proposed so far. See [23] for a comprehensive survey and [5,18,4,10,2,12,24] for individual solutions. However most efforts has been devoted to computation of the correspondences holding among the classes of description logic ontologies. Recently, several approaches allowed computation of correspondences holding among the object properties (or binary predicates) [26]. The approach taken in [11] facilitates the finding of correspondences holding among parts of description logic ontologies or subgraphs extracted from the ontology graphs. In contrast to these approaches, we allow the computation of correspondences holding among first order terms.

9 Acknowledgements

This work has been supported by the STReP OpenKnowledge (http://www.openk.org/). The authors are indebted to Pavel Shvaiko for his extensive feedback on earlier versions of this paper.

References

1. R. Aggarwal, K. Verma, J. A. Miller, and W. Milnor. Constraint driven web service composition in METEOR-S. In *Proceedings of IEEE SCC*, pages 23–30, 2004.
2. S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1):54–59, 1999.
3. W. Chen. New algorithm for ordered tree-to-tree correction problem. *Journal of Algorithms*, 40(2):135–158, 2001.
4. M. Ehrig, S. Staab, and Y. Sure. Bootstrapping ontology alignment methods with APFEL. In *Proceedings of ISWC*, pages 186–200, 2005.
5. J. Euzenat and P. Valtchev. Similarity-based ontology alignment in OWL-lite. In *Proceedings of ECAI*, pages 333–337, 2004.

6. F. Giunchiglia, F. McNeill, and M. Yatskevich. Web service composition via semantic matching of interaction specifications. Technical Report DIT-06-080, University of Trento, 2006.
7. F. Giunchiglia and T. Walsh. A theory of abstraction. *Artificial Intelligence*, 57(2-3):323–389, 1992.
8. F. Giunchiglia, M. Yatskevich, and P. Shvaiko. Semantic matching: Algorithms and implementation. *Journal on Data Semantics*, IX:1–38, 2007.
9. F. Giunchiglia and I. Zaihrayeu. Making peer databases interact – a vision for an architecture supporting data coordination. In *Proceedings of the CIA workshop*, pages 18–35, 2002.
10. R. Gligorov, Z. Aleksovski, W. ten Kate, and F. van Harmelen. Using google distance to weight approximate ontology matches. In *Proceedings of WWW*, 2007.
11. W. Hu and Y. Qu. Block matching for ontologies. In *Proceedings of ISWC*, pages 300–313, 2006.
12. Y. Kalfoglou and M. Schorlemmer. IF-Map: an ontology mapping method based on information flow theory. *Journal on Data Semantics*, I:98–127, 2003.
13. J. Kim, M. Jang, Y.-G. Ha, J.-C. Sohn, and S.-J. Lee. MoA: OWL ontology merging and alignment tool for the semantic web. In *Proceedings of IEA/AIE*, pages 722–731, 2005.
14. M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with OWLS-MX. In *Proceedings of AAMAS*, pages 915–922, 2006.
15. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of WWW*, pages 331–339, 2003.
16. V. Lopez, M. Sabou, and E. Motta. Powermap: Mapping the real semantic web on the fly. In *Proceedings of ISWC*, pages 414–427, 2006.
17. P. Mitra, N. Noy, and A. Jaiswal. Omen: A probabilistic ontology mapping tool. In *Proceedings of ISWC*, pages 537–547, 2005.
18. N. Noy and M. Musen. The PROMPT suite: interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59(6):983–1024, 2003.
19. OpenKnowledge Manifesto. http://www.openk.org/, 2006.
20. S. Oundhakar, K. Verma, K. Sivashanugam, A. Sheth, and J. Miller. Discovery of web services in a multi-ontology and federated registry environment. *International Journal of Web Services Research*, 2(3):1–32, 2005.
21. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of ISWC*, pages 333–347, 2002.
22. Y. Qu, W. Hu, and G. Cheng. Constructing virtual documents for ontology matching. In *Proceedings of WWW*, pages 23–31, 2006.
23. P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics*, IV:146–171, 2005.
24. U. Straccia and R. Troncy. oMAP: Combining classifiers for aligning automatically OWL ontologies. In *Proceedings of WISE*, pages 133–147, 2005.
25. K.-C. Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3):422–433, 1979.
26. J. Tang, J. Li, B. Liang, X. Huang, Y. Li, and K. Wang. Using Bayesian decision for ontology mapping. *Journal of Web Semantics*, 4(1):243–262, 2006.
27. G. Valiente. *Algorithms on Trees and Graphs*. Springer, 2002.
28. J. T.-L. Wang, B. Shapiro, D. Shasha, K. Zhang, and K. Currey. An algorithm for finding the largest approximately common substructures of two trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):889–895, 1998.