ECAI 2008
M. Ghallab et al. (Eds.)
IOS Press, 2008

doi:10.3233/978-1-58603-891-5-743

Approximate structure preserving semantic matching

Fausto Giunchiglia[1], Mikalai Yatskevich [1], Fiona McNeill [2]
Pavel Shvaiko[1], Juan Pane[1], Paolo Besana[2]

Abstract. Typical ontology matching applications, such as ontology integration, focus on the computation of correspondences holding between the nodes of two graph-like structures, e.g., between concepts in two ontologies. However, there are applications, such as web service integration, where we may need to establish whether full graph structures correspond to one another globally, preserving certain structural properties of the graphs being considered. The goal of this paper is to provide a new matching operation, called *structure preserving matching*. This operation takes two graph-like structures and produces a set of correspondences between those nodes of the graphs that correspond semantically to one another, (i) still preserving a set of structural properties of the graphs being matched, (ii) only in the case if the graphs are *globally* similar to one another. We present a novel approximate structure preserving matching approach that implements this operation. It is based on a formal theory of abstraction and on a tree edit distance measure. We have evaluated our solution with encouraging results.

1 INTRODUCTION

Many varied solutions of matching have been proposed so far [1][3]. In this paper we focus on a particular type of matching, namely *structure preserving matching*. Similarly to the conventional ontology matching, structure preserving matching finds correspondences between semantically related nodes of the graphs. Differently from it, it preserves a set of structural properties (e.g., vertical ordering of nodes) and establishes whether two graphs are globally similar. These characteristics of matching are required in web service integration applications, see, e.g., [5].

Let us consider an example of approximate structure preserving matching between two web services: get_wine(Region, Country, Color, Price, Number_of_bottles) and get_wine(Region(Country, Area), Colour, Cost, Year, Quantity), see Figure 1. In this case the first web service description requires the fourth argument of the get_wine function (Color) to be matched to the second argument (Colour) of the get_wine function in the second description. Also, Region in $T2$ is defined as a function with two arguments (Country and Area), while in $T1$, Region is an argument of get_wine. Thus, Region in $T1$ must be passed to $T2$ as the value of the Area argument of the Region function. Moreover, Year in $T2$ has no corresponding term in $T1$. Notice that detecting these correspondences would have not been possible in the case of exact matching by its definition.

In order to guarantee a successful web service integration, we are only interested in the correspondences holding among the nodes of the trees underlying the given web services in the case when the web

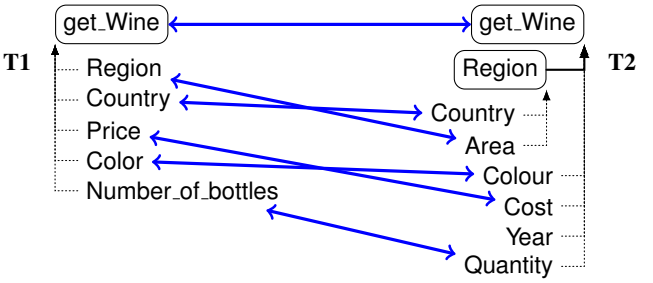

Figure 1: Two approximately matched web services represented as trees. Functions are in rectangles with rounded corners; they are connected to their arguments by dashed lines. Node correspondences are indicated by arrows.

services themselves are similar enough. At the same time the correspondences have to preserve two structural properties of the descriptions being matched: (i) functions have to be matched to functions and (ii) variables to variables. Thus, for example, Region in $T1$ is not linked to Region in $T2$. Finally, let us suppose that the correspondences on the example of Figure 1 are aggregated into a single similarity measure between the trees under consideration, e.g., 0.62. If this global similarity measure is higher than empirically established threshold (e.g., 0.5), the web services under scrutiny are considered to be similar enough, and the set of correspondences showed in Figure 1 is further used for the actual web service integration.

2 THE APPROACH

The matching process is organized in two steps: (i) node matching and (ii) tree matching. Node matching solves the semantic heterogeneity problem by considering only labels at nodes and contextual information of the trees. We use here the S-Match system [4]. Technically, two nodes $n_1 \in T1$ and $n_2 \in T2$ match iff: $c@n_1\ R\ c@n_2$ holds, where $c@n_1$ and $c@n_2$ are the concepts at nodes n_1 and n_2, and $R \in \{=, \sqsubseteq, \sqsupseteq\}$. In semantic matching [2] as implemented in the S-Match system [4] the key idea is that the relations, e.g., equivalence and subsumption, between nodes are determined by (i) expressing the entities of the ontologies as logical formulas and by (ii) reducing the matching problem to a logical validity problem. Specifically, the entities are translated into logical formulas which explicitly express the concept descriptions as encoded in the ontology structure and in external resources, such as WordNet. This allows for a translation of the matching problem into a logical validity problem, which can then be efficiently resolved using sound and complete state of the art satisfiability solvers. Notice that the result of this stage is the set of one-to-many correspondences holding between the nodes of the trees. For example, initially Region in $T1$ is matched to both Region and Area in $T2$.

Tree matching exploits the results of the node matching and the structure of the trees to find if these globally match each other as

[1] University of Trento, Italy, email:{fausto,yatskevi,pavel,pane}@dit.unitn.it
[2] University of Edinburgh, Scotland, email:{f.j.mcneill,p.besana}@ed.ac.uk
[3] See, http://www.ontologymatching.org for a complete information on the topic.

Table 1: The correspondence between abstraction operations, tree edit operations and costs.

Abstraction operations	Tree edit operations	Preconditions of operations	$Cost_{T1=T2}$	$Cost_{T1\sqsubseteq T2}$	$Cost_{T1\sqsupseteq T2}$
$t_1 \sqsupseteq_{Pd} t_2$	$a \rightarrow b$	$a \sqsupseteq b$; a and b correspond to predicates	1	∞	1
$t_1 \sqsupseteq_{D} t_2$	$a \rightarrow b$	$a \sqsupseteq b$; a and b correspond to functions or constants	1	∞	1
$t_1 \sqsupseteq_{P} t_2$	$a \rightarrow \lambda$	a corresponds to predicates, functions or constants	1	∞	1
$t_1 \sqsubseteq_{Pd} t_2$	$a \rightarrow b$	$a \sqsubseteq b$; a and b correspond to predicates	1	1	∞
$t_1 \sqsubseteq_{D} t_2$	$a \rightarrow b$	$a \sqsubseteq b$; a and b correspond to functions or constants	1	1	∞
$t_1 \sqsubseteq_{P} t_2$	$a \rightarrow \lambda$	a corresponds to predicates, functions or constants	1	1	∞
$t_1 = t_2$	$a = b$	$a = b$; a and b correspond to predicates, functions or constants	0	0	0

follows:

Matching via abstraction. Given the correspondences produced by the node matching and based on the work in [3], the following abstraction operations are used in order to select only those correspondences that preserve the desired properties, namely that functions are matched to functions and variables to variables:

Predicate: Two or more predicates are merged, typically to the least general generalization 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 generalization in the domain type hierarchy, e.g., *Acura + Nissan* $\mapsto$ *Nissan*. Similarly to the previous item we call *Nissan* a domain abstraction of *Acura* or *Nissan* $\sqsupseteq_{D}$ *Acura*.

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)*.

For example, predicate and domain abstraction/refinement operations do not convert a function into a variable. Therefore, the one-to-many correspondences returned by the node matching should be further filtered based on the allowed abstraction/refinement operations. For instance, the correspondence that binds Region in $T1$ and Region in $T2$ should be discarded, while the correspondence that binds Region in $T1$ and Area in $T2$ should be preserved.

Tree edit distance via abstraction operations. We look for a composition of the abstraction/refinement operations allowed for the given relation R that are necessary to convert one tree into another. We represent abstraction/refinement operations as tree edit distance operations applied to the term trees.

The tree edit distance problem involves three operations: (i) vertex deletion ($\upsilon \rightarrow \lambda$), (ii) vertex insertion ($\lambda \rightarrow \upsilon$), and (iii) vertex replacement ($\upsilon \rightarrow \omega$) [6]. 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 redefine the tree edit distance operations in a way that will allow them to have one-to-one correspondence to the abstraction/refinement operations, see Table 1.

Global similarity between trees. Since we compute the composition of the abstraction/refinement operations that are necessary to convert one term tree into the other, we are interested in the minimal cost of this composition. Global similarity between two trees is computed as shown in Eq. 1, where S stands for the set of the allowed tree edit operations; k_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, see Table 1.

$$TreeSim(T1,T2) = 1 - \frac{min \sum_{i \in S} k_i * Cost_i}{max(sizeof(T1), sizeof(T2))} \quad (1)$$

The highest value of *TreeSim* computed for $Cost_{T1=T2}$, $Cost_{T1\sqsubseteq T2}$ and $Cost_{T1\sqsupseteq T2}$ is selected as the one ultimately returned. In the case of example of Figure 1, when we match $T1$ with $T2$, *TreeSim* would be 0.62 for both $Cost_{T1=T2}$ and $Cost_{T1\sqsubseteq T2}$.

3 EVALUATION

We have evaluated our approach on different versions of SUMO and AKT ontologies[4]. These are both first-order ontologies, out of which 132 pairs of trees (first-order logic terms) were used. The matching quality results are shown in Figure 2. Note that F-Measure values exceed 70% for the given range of the cut-off thresholds. The average execution time per matching task on a standard laptop was 93ms.

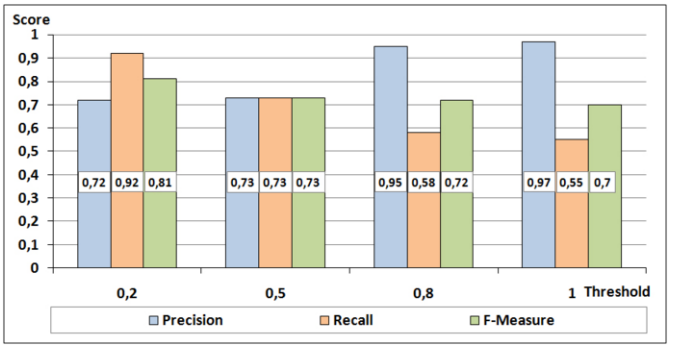

Figure 2: Evaluation results.

4 CONCLUSIONS

We have presented an approximate structure preserving semantic matching approach that implements the *structure preserving matching* operation. It is based on a theory of abstraction and a tree edit distance. We have evaluated our solution with encouraging results. Future work includes conducting extensive and comparative testing in real-world scenarios.

Acknowledgements. We appreciate support from the OpenKnowledge European STREP (FP6-027253).

REFERENCES

[1] J. Euzenat and P. Shvaiko, *Ontology matching*, Springer, 2007.
[2] F. Giunchiglia and P. Shvaiko, 'Semantic matching', *The Knowledge Engineering Review*, **18**(3), (2003).
[3] F. Giunchiglia and T. Walsh, 'A theory of abstraction', *Artificial Intelligence*, **57**(2-3), (1992).
[4] F. Giunchiglia, M. Yatskevich, and P. Shvaiko, 'Semantic matching: Algorithms and implementation', *Journal on Data Semantics*, **IX**, (2007).
[5] M. Klusch, B. Fries, and K. Sycara, 'Automated semantic web service discovery with OWLS-MX', in *Proceedings of AAMAS*, (2006).
[6] K.-C. Tai, 'The tree-to-tree correction problem.', *Journal of the ACM*, **26**(3), (1979).

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