APPROXIMATE STRUCTURE-PRESERVING SEMANTIC MATCHING

Fausto Giunchiglia, Fiona McNeill, Mikalai Yatskevich, Juan Pane, Paolo Besana and Pavel Shvaiko

August 2008

Technical Report # DISI-08-039

Also: accepted to the 7th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2008), Monterrey, Mexico, Nov 11 - 13, 2008.

Approximate structure-preserving semantic matching

Fausto Giunchiglia[1], Fiona McNeill[2], Mikalai Yatskevich[1], Juan Pane[1], Paolo Besana[2], Pavel Shvaiko[3]

[1] University of Trento, Povo, Trento, Italy,
{fausto|yatskevi|pane|pavel|}@dit.unitn.it
[2] University of Edinburgh, Scotland,
f.j.mcneill@ed.ac.uk|p.besana@sms.ed.ac.uk
[3] TasLab, Informatica Trentina, Italy,
Pavel.Shvaiko@infotn.it

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, for applications such as web service integration, we 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 semantic matching*. This operation takes two graph-like structures and produces a set of correspondences, (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. Our approach is based on a formal theory of abstraction and on a tree edit distance measure. We have evaluated our solution in various settings. Empirical results show the efficiency and effectiveness of our approach.

1 Introduction

Ontology matching is a critical operation in many applications, such as Artificial Intelligence, the Semantic Web and e-commerce. It takes two graph-like structures, for instance, lightweight ontologies [9], and produces an alignment, that is, a set of correspondences, between the nodes of those graphs that correspond semantically to one another [6].

Many varied solutions of matching have been proposed so far; see [6,29,24] for recent surveys[4]. In this paper we introduce a particular type of matching, namely *Structure-preserving semantic matching (SPSM)*. In contrast to conventional ontology matching, which aims to match single words through considering their position in hierarchical ontologies, structure-preserving semantics matching aims to match complex, structured terms. These terms are not structured according to their semantics, as terms are in an ontology, but are structured to express relationships: in the case of our approach, first-order relationships. This structure-preserving matching is therefore a two-step process, the first step of which is to match individual words within the terms through techniques used for conventional ontology matching, and the second - and novel - step of

[4] See, `http://www.ontologymatching.org` for a complete information on the topic.

which is to match the structure of the terms. For example, consider a first-order relation $buy(car, price)$ and another, $purchase(price, vehicle, number)$, both expressing buying relations between vehicles and cost. If the words used in these terms are from known ontologies, then we can use standard ontology matching techniques to determine, for example, that buy is equivalent to $purchase$ and that car is a sub-type of $vehicle$. If they are not from known ontologies we can still use WordNet to gather this information. Our work is concerned with understanding and using this information about how the words are related to determine how the full structured terms are related. Therefore, SPSM needs to preserve a set of structural properties (e.g., vertical ordering of nodes) to establish whether two graphs are globally similar and, if so, how similar they are and in what way. These characteristics of matching are required in web service integration applications, see, e.g., [21,23,18].

More specifically, most of the previous solutions to web service matching 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 for the reduction of the matching problem to the problem of reasoning within the shared ontology [21,27]. In contrast, following the work in [1,26,31], we assume that web services are described using terms from different ontologies and that their behavior is described using complex terms; we consider first-order terms. This allows us to provide detailed descriptions of the web services' input and output behavior. The problem becomes therefore that of matching two web service descriptions, which in turn, can be viewed as first-order terms and represented as tree-like structures. An alignment between these structures is considered as successful only if two trees are *globally* similar, e.g., $tree_1$ is 0.7 similar to $tree_2$, according to some measure in [0 1]. A further requirement is that the alignment must preserve certain structural properties of the trees being considered. In particular, the syntactic types and sorts have to be preserved: (i) a function symbol must be matched to a function symbol and (ii) a variable must be matched to a variable. We are mainly interested in approximate matching, since two web service descriptions may only rarely match perfectly.

The contributions of this paper include: (i) a new approach to approximate web service matching, called *Structure-preserving semantic matching (SPSM)*, and (ii) an implementation and evaluation of the approach in various settings (both with automatically generated tests and real-world first-order ontologies) with encouraging results. SPSM takes two tree-like structures and produces an alignment between those nodes of the trees that correspond semantically to one another, preserving the above mentioned two structural properties of the trees being matched, and only in the case that the trees are globally similar. Technically, the solution is based on the fusion of ideas derived from the theory of abstraction [11,12] and tree edit distance algorithms [3]. To the best of our knowledge, this is the first work taking this view.

The rest of the paper is organized as follows. Section 2 explains how calls to web services can be viewed as first-order trees. It also provides a motivating example. We overview the approximate SPSM approach in Section 3, while its details, such as abstraction operations, their correspondence to tree edit operations as well as computation of global similarity between trees are presented in Section 4 and Section 5, respectively.

Evaluation is discussed in Section 6. Section 7 relates our work to similar approaches. Finally, Section 8 summarizes the major findings.

2 Matching Web Services

Our hypothesis is that we can consider web services inputs and outputs as trees and therefore apply SPSM to calls to web services. This kind of structural matching can then allow us to introduce flexibility to calls to web services so that we no longer need to rely on (i) terms used in these calls coming from a global ontology; instead local ontologies adapted to purpose can be used; (ii) the structuring of these calls being fixed.

The structure is important because each argument in a call to a web service is defined according to its position in the input or output. However, expecting this structure to be fixed is just as problematic as expecting a global ontology. Individual web service designers will use different structure just as they will use different vocabulary and changes to web service descriptions over time will be mean that previous calls to web services become inappropriate. In order to remove the need both for a global ontology and a fixed structure for every web service call, we therefore need to employ structured matching techniques for matching between web service calls and returns and web service inputs and outputs.

The first-order terms that we match do not distinguish between inputs and outputs in the same manner as, for example, Web Service Description Language (WSDL). Instead, both inputs and outputs are arguments of the same predicate. In Prolog notation, this is indicated by using a $+$ for an input and a $-$ for an output. Thus the term:

$$purchase(-Price, +Vehicle, +Number)$$

indicates that $Vehicle$ and $Number$ are inputs and $Price$ is an output. During run-time, we can distinguish between inputs and outputs because inputs must be instantiated and outputs must be uninstantiated. In order to use our tree matching techniques for web services, we therefore make use of an automated translation process we have created that will map between a first-order term such as the above and a standard WSDL representation of the same information. This approach can also be used for other kinds of services in addition to web services; all that is required is that a translation process is created to convert between the representation of the service and first-order terms.

We make the assumption that web services written in WSDL will contain some kind of semantic descriptions of what the inputs and outputs are: that arguments are labelled descriptively and not merely as ‘input1’ and so on. This is after all what WSDL, as a description language, is designed to do. We appreciate that in practice designers of web services adopt a lazy approach and label inputs and outputs with terms that do not describe their semantics, especially when the WSDL files are generated automatically from classes or interfaces written in a programming language. In such cases, our techniques will have a very low success rate. However, such web services are of little value for any automated process and do not make use of the full potential of WSDL. We believe that as they become more widely used, the need for them to be properly descriptive becomes imperative so that they can be located and invoked automatically. In the meantime, any mark-up that is used to provide semantics for web services outside of the WSDL can also be amenable to our techniques, provided, as is usually the case, that descriptions of inputs and outputs can be expressed as a tree.

Let us consider an example of approximate SPSM between the following 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.

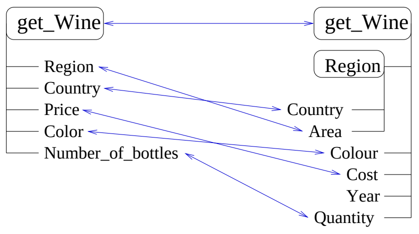

Fig. 1: Two approximately matched web services represented as trees: $T1$: get_wine(Region, Country, Color, Price, Number_of_bottles) and $T2$: get_wine(Region(Country, Area), Colour, Cost, Year, Quantity). Functions are in rectangles with rounded corners; they are connected to their arguments by dashed lines. Node correspondences are indicated by arrows.

In order to guarantee 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 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.

3 Overview of 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 [14]. Technically, two nodes $n_1 \in T1$ and $n_2 \in T2$ match iff: $c@n_1 \; R \; c@n_2$ holds,

Matcher name	Execution order	Approximation level	Matcher type	Schema info
WordNet	1	1	Sense-based	WordNet senses
Prefix	2	2	String-based	Labels
Suffix	3	2	String-based	Labels
Edit distance	4	2	String-based	Labels
Ngram	5	2	String-based	Labels

Table 1: Element level matchers. The first column contains the names of the matchers. The second column lists the order in which they are executed. The third column introduces the matcher's approximation level. The relations produced by a matcher with the first approximation level are always correct. Notice that matchers are executed following the order of increasing approximation. The fourth column reports the matcher's type, while the fifth column describes the matcher's input, see [14] for details.

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 [10] as implemented in the S-Match system [14] 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 [8]. Besides WordNet, the basic version of S-Match also uses four string-based matchers, see Table 1. 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 [13]. 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, in turn, exploits the results of the node matching and the structure of the trees to find if these globally match each other. Specifically, given the correspondences produced by the node matching, the abstraction operations (§4) 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. Thus, for example, 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. Then, the preserved correspondences are used as allowed operations of a tree edit distance in order to determine global similarity (§5) between trees under consideration. If this global similarity measure is higher than an empirically established threshold, the trees are considered to be similar enough, and not similar otherwise. Technically, two trees $T1$ and $T2$ approximately match iff there is at least one node n_{1i} in $T1$ and a node n_{2j} in $T2$ 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}, where i=1,...,N1; j=1,...,N2; N1 and N2 are the number of nodes in $T1$ and $T2$, respectively.

Semantic heterogeneity is therefore reduced to two steps: (i) matching the web services, thereby obtaining an alignment, and (ii) using this alignment for the actual web service integration. This paper focuses only on the matching step.

4 Matching Via Abstraction

In this section we first discuss the abstraction operations (§4.1), then discuss how these operations are used in order to drive a tree edit distance computation (§4.2), and, finally, discuss the implementation details (§4.3).

4.1 Abstraction Operations

The work in [12] categorizes the various kinds of abstraction operations in a wide-ranging survey. It also introduces a new class of abstractions, called TI-abstractions (where TI means "Theorem Increasing"), which have the fundamental property of maintaining completeness, while loosing 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 loosing 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.

The work in [12] provides the following major categories of abstraction operations:

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., *Micra* + *Nissan* $\mapsto$ *Nissan*. Similarly to the previous item we call *Nissan* a domain abstraction of *Micra* or *Nissan* $\sqsupseteq_D$ *Micra*. Conversely, we call *Micra* a domain refinement of *Nissan* or *Micra* $\sqsubseteq_D$ *Nissan*.

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*.

Let us consider the following pair of first-order terms *(Bottle A)* and *(Container)*. In this case there is no abstraction/refinement operation that makes them equivalent. However, consequent applications of propositional and domain abstraction operations make the two terms equivalent:

$$\textit{(Bottle A)} \mapsto^{\sqsubseteq_P} \textit{(Bottle)} \mapsto^{\sqsubseteq_D} \textit{(Container)}$$

In fact the relation holding among the terms is a composition of two refinement operations, namely *(Bottle A)* $\sqsubseteq_P$ *(Bottle)* and *(Bottle)* $\sqsubseteq_D$ *(Container)*.

The abstraction/refinement operations discussed above allow us to preserve the desired properties: that functions are matched to functions and variables to variables. 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: $\{=, \sqsupseteq, \sqsubseteq\}$, where $=$ stands for equivalence; $\sqsupseteq$ represents an abstraction relation and connects the precondition and the result of a composition of arbitrary number 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 number of predicate, domain and propositional refinement operations.

Since abstractions and refinements cover every way in which first-order terms can differ (either in the predicate, in the number of arguments or in the types of arguments), we can consider every relation between terms that are in some way related as a combination of these six basic refinements and abstractions. Therefore, every map between first-order trees can be described using these operations. The only situation in which we cannot use these techniques is if there is no semantic relation between the predicates of the two terms, but in this situation, a failed mapping is the appropriate outcome since we do not consider them to be related even though the arguments may agree. Note that we can match non-related arguments using these operations by applying propositional abstraction and then propositional refinement.

4.2 Tree Edit Distance Via Abstraction Operations

Now that we have defined the operations that describe the differences between trees, we need some way of composing them so that we can match entire trees to one another. We look for a composition of the abstraction/refinement operations allowed for the given relation R (see §3) that are necessary to convert one tree into another. In order to solve this problem we propose to represent abstraction/refinement operations as tree edit distance operations applied to the term trees.

In its traditional formulation, the tree edit distance problem considers three operations: (i) vertex deletion, (ii) vertex insertion, and (iii) vertex replacement [32]. Often these operations are presented as rewriting rules:

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

where υ and ω correspond to the labels of nodes in the trees while λ stands for the special blank symbol.

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. Table 2 illustrates the correspondence between abstraction/refinement and tree edit operations. Let us focus for the moment on the first three columns of Table 2. 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.

Table 2: 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***$_{T1\sqsupseteq T2}$
$t_1 \sqsupseteq_{Pd} t_2$	$a \to b$	$a \sqsupseteq b$; a and b correspond to predicates	1	∞	1
$t_1 \sqsupseteq_D t_2$	$a \to b$	$a \sqsupseteq b$; a and b correspond to functions or constants	1	∞	1
$t_1 \sqsupseteq_P t_2$	$a \to \lambda$	a corresponds to predicates, functions or constants	1	∞	1
$t_1 \sqsubseteq_{Pd} t_2$	$a \to b$	$a \sqsubseteq b$; a and b correspond to predicates	1	1	∞
$t_1 \sqsubseteq_D t_2$	$a \to b$	$a \sqsubseteq b$; a and b correspond to functions or constants	1	1	∞
$t_1 \sqsubseteq_P t_2$	$a \to \lambda$	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

Let us consider, for example, the first line of Table 2. 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 t_1 of the first tree that replaces the node a with the node b of the second tree $(a \to b)$. Moreover, the operation can be applied only in the case that: (i) label a is a generalization of label b and (ii) both nodes with labels a and b in the term trees correspond to predicates in the first-order terms.

4.3 Implementation

We have implemented our approximate SPSM solution in Java. Many existing tree edit distance algorithms allow the tracking of the nodes to which a replace operation is applied. According to [32], the minimal cost correspondences are: (i) one-to-one, (ii) horizontal order preserving between sibling nodes, and (iii) vertical order preserving. The alignment depicted in Figure 1 complies with (i), (iii) and violates (ii). In fact, the fourth sibling Color in $T1$ is matched to the second sibling Colour in $T2$ (see below for an explanation).

For the tree edit distance operations depicted in Table 2, 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 an alignment among the nodes of the term trees with the desired properties, i.e., that there are only one-to-one correspondences in it and that functions are matched to functions and variables are matched 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 2, have a one-to-one correspondence with abstraction/refinement operations.

At the same time, an alignment used in a tree edit distance computation preserves the horizontal order among the sibling nodes, but this is not a desirable property for the web service integration purposes. In fact, we would want the fourth sibling Colour in $T1$ to match the second sibling Color in $T2$ of Figure 1. However, as from Table 2, 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 generalizations or specializations of each other, as computed by the S-Match node matching algorithm. Therefore, given the alignment produced by the S-Match node matching algorithm, we identify the cases when the horizontal order between sibling nodes is not preserved and change the ordering of the sibling nodes to make the alignment horizontal order preserving. For example, swapping the nodes Cost and Colour in $T2$ of Figure 1 does not change the meaning of these terms but it allows the correspondence holding between Colour and Color in Figure 1 to be included in the alignment without increasing the cost during the tree edit distance computation. This switching means that the original horizontal order of siblings is not preserved in most cases. If there are arguments with identical names, such cases are resolved with the help of indexing schemes.

5 Global Similarity Between Trees

Our goal now is to compute the similarity between two term 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 a minimal cost of this composition. Therefore, we have to determine the minimal set of operations which transforms one tree into another, see Eq. 1:

$$\mathit{Cost} = min \sum_{i \in S} k_i * \mathit{Cost}_i \tag{1}$$

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. Our goal here is to define the Cost_i in a way that models the semantic distance.

A possible uniform proposal is to assign the same unit cost to all tree edit operations that have their abstraction theoretic counterparts. The last three columns of Table 2 illustrate the costs of the abstraction/refinement (tree edit) operations, depending on the relation (equivalence, abstraction or refinement) being computed between trees. Notice that the costs for estimating abstraction ($\sqsupseteq$) and refinement ($\sqsubseteq$) relations have to be adjusted according to their definitions. In particular, the tree edit operations corresponding to abstraction/refinement operations that are not allowed by 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.

Let us consider, for example, the first line of Table 2. The cost of the tree edit distance operation that corresponds to the predicate abstraction ($t_1 \sqsupseteq_{Pd} t_2$) is equal to 1 when used for the computation of equivalence ($\mathit{Cost}_{T1=T2}$) and abstraction ($\mathit{Cost}_{T1 \sqsupseteq T2}$) relations between trees. It is equal to ∞ when used for the computation of refinement ($\mathit{Cost}_{T1 \sqsubseteq T2}$) relation.

Eq. 1 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 a tree edit distance into the similarity score:

$$\mathit{TreeSim} = 1 - \frac{\mathit{Cost}}{\mathit{max}(\mathit{T1}, \mathit{T2})} \tag{2}$$

where $Cost$ is taken from Eq. 1 and is normalized by the size of the biggest tree. Note that for the special case of *Cost* equal to ∞, *TreeSim* is estimated as 0. Finally, the highest value of *TreeSim* computed for $\mathit{Cost}_{T1=T2}$, $\mathit{Cost}_{T1\sqsubseteq T2}$ and $\mathit{Cost}_{T1\sqsupseteq T2}$ is selected as the one ultimately returned. For example, in the case of example of Figure 1, when we match $T1$ with $T2$ this would be 0.62 for both $\mathit{Cost}_{T1=T2}$ and $\mathit{Cost}_{T1\sqsubseteq T2}$.

6 Evaluation

On top of the implementation discussed in §4.3 we exploited a modification of simple tree edit distance algorithm from [34]. The evaluation set-up is discussed in §6.1, while the evaluation results are presented in §6.2.

6.1 Evaluation Set-up

Ontology and web service engineering practices suggest that often the underlying trees to be matched are derived or inspired from one another. Therefore, it is reasonable to compare a tree with another one derived from the original one. We have evaluated efficiency and quality of the results of our matching solution on two test cases. Note that this is not the largest data set we have access to - a larger set is described, for example, it [15]. However, such data sets are not useful to us in this instance because they do not allow us to evaluate our approximate matching.

Test case 1: real-world ontologies. We used different versions of the Standard Upper Merged Ontology (SUMO)[5] and the Advance Knowledge Transfer (AKT)[6] 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 and 2.1, and 2.1 and 2.2 of the AKT-portal and AKT-support ontologies[7]. These are all first-order ontologies (hence, their expressivity is far beyond generalization/specialization hierarchies), so many of these differences matched well to the potential differences between terms that we are investigating. However, some of them were more complex, such as differences in inference rules, and had no parallel in our work; therefore, these were discarded, and our tests were run on all remaining differences. Specifically, 132 pairs of trees (first-order logic terms) were used. Half of the pairs were composed of the equivalent terms (e.g., journal(periodical-publication) and magazine (periodical-publication))

[5] `http://ontology.teknowledge.com/`

[6] `http://www.aktors.org`

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

while the other half was composed from similar but not equivalent terms (e.g., web-reference(publication-reference) and thesis-reference (publication-reference)).

Test case 2: systematic benchmarks. Different application programming interfaces (APIs) suggest that the terms within a tree are likely not to be semantically related to each other. Examples from the Java API include: set(index, element) and put(key, value). Thus, trees can be considered as being composed of nodes whose labels are random terms.

This test case was composed of trees that are alterations of the original trees. Unlike the work on systematic benchmarks in Ontology Alignment Evaluation Initiative-OAEI [5], the original trees here were generated automatically. We have generated 100 trees. For each original tree, 30 altered ones were created, see Table 3. Pairs composed of the original tree and one varied tree were fed to our SPSM solution. The experiment described above was repeated 5 times in order to remove noise in the results.

For tree generation, node labels were composed of a random number of words, selected from 9000 words extracted from the Brown Corpus[8]. The average number of nodes per tree was 8; in fact, functions usually have fewer parameters. In turn, the tree alterations were inspired by the approach in [5]. These are summarized in Table 3 and include: (i) syntactic alterations, such as adding or removing characters, and (ii) semantic alterations, word addition in labels by using related words (e.g., synonyms) extracted from the Moby thesaurus[9]. The probabilities used for these two types of alterations represent the fact that in most of the cases (0.8) the modifications made during an evolution process concern the altering in meaning, while syntactic modifications, such as introducing acronyms, usually have less occurrences (0.3).

Table 3: Parameters used for generating and modifying the trees

Parameter	**Syntactic**	**Semantic**	**Combined**
Number of trees	100	100	100
Number of modifications per tree	30	30	30
Average number of nodes per tree	8	8	8
Probability of replacing a word in a node label for a related one	0.0	0.8	0.8
Probability of making a syntactic change in a word of a node label	0.3	0.0	0.3

Since the tree alterations made are known, these provide the ground truth, and hence, the reference results are available for free by construction, see also [5,22]. This allows for the computation of the matching quality measures. In particular, the standard matching quality measures, such as *Recall*, *Precision* and *F-measure* for the similarity between trees have been computed [6]. In computation of these quality measures we 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 considered as a single correspondence rather than two correspondences, namely journal=magazine and periodical-publication$_1$=periodical-publication$_2$.

[8] `http://icame.uib.no/brown/bcm.html`

[9] `http://www.mobysaurus.com/`. Since the SPSM node matching uses WordNet 2.1, an alternative thesaurus was used here.

The evaluation was performed on a standard laptop Core Duo CPU-2Ghz, 2GB RAM, with the Windows Vista operating system, and with no applications running but a single matching system.

6.2 Evaluation Results

The matching quality results for the first test case are shown in Figure 2. Quality measures depend on the cut-off threshold values and the SPSM solution demonstrates high matching quality on the wide range of these values. In particular, F-Measure values exceed 70% for the given range.

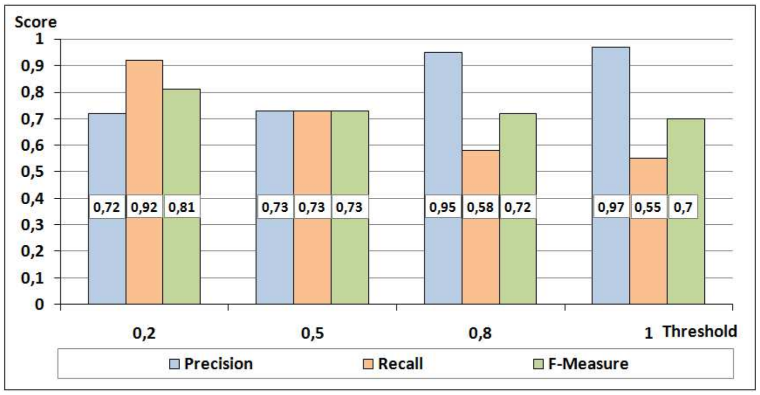

Fig. 2: Test case 1: Evaluation results.

The evaluation results for the second test case are summarized in Figures 3, 4 and 5. In order to obtain those results there have been used: (i) the tree matcher discussed in §4 and §5 and (ii) the various matchers used in isolation (namely, edit distance, NGram, prefix, suffix and WordNet) and all these matchers as combined by S-Match, see Table 1. Figures 3, 4 and 5 are composed of four plots (from top to bottom): (i) standard precision-recall plot, (ii) recall *vs* various cut-off threshold values in [0 1], (iii) precision *vs* various cut-off threshold values in [0 1], and (iv) F-measure *vs* various cut-off threshold values in [0 1].

In particular, Figure 3 shows that for the syntactic alterations, as expected, string-based matchers outperform the WordNet matcher. Also, edit distance performs as well as S-Match. The best performance in terms of F-Measure (which is 0.52) is reached at the threshold of 0.8. In turn, Figure 4 shows that for the semantic alterations, as expected, the WordNet matcher outperforms the string-based matchers. The best performance in terms of F-Measure (which is 0.73) is demonstrated by S-Match and is reached at the threshold of 0.8. Finally, Figure 5 shows that when both types of alterations, namely syntactic and semantics, are applied the best performance in terms of F-Measure (which is 0.47) is demonstrated by S-Match and is reached at the threshold of 0.8.

The efficiency of our solution is such that the average execution time per matching task in the two test cases under consideration was 93ms. The quantity of main memory used by SPSM during matching did not rise more than 3Mb higher than the standby level. Finally, the evaluation results show that conventional ontology matching technology that we previously applied to matching classifications and XML schemas (see [14]) can also provide encouraging results in the web services domain. Of course, additional extensive testing is needed, especially with WSDL services, for example as done in [31].

7 Related Work

We believe that this approach to structured matching is unique and therefore it is difficult to perform any comparative analysis. In order to demonstrate that we make use of powerful ontology matching tools for the standard ontology matching step of the process, we can compare S-Match against other ontology matching tools. However, the full structure-preserving semantic matching addresses a previously unsolved problem. In this section, we discuss other methods that address similar problems.

Our work builds on standard work in tree-edit distance measures, for example, as espoused by [28]. The key difference with our work is the integration of the semantics that we gain through the application of the abstraction and refinement rules. This allows us to consider questions such as *what is the effect to the overall* meaning *of the term (tree) if* node a *is relabelled to* node b?, or *how significant is the removal of a node to the overall semantics of the term*? These questions are crucial in determining an intuitive and meaningful similarity score between two terms, and are very context dependent. Altering the scores given in Table 2 enables us to provide different answers to these questions depending on the context, and we are working on giving providing even more subtle variations of answers reflecting different contexts (see Section 8).

Work based on these ideas, such as Mikhaiel and Stroudi's work on HTML differencing [16], tends to focus only on the structure and not on the semantics. This work never considers what the individual nodes in their HTML trees mean and only considers context in the sense that, for example, the cost of deleting a node with a large subtree is higher than the cost of deleting a leaf node; the semantic meanings of these nodes is not considered.

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 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 [21,23,27] for example. Probably the most similar to ours is the approach taken in METEOR-S [1] and in [26], 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 structure provided by the first order term.

Many diverse solutions to the ontology matching problem have been proposed so far. See [29] for a comprehensive survey and [7,25,4,17,2,20,30] 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) [33]. The approach taken in [19] 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.

In summary, much work has been done on structure-preserving matching and much has been done on semantic matching, and our work depends heavily on the work of others in these fields. The novelty of our work is in the combination of these two approaches to produce a structure-preserving semantic matching algorithm, thus allowing us to determine fully how structured terms, such as web service calls, are related to one another.

8 Conclusions and Future Work

We have presented an approximate SPSM approach that implements the *SPSM* operation. It is based on a theory of abstraction and a tree edit distance. We have evaluated our solution on test cases composed of hundreds of trees. The evaluation results look promising, especially with reference to the efficiency indicators.

Future work proceeds at least along the following directions: (i) studying a best suitable cost model, (ii) incorporating preferences in order to drive approximation, thus allowing/prohibiting certain kinds of approximation (e.g., not approximating red wine with white wine, although these are both wines), and (iii) conducting extensive and comparative testing in real-world scenarios.

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

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*, 2004.
2. S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1), 1999.
3. W. Chen. New algorithm for ordered tree-to-tree correction problem. *Journal of Algorithms*, 40(2), 2001.
4. M. Ehrig, S. Staab, and Y. Sure. Bootstrapping ontology alignment methods with APFEL. In *Proceedings of ISWC*, 2005.
5. J. Euzenat, A. Isaac, C. Meilicke, P. Shvaiko, H. Stuckenschmidt, O. Šváb, V. S., W. van Hage, and M. Yatskevich. Results of the ontology alignment evaluation initiative 2007. In *Proceedings of the ISWC + ASWC International Workshop on Ontology Matching (OM)*, 2007.
6. J. Euzenat and P. Shvaiko. *Ontology matching*. Springer, 2007.

7. J. Euzenat and P. Valtchev. Similarity-based ontology alignment in OWL-lite. In *Proceedings of ECAI*, 2004.
8. C. Fellbaum. *WordNet: an electronic lexical database*. MIT Press, 1998.
9. F. Giunchiglia, M. Marchese, and I. Zaihrayeu. Encoding classifications into lightweight ontologies. *Journal on Data Semantics*, VIII, 2007.
10. F. Giunchiglia and P. Shvaiko. Semantic matching. *The Knowledge Engineering Review*, 18(3), 2003.
11. F. Giunchiglia and T. Walsh. Abstract theorem proving. In *11th international joint conference on artificial intelligence (IJCAI'89)*, volume 1, Detroit, Mich., 20-25 August 1989.
12. F. Giunchiglia and T. Walsh. A theory of abstraction. *Artificial Intelligence*, 57(2-3), 1992.
13. F. Giunchiglia, M. Yatskevich, and E. Giunchiglia. Efficient semantic matching. In *Proceedings of ESWC*, 2005.
14. F. Giunchiglia, M. Yatskevich, and P. Shvaiko. Semantic matching: Algorithms and implementation. *Journal on Data Semantics*, IX, 2007.
15. Fausto Giunchiglia, Mikalai Yatskevich, Paolo Avesani, and Pavel Shvaiko. A large scale dataset for the evaluation of ontology matching systems. *The Knowledge Engineering Review Journal*, 2008, to appear.
16. R. Gligorov, Z. Aleksovski, W. ten Kate, and F. van Harmelen. Accurate and efficient html differencing. In *Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice (STEP)*, pages 163–172. IEEE Press, 2005.
17. R. Gligorov, Z. Aleksovski, W. ten Kate, and F. van Harmelen. Using google distance to weight approximate ontology matches. In *Proceedings of WWW*, 2007.
18. N. Gooneratne and Z. Tari. Matching independent global constraints for composite web services. In *In Proceedings of WWW*, pages 765–774, 2008.
19. W. Hu and Y. Qu. Block matching for ontologies. In *Proceedings of ISWC*, 2006.
20. Y. Kalfoglou and M. Schorlemmer. IF-Map: an ontology mapping method based on information flow theory. *Journal on Data Semantics*, I, 2003.
21. M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with OWLS-MX. In *Proceedings of AAMAS*, 2006.
22. Y. Lee, M. Sayyadian, A. Doan, and A. Rosenthal. eTuner: tuning schema matching software using synthetic scenarios. *VLDB Journal*, 16(1), 2007.
23. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of WWW*, 2003.
24. N. Noy, A. Doan, and A. Halevy. Semantic integration. *AI Magazine*, 26(1), 2005.
25. N. Noy and M. Musen. The PROMPT suite: interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59(6), 2003.
26. S. Oundhakar, K. Verma, K. Sivashanugam, A. Sheth, and J. Miller. Discovery of web services in a multi-ontology and federated registry environment. *Journal of Web Services Research*, 2(3), 2005.
27. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of ISWC*, 2002.
28. Dennis Shasha and Kaizhong Zhang. Approximate tree pattern matching. In *In Pattern Matching Algorithms*, pages 341–371. Oxford University Press, 1997.
29. P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics*, IV, 2005.
30. U. Straccia and R. Troncy. oMAP: Combining classifiers for aligning automatically OWL ontologies. In *Proceedings of WISE*, 2005.
31. E. Stroulia and Y. Wang. Structural and semantic matching for assessing web-service similarity. *International Journal of Cooperative Information Systems*, 14(4):407–438, 2005.
32. K.-C. Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3), 1979.
33. 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), 2006.
34. G. Valiente. *Algorithms on Trees and Graphs*. Springer, 2002.

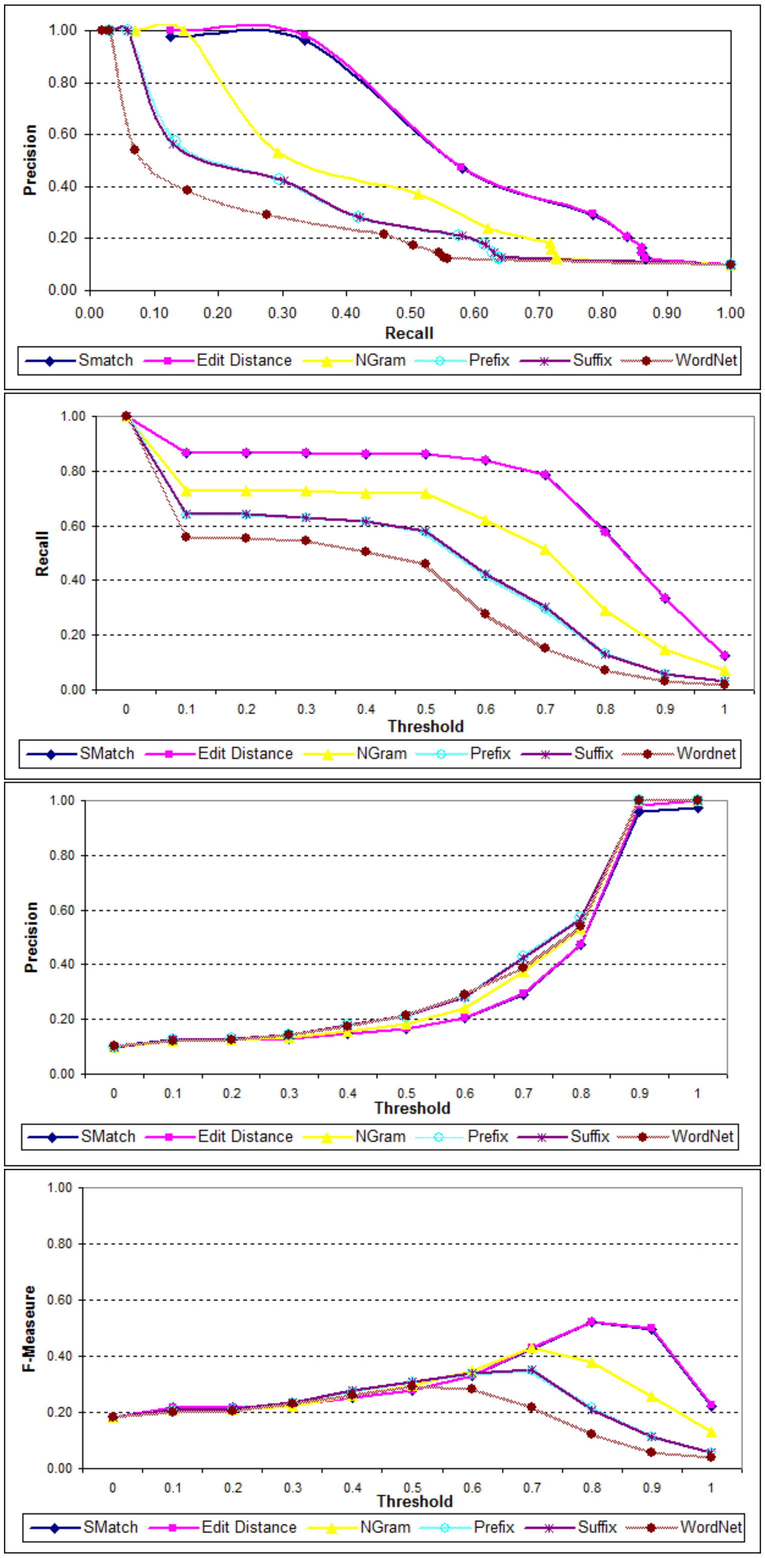

Fig. 3: Test case 2: Evaluation results for syntactic changes.

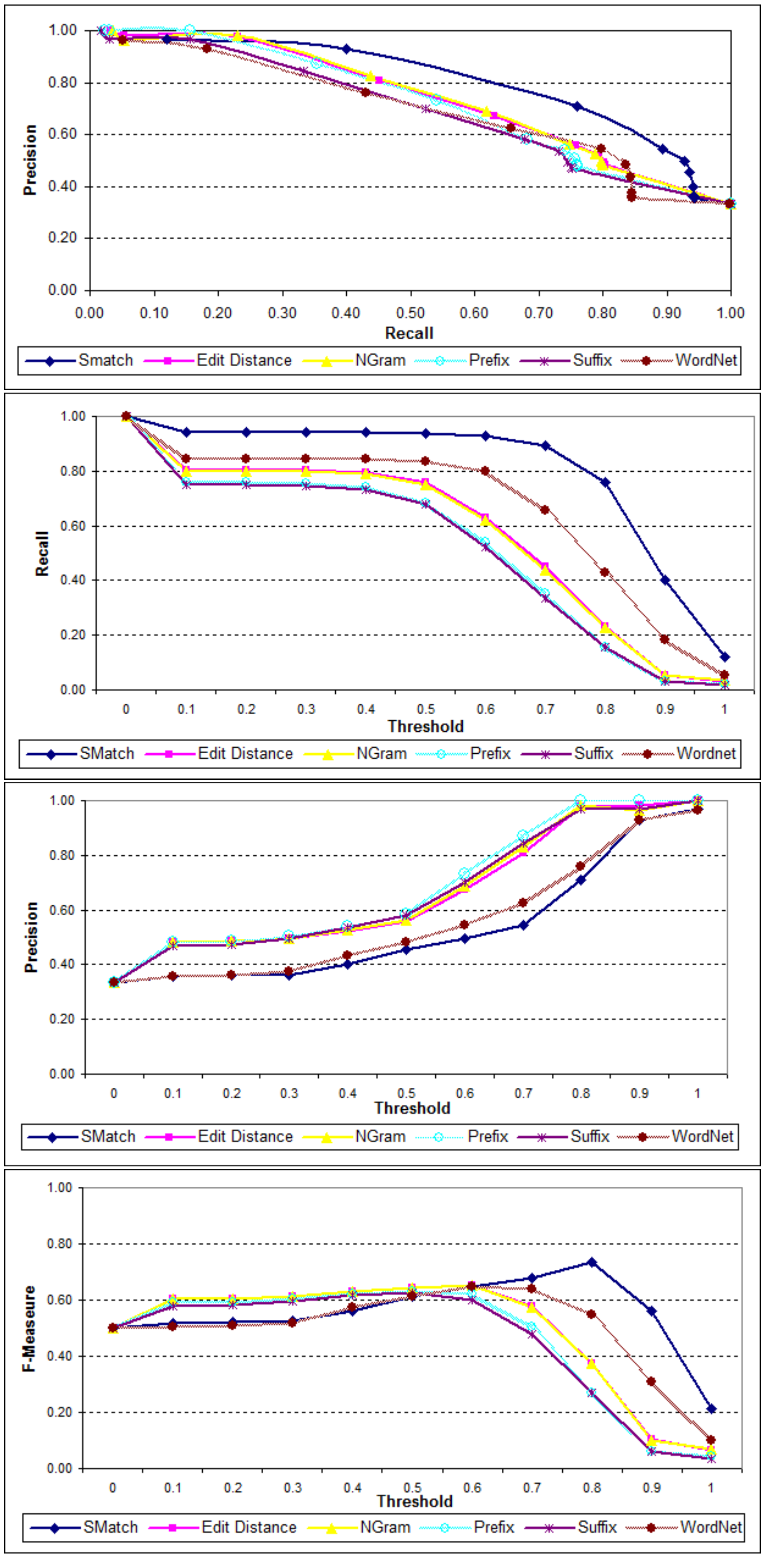

Fig. 4: Test case 2: Evaluation results for semantic changes.

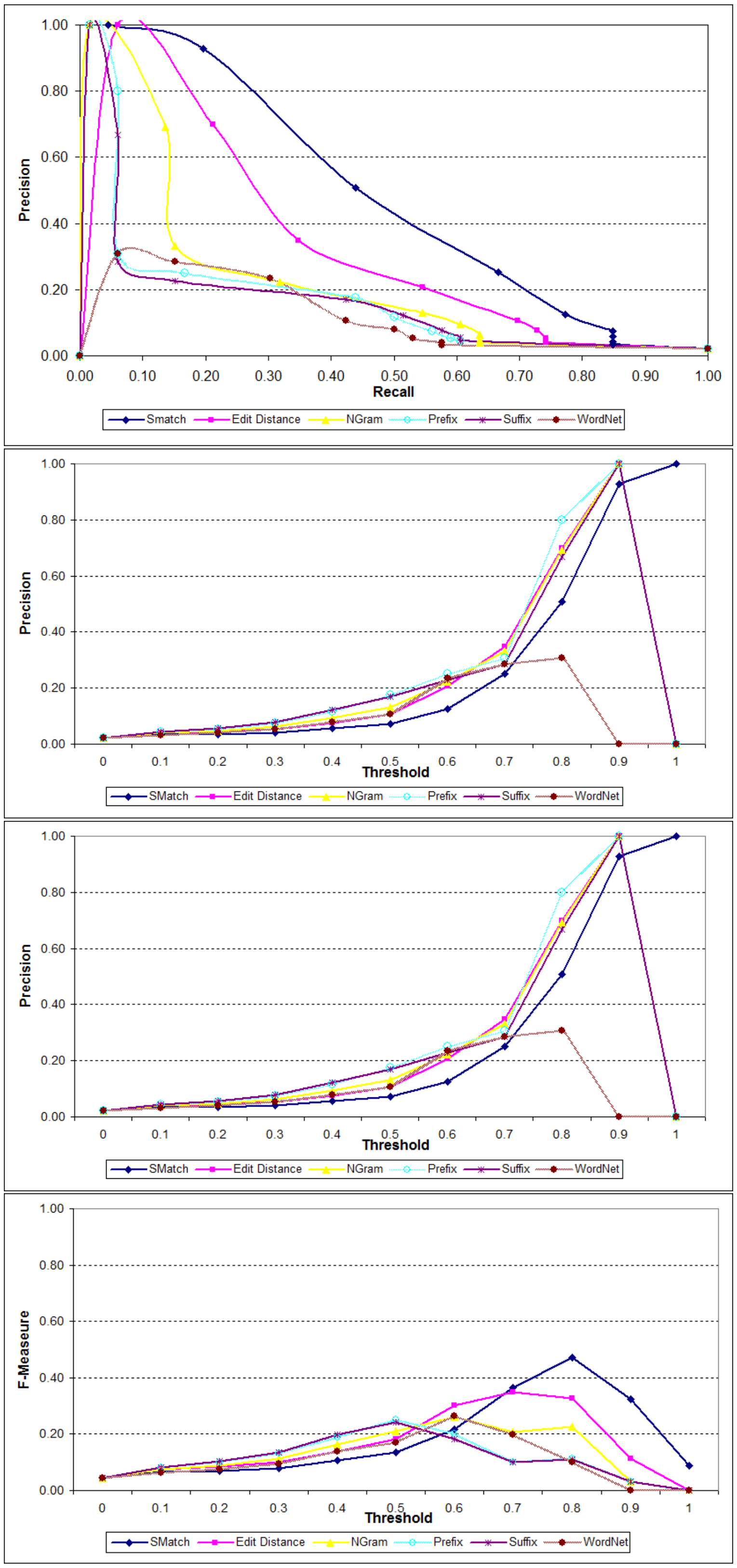

Fig. 5: Test case 2: Evaluation results for combined changes.