UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

ELEMENT LEVEL SEMANTIC MATCHING

Fausto Giunchiglia and Mikalai Yatskevich

June 2004

Technical Report # DIT-04-035

Element Level Semantic Matching

Fausto Giunchiglia, Mikalai Yatskevich

Dept. of Information and Communication Technology
University of Trento,
38050 Povo, Trento, Italy
{fausto, yatskevi}@dit.unitn.it

Abstract. We think of Match as an operator which takes two graph-like structures and produces a mapping between semantically related nodes. The matching process is essentially divided into two steps: element level and structure level. Element level matchers consider only labels of nodes, while structure level matchers start from this information to consider the full graph. In this paper we present various element level semantic matchers, and discuss their implementation within the S-Match system. The main novelty of our approach is in that element level semantic matchers return semantic relations ($=, \supseteq, \subseteq, \perp$) between concepts rather than similarity coefficients between labels in the [0, 1] range.

1. Introduction

We think of matching as the task of finding semantic correspondences between elements of two graph-like structures (e.g., conceptual hierarchies, database schemas or ontologies). Matching has been successfully applied to many well-known application domains, such as schema/ontology integration, data warehouses, and XML message mapping.

Semantic matching, as introduced in [4, 6], and its implementation within the S-Match system [7] are based on the intuition that mappings should be calculated between the concepts (but not labels) assigned to nodes. Thus, for instance, two concepts can be equivalent; one can be more general than the other, and so on. As from [6], all previous approaches are classified as syntactic matching. These approaches, though implicitly or explicitly exploiting the semantic information codified in graphs, differ substantially from our approach in that, instead of computing semantic relations between nodes, they compute syntactic "similarity" coefficients between labels, in the [0,1] range (see [6] for an in depth discussion about syntactic and semantic matching).

The system we have developed, called S-Match, takes two trees, and for any pair of nodes from these two trees, it computes the strongest semantic relation holding between them. In order to perform this, the matching task is articulated into two basic steps, namely element and structure level matching (See [7] for details). Element level matchers consider only the information at the atomic level (e. g., the information contained in elements of the schemas), while structure level matchers consider also the information about the structural properties of the schemas.

Our goal in this paper is to describe a set of element level semantic matchers, as they have been implemented within S-Match. In order to satisfy the input requirements of the structure level matchers the element level matchers return semantic relations (=, ⊒, ⊆, ⊥). Some matchers are modifications of previously developed syntactic matchers. The main novelty is the output returned. However, we have introduced two new methods for determining semantic words relatedness namely *semantic gloss comparison* and *extended semantic gloss comparison*.

The rest of the paper is organized as follows. Section 2 provides an overview of S-Match. Section 3 is dedicated to semantic element level matchers. *String based* matchers are discussed in Section 4, while *sense* and *gloss based* matchers are described in Sections 5 and 6, respectively. The descriptions of matchers are structured as follows. First, we give the overview of the matcher under consideration. Afterwards, we provide some examples of the matcher results with execution times (all tests were performed on a P4 computer with 512 Mb RAM installed). Finally, we discuss the results obtained. Section 7 concludes the paper.

2. S-Match: Algorithm and Implementation

According to [6] possible semantic relations returned by element level matchers are: *equivalence* (=); *more general* (⊒); *less general* (⊆); *mismatch* (⊥); *overlapping* (∩). They are ordered according to decreasing binding strength, e.g., from the strongest (=) to the weakest (∩). When no relations are found the special *Idk* (I don't know) relation returned.

As from [7], the S-Match algorithm is organized in the following four macro steps:

- *Step 1*: for all labels in the two trees, compute concepts denoted by labels
- *Step 2*: for all nodes in the two trees, compute concepts at nodes
- *Step 3*: for all pairs of labels in the two trees, compute semantic relations among concepts denoted by labels
- *Step 4*: for all pairs of nodes in the two trees, compute semantic relations among concepts at nodes

Let us consider, for instance, the two trees depicted in Figure 1a.

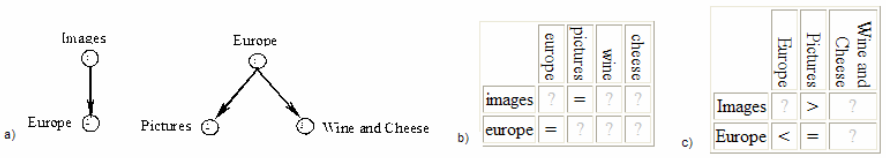

Fig. 1. Simple schemas (a). Matrices of relations between labels (b) and concepts at nodes (c).

During Step 1 we try to capture the meaning of the labels in the trees. In order to perform this we first tokenize the complex labels. Then for instance "*Wine and Cheese*" from Figure 1 becomes <Wine, and, Cheese>. Then, we lemmatize tokens; and "*Images*" becomes "*image*". Then, an Oracle (at the moment we use WordNet 2.0 as an Oracle) is queried in order to obtain the senses of the lemmatized tokens. Afterwards, these senses are attached to the atomic concepts. Finally, the complex concepts

are built from the atomic ones. Thus, the concept of label *Wine and Cheese*, $C_{Wine\ and\ Cheese}$ is computed as $C_{Wine\ and\ Cheese}$ = <*wine*, {*senses*$_{WN}$*#4*}>&<*cheese*, {*senses*$_{WN}$*#4*}>, where <*cheese*, {*senses*$_{WN}$*#4*}> is taken to be the union of the four WordNet senses.

Step 2 takes into account structural schema properties. The logical formula for a concept at node is constructed, most often as the conjunction of the formulas in the concept path to the root (see [7] for more details).

Element level semantic matchers are applied during Step 3 while determining the semantic relations between labels. For example, we can derive from WordNet the information that *image* and *picture* are synonyms (C_{Images} = $C_{Pictures}$). The relations between the atomic concepts in our example in Figure 1a are depicted in Figure 1b. Element level semantic matchers provide the input to the structure level matcher, which is applied on the Step 4 and produces the matching result, which is depicted in Figure 1c.

The pseudo code of Step 3 which contains the calls of element level semantic matchers is provided in Figure 2. **`getRelation`** takes two concept labels (*`sLabel`*, *`tLabel`*) and their WordNet senses (arrays *`sSenses`*, *`tSenses`*) as input, and produces a semantic relation between these two labels (*`rel`*). First, it tries to obtain the relation from WordNet (line 2). If it does not succeed (the result is equal to *Idk*) the *string, sense, and gloss based* matchers are executed in sequential order (line 4).

```
1.String getRelation(String[] sSenses,String[] tSenses,
                              String sLabel,String tLabel)
2.  String rel=getWordNetRel (sSenses,tSenses);
3.  if (rel==”Idk”)
4.    rel=getMLibRel(sLabel,tLabel,sSenses,tSenses);
5.  return rel;
```

Fig. 2. Pseudo code of weak semantic matchers library.

`getWordNetRel` takes two arrays of WordNet senses and produces the strongest semantic relation between any two senses. In order to perform this, it triple loops on relations, source, and target senses. If there no semantic relations are found, it returns *Idk* .

`getMLibRel` takes two labels and two arrays of WordNet senses and returns a semantic relation between them. In order to perform this, it sequentially executes different *string, sense, and gloss based* matchers. *String based* matchers are executed once for each pair of input labels. *Sense* and *gloss based* matchers are executed for each pair of concept senses and each possible semantic relation between them. If the matchers fail to determine the relation, *Idk* is returned.

Notice that element level semantic matchers are executed only in the case we cannot obtain the necessary information from *WordNet* which is the only element level matcher whose result is guaranteed to be correct.

3. Element level semantic matchers

S-Match is implemented in Java 1.4. The current version contains 13 semantic element level matchers listed in Table 1.

Table 1. Element level semantic matchers implemented so far

Matcher name	Execution Order	Approximation level	Matcher type	Schema info
Prefix	2	2	String based	Labels
Suffix	3	2		
Edit distance	4	2		
Ngram	5	2		
Text Corpus	13	3		Labels + Corpus
WordNet	1	1	Sense based	WordNet senses
Hierarchy distance	6	3		
WordNet Gloss	7	3	Gloss based	WordNet senses
Extended WordNet Gloss	8	3		
Gloss Comparison	9	3		
Extended Gloss Comparison	10	3		
Semantic Gloss Comparison	11	3		
Extended semantic gloss comparison	12	3		

The first column lists the matcher names. The second column lists the order in which they are called. The third column introduces the notion of approximation level. The relation produced by matcher with first approximation level is always considered to be correct (e. g., *auto=car* returned by *WordNet*). The relations of second approximation level matcher is likely to be correct (e.g., *net=network* but *hot=hotel* by *Suffix*). The third approximation level relations are fuzzier in the sense that they depend on the context of the matching task (e.g., *cat* can be considered equivalent to *dog* by *Extended Gloss Comparison* in the sense they are both pets). It can be notified that matchers are executed following the order of increasing approximation. The fourth column reports the matcher type. The fifth column reports the matchers' input. At the moment we have three main categories of matchers. *String based* matchers have two labels as input (with exception of the *Text Corpus* which takes also a text corpus). *Sense based* matchers have two WordNet senses in input. *Gloss based* matchers also have two WordNet senses as input and produce relations exploiting gloss comparison techniques.

4. String based matchers

Approximate string matching techniques [10] are widely used in various schema matching systems [5, 13]. Our *String based* matchers are modifications of well known element level syntactic matchers, and produce an equivalence relation if the input labels satisfy the given criteria, which are specific for each matcher. Otherwise, *Idk* is returned.

4.1 Prefix

Prefix checks whether one input label starts with the other. It returns an equivalence relation in this case, and *Idk* otherwise. The examples of relations *Prefix* produced and the time it needs to compute them are summarized in Table 2.

Prefix is efficient in matching cognate words and similar acronyms (e.g., *RDF* and *RDFS*) but often syntactic similarity does not imply semantic relatedness. Consider the examples in Table 2. The matcher returns equality for *hot* and *hotel* which is wrong but it recognizes the right relations in the case of the pairs *net*, *network* and *cat*, *core*.

Table 2. The relations produced by the prefix matcher and its execution time

Source label	Target label	Relation	Time, ms
net	*network*	=	0.00006
hot	*hotel*	=	0.00006
cat	*core*	*Idk*	0.00005

4.2 Suffix

Suffix checks whether one input label ends with the other. It returns the equivalence relation in this case and *Idk* otherwise. The results produced and the time needed are summarized in Table 3.

Table 3. The relations produced by the suffix matcher and its execution time

Source label	Target label	Relation	Time, ms
phone	*telephone*	=	0.00007
word	*sword*	=	0.00007
door	*floor*	*Idk*	0.00005

Suffix performs very similarly to *Prefix*. It correctly recognizes cognate words (*phone*, *telephone*) but makes mistakes with syntactically similar but semantically different words (*word*, *sword*).

4.3 Edit Distance

Edit distance calculates the edit distance measure between two labels. The calculation includes counting the number of the simple editing operations (delete, insert and replace) needed to convert one label into another and dividing the obtained number of operations with *max(length(label1),length(label2))*. The result is a value in [0..1]. If the value exceeds a given threshold (0.6 by default) the equivalence relation is returned, otherwise, *Idk* is produced.

Table 4. The relations produced by the edit distance matcher and its execution time

Source label	Target label	Relation	Time, ms
street	*street1*	=	0.019
proper	*propel*	=	0.016
owe	*woe*	*Idk*	0.007

Edit Distance is useful with some unknowns to WordNet labels. For example, it can easily match labels *street1, street2, street3, street4* to *street* (edit distance measure is 0.86). In the case of matching *proper* with *propel* the edit distance similarity measure has 0.83 value, but equivalence is obviously the wrong output.

4.4 NGram

NGram counts the number of the same ngrams (e. g., sequences of *n* characters) in the input labels. For example, trigrams for the word *address* are *add, ddr, dre, res, ess*. If the value exceeds a given threshold the equivalence relation is returned. Otherwise *Idk* is produced. The relations produced by *NGram* and the time needed to calculate them are summarized in Table 5.

Table 5. The relations produced by the ngram matcher and its execution time

Source label	Target label	Relation	Time, ms
address	*address1*	=	0.008
behavior	*behaviour*	=	0.011
door	*floor*	*Idk*	0.011

In the past, *Ngram* and *Edit Distance* have been applied to matching database and XML schemas [5, 13]. Both matchers are effective in matching acronyms and words with minor syntactic differences like (*address*, *address1*).

Table 6 compares the results of *Edit distance* and *Ngram*. The first column contains the matcher names. Input labels are reported in the second and the third column. The relations returned by matchers are in the fourth column. The values of similarity measures are in the last column. For all tests the threshold value was 0.6 (this is default value for the *string based* matchers).

Table 6. Edit distance and Ngram results comparison

Matcher name	Source label	Target label	Relation	Similarity value
Edit distance	*fin*	*find*	=	0.75
Ngram	*fin*	*find*	*Idk*	0.54
Edit distance	*PO*	*PO1*	=	0.66
Ngram	*PO*	*PO1*	*Idk*	0.5
Edit distance	*impact*	*compact*	=	0.71
Ngram	*impact*	*compact*	*Idk*	0.58

Edit distance behaves on the most examples more optimistically than *Ngram*. It means that *Edit distance* returns more false positives than true negatives. In some cases (for example, matching *PO* with *PO1*) it can be useful. In other cases (matching *fin* and *find*, *impact* and *compact*) *Ngram* performs better.

Our intuition (confirmed by the preliminary testing results) is that *string based* matchers are better suited for XML/relational schema matching. They work better with acronyms and abbreviations, which are quite common in this domain. See also [5, 13].

4.5 Text Corpus

Corpus based matchers exploit natural language processing and sense disambiguation techniques [2, 3, 12, 16]. However, with the noticeable exception of [15], they have never been used in the schema matching/ontology alignment context.

Corpus based matchers find occurrences of the first label in the second label immediate vicinity (or text window, whose size typically varies from a few to several thousand characters) in a corpus. *Text Corpus* has in input two labels and a text corpus and produces a relation between the labels. If a sufficient number of labels co-occurrences is found, then *Text Corpus* returns the equivalence relation. With these matchers the major problem is the choice of the right corpus. For example, using the Genesis, from The King James Holy Bible as corpus we can infer (if we have the text window of size at least 3 words ahead) that *darkness* is related to *night* because of the following sentence.

5: *And God called the light Day, and the darkness he called Night. And the evening and the morning were the first day.*

But using the same example (and window size of 4 words ahead) we can infer that *God* is related to *Day* which is wrong. Table 7 illustrates the example.

Table 7. The relations produced by the text corpus matcher and its execution time

Source label	Target label	Relation	Time, ms
Night	*darkness*	=	0.008
God	*Day*	=	0.008
light	*first*	*Idk*	0.009

At the moment we use a very simple version of this matcher, which calculates the label co-occurrences within a given text window in a given corpus. If these co-occurrences exceed a given threshold (at the moment this value depends on the corpus size) the equivalence relation is produced. Otherwise, the matcher returns *Idk*. A possible solution for the "right" corpus selection strategy is to query Internet search engines as sources of relevant corpuses [15].

5. Sense based matchers

Sense based matchers take in input two WordNet senses and exploit the structural properties of WordNet hierarchies.

WordNet [14] is a lexical database which is available online [20] and provides a large repository of English lexical items. WordNet contains synsets (or senses), structures containing sets of terms with synonymous meanings. Each synset has a gloss that defines the concept that it represents. For example the words *night*, *nighttime* and

dark constitute a single synset that has the following gloss: *the time after sunset and before sunrise while it is dark outside*. Synsets are connected to one another through explicit semantic relations. Some of these relations (hypernymy, hyponymy for nouns and hypernymy and troponymy for verbs) constitute *kind-of* and *part-of* (holonymy and meronymy for nouns) hierarchies. In example, *tree* is a kind of *plant*, *tree* is hyponym of *plant* and *plant* is hypernym of *tree*. Analogously from *trunk* is a part of *tree* we have that *trunk* is meronym of *tree* and *tree* is holonym of *trunk*.

We translate the relations provided by WordNet to semantic relations according to the following rules:

- A $\subseteq$ B if A is a hyponym, meronym or troponym of B;
- A $\supseteq$ B if A is a hypernym or holonym of B;
- A = B if they are connected by synonymy relation or they belong to one synset (*night* and *nighttime* from abovementioned example);
- A $\perp$ B if they are connected by antonymy relation or they are the siblings in the *part of* hierarchy

Further, we use the notion of extended gloss [3]. An extended gloss is a text corpus obtained by concatenating the glosses of synsets known to be related, via a WordNet hierarchy, with a given WordNet synset. For example, two extended glosses can be built for the concept *tree*. The first consists the glosses of the less general synsets (*trunk*, *oak*, etc). The second is obtained from the glosses of the more general synsets (*plant*, *object*, etc.).

5.1 WordNet

WordNet is an exact element level semantic matcher. It provides if it exists, a relation between two input senses and *Idk* otherwise. For example, *car*, according to WordNet, is more general than *minivan*. Thus, we return $\supseteq$ relation (See Table 8). On the other hand, *red* and *pink* are not connected by any of WordNet relations and we return *Idk*.

Table 8. The relations produced by WordNet matcher and its execution time

Source label	Target label	Relation	Time, ms
car	*minivan*	$\supseteq$	2,3
car	*auto*	=	0.6
tail	*dog*	$\subseteq$	0.2
red	*pink*	*Idk*	0.4

The results depend heavily on the content of WordNet. Our work with this matcher is basically a reimplementation of the work described in [4]. An interesting extension of this work is the possibility of extending WordNet with domain specific information.

5.2 Hierarchy distance

Hierarchy based matchers measure the distance between two concepts in a given input hierarchy. Several semantic word relatedness measures have been proposed. See, for instance [1, 8, 17, 19]. At the moment we use a very simple hierarchy distance measure, which is a slight modification of the method used in [17]. In particular, *Hierarchy distance* returns the equivalence relation if the distance between two input senses in a WordNet hierarchy is less than a given threshold value and *Idk* otherwise. The number of less general and more general arcs is also considered. In the case of equivalence they must be nearly of the same number.

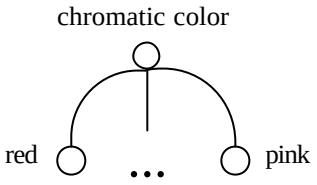

Fig. 3. The immediate vicinity of *red* and *pink* concepts in WordNet is-a hierarchy

Consider the example in Fig. 3.It can be noticed that, there is no direct relation between *red* and *pink*. However, the distance between these concepts is 2 (1 more general link and 1 less general). Thus, we can infer that *red* and *pink* are close in their meaning and return the equivalence relation. On the other hand, synsets of *catalog* and *classification* are not connected through a WordNet hierarchy (e.g., they have different top level ancestors). Thus, *Idk* is returned. Table 9 illustrates these examples.

Table 7. The relations produced by hierarchy distance matcher and its execution time

Source label	Target label	Relation	Time, ms
red	*pink*	=	0.159
catalog	*classification*	*Idk*	0.203

This matcher has several modifications regarding the way the distance between the concepts is calculated. In the current implementation we use a simple hierarchy distance measure. We count the number of arcs in a *is-a* hierarchy and if this value is less than a given threshold the equivalence relation is returned.

Hierarchy distance works relatively fast and provides a good approximation of the concepts similarity. The major drawback of this matcher is the strong dependence on the concepts vicinity structure in WordNet. The hierarchy based word relatedness measures from [19, 9, 1] can also be adapted to our matching application.

6. Gloss based matchers

Gloss based matchers, similarly to *sense based* matchers, have in input two WordNet senses and return the semantic relation holding between them. However, *gloss based*

matchers differ in that they use the information contained in WordNet glosses. Many of them exploit techniques from natural language processing [2, 3, 12, 16].

The majority of *gloss based* matchers use corpus based and corpus comparison techniques. As a result, two questions arise: which corpuses should be compared and how comparison should be performed? Concerning the second problem, at the moment we use two methods.

According to the first method, we calculate the number of occurrences of the same words in two corpuses. If the number exceeds a given threshold the relation is produced. We call this method *syntactic corpus comparison*.

The second method is based on the calculation not only of the number of occurrences, but also of the number of synonyms, and more and less general words between corpuses. We call this method as *semantic corpus comparison*.

Table 10 reports what is compared and how comparison is performed in the *gloss based* matchers implemented so far.

Table 10. Sense based matchers: what is compared and how comparison is performed

Matcher name	What is compared?	Comparison method
WordNet gloss	labels and gloss	syntactic
WordNet extended gloss	labels and extended gloss	syntactic
Gloss comparison	gloss and gloss	syntactic
Extended gloss comparison	gloss and extended gloss	syntactic
Semantic gloss comparison	gloss and gloss	semantic
Extended semantic gloss comparison	gloss and extended gloss	semantic

6.1 WordNet gloss

WordNet gloss compares the labels of the first input sense with the WordNet gloss of the second. First, it extracts the labels of the first input sense from WordNet. Then, it computes the number of their occurrences in the second gloss. If this number exceeds a given threshold, $\subseteq$ is returned. Otherwise, *Idk* is produced.

The reason why the less general relation is returned comes from the lexical structure of the WordNet gloss. Very often the meaning of the index words is explained through a specification of the more general concept. In the following example, *hound* (*any of several breeds of dog used for hunting typically having large drooping ears*) *hound* is described through the specification of the more general concept *dog*. In this example *hound* is a *dog* with special properties (*large drooping ears, used for hunting*).

Counting the label occurrences in the gloss does not give a strong evidence of what relation holds between concepts. For example, *WordNet gloss* returns the less general relation for *hound* and *ear* in the abovementioned example, which is clearly wrong. Table 11 illustrates the example.

Table 11. The relations produced by the WordNet gloss matcher and its execution time

Source label	Target label	Relation	Time, ms
hound	*dog*	$\subseteq$	0.031
hound	*ear*	$\subseteq$	0.014
dog	*cat*	*Idk*	0.033

This matcher implements the ideas developed in [12, 2]. The main difference is that the matcher returns a semantic relation rather than a numerical similarity coefficient.

6.2 WordNet extended gloss

WordNet extended gloss compares the labels of the first input sense with the extended gloss of the second. This extended gloss is obtained from the input sense descendants (ancestors) descriptions in the *is-a* (*part-of*) WordNet hierarchy. A given threshold determines the maximum allowed distance between these descriptions and the input sense in the WordNet hierarchy. By default, only direct descendants (ancestors) are considered.

The idea of using extended gloss originates from [3]. Unlike [3], we do not calculate the *extended gloss overlaps measure*, but count the number of first input sense labels occurrences in the extended gloss of the second input sense. If this number exceeds a given threshold, a semantic relation is produced. Otherwise, *Idk* is returned. The type of relation produced depends on the glosses we use to build the extended gloss. If the extended gloss is built from descendant (ancestor) glosses, then the $\supseteq$ ($\subseteq$) relation is produced.

For example, the relation holding between the words *dog* and *breed* can be easily found by this matcher. These concepts are not related in WordNet, but the word *breed* occurs very often in the *dog's* descendant glosses. Table 12 illustrates the example.

Table 12. The relations produced by the WordNet extended gloss matcher and its execution time

Source label	Target label	Relation	Time, ms
dog	*breed*	$\supseteq$	0.268
dog	*cat*	*Idk*	4.3
wheel	*machinery*	$\subseteq$	2.6

6.3 Gloss comparison

Within *Gloss comparison* the number of the same words occurring in the two input glosses increases the similarity value. The equivalence relation is returned if the resulting similarity value exceeds a given threshold. *Idk* is produced otherwise.

Let us try to find the relation holding, for example, between *Afghan hound* and *Maltese dog* using gloss comparison strategy. These two concepts are breeds of dog, but unfortunately WordNet does not have explicit relation between them. However, the glosses of both concepts are very similar. Let us compare:

Maltese dog is a breed of toy dogs having a long straight silky white coat.
And:
Afghan hound is a tall graceful breed of hound with a long silky coat; native to the Near East.

There are 4 shared words in both glosses (*breed, long, silky, coat*). Hence, the two concepts are taken to be equivalent. Table 13 illustrates the example.

Table 13. The relations produced by the gloss comparison matcher and its execution time

Source label	Target label	Relation	Time, ms
Afghan hound	*Maltese dog*	=	0,074
dog	*cat*	*Idk*	0,019

Several modifications of this matcher exist. One can assign a higher weight to the phrases or particular parts of speech than single words [16]. In the current implementation we have exploited the approach used in [16], but changed the output to be a semantic relation.

6.4 Extended Gloss comparison

Extended gloss comparison compares two extended glosses built from the input senses. Thus, if the first gloss has a lot of words in common with descendant glosses of the second then the first sense is more general than the second and vice versa. If the corpuses (extended glosses) formed from descendant (ancestor) glosses of both labels have a lot of words in common (this value is controlled by a given threshold) then the equivalence relation is returned.

For example, *dog* and *cat* are not connected by any relation in WordNet. Comparing the corpuses obtained from descendants glosses of both concepts we can find a lot of words in common (*breed, coat*, etc). Thus, we can infer that *dog* and *cat* are related (they are both pets), and return the equivalence relation. The relations produced by the matcher and its execution time are summarized in Table 14.

Table 14. The relations produced by the extended gloss comparison matcher and its execution time

Source label	Target label	Relation	Time, ms
dog	*cat*	=	4.3
house	*animal*	*Idk*	79.8

The idea of this matcher originates from the *extended gloss overlaps measure*, as described in [3]. Unlike [3], we do not calculate the extended gloss overlaps measure, but return semantic relations produced by the rules stated above.

6.5 Semantic Gloss comparison

Semantic Gloss comparison is based on a new method for determining semantic words relatedness. This method extends the work of [16]. The key idea is to maintain statistics not only for the same words in the input senses glosses (like in *Gloss comparison*) but also for words which are connected through *is-a* (*part-of*) relationships in WordNet. This can help finding the gloss relevance not only at the syntactic but also

at the semantic level. In *Semantic Gloss Comparison* we consider synonyms, less general and more general concepts which (hopefully) lead to better results.

In the first step the glosses of both senses are obtained. Then, they are compared by checking which relations hold in WordNet between the words of both glosses. If there is a sufficient amount (in the current implementation this value is controlled by a threshold) of synonyms the equivalence relation is returned. In the case of a large amount of more (less) general words, the output is $\supseteq$ ($\subseteq$) correspondingly. *Idk* is returned if we have a nearly equal amount of more and less general words in the glosses or there are no relations between words in glosses. Table 15 contains the results produced by semantic gloss comparison matcher.

Table 15. The relations produced by the semantic gloss comparison matcher and its execution time

Source label	Target label	Relation	Time, ms
dog	*breed*	$\supseteq$	6.7
cat	*dog*	*Idk*	10.9
wheel	*machinery*	$\subseteq$	1419

6.6 Extended Semantic Gloss comparison

This matcher is based on a new method for determining semantic relatedness between words. The key idea is to count not only the same words in the corpuses (built from the gloss of the first input sense and extended gloss of the second) but also the number of words connected by $=$, $\subseteq$, $\supseteq$. This setting allows us to combine the features of *Extended gloss comparison* (considering extended glosses) and *Semantic gloss comparison* (considering not only the same words, but also synonyms, less and more general words).

In the first step, we obtain the gloss of the first sense and the extended glosses (for less and more general relations) of the second. Afterwards, we calculate the number of more, less general and synonym words. The output relation is produced following the rules depicted in Table 16. Relations exploited to build extended gloss are depicted in the columns. Rows determine the major number of relations between the words of both corpuses. Table 17 illustrates the matcher results.

Table 16. Output of extended semantic gloss comparison matcher depending from the relations used to build the corpuses and the major number of relations between words in the corpuses

Major number of relations between words in the corpuses	Extended gloss are built from glosses of	
	less general words	more general words
$=$	$\subseteq$	$\supseteq$
$\subseteq$	$\subseteq$	*Idk*
$\supseteq$	*Idk*	$\supseteq$

Table 17. The results of extended semantic gloss comparison matcher and its execution time

Source label	Target label	Relation	Time, ms
car	*train*	$\subseteq$	43047
red	*pink*	$\supseteq$	5157
dog	*breed*	*Idk*	0.28

The pseudo-code of this matcher algorithm is depicted in Figure 4.

```
1.String ExtSemGlossComp(String sSense, String tSense)
2. String[] Rels={"<",">"};
3. String sCorpus=getGloss(sSense);
4. for each Rel in Rels
5.  String tCorpus=BuildCorpus(tSense, Rel);
6.  String corpusRel =compareCorpus(sCorpus,tCorpus);
7.  if (corpusRel!="Idk")
8.   return MakeRelation (corpusRel, Rel);
```

Fig. 4. Pseudo code of extended semantic gloss comparison matcher

The main routine of the matcher **extSemGlossComp** takes two WordNet senses (*sSense*, *tSense*) as input and produces the relation between them as output. First, it takes the gloss of the source sense (line 3). Afterwards, the routine loops on semantic relations (more and less generality) for which we build an extended gloss (lines 4-8). Within the loop the system builds the appropriate extended gloss (line 5) and then compares it with the gloss we built on the line 3 (line 5). Finally, it produces an output relation (line 8).

CompareCorpus takes two corpuses (*sCorpus, tCorpus*) and compares them by calculating the number of synonyms, and more and less general links between words in the corpuses. In order to perform this, it loops on all words in both corpuses and obtains (for each possible pair) the relation from WordNet. Afterwards, the statistics of more, less general and synonym word occurrences in the corpuses is calculated (lines 15-18). Finally, **MakeRelation** is executed in order to obtain the semantic relation holding between two corpuses. **BuildCorpus** takes the WordNet sense and a semantic relation as input. This routine obtains the right (with respect to a particular relation) extended gloss from WordNet by concatenating the appropriate (more general or less general to the input sense) glosses. **ChooseRelation** take in input the number of less general, more general and synonym words in two corpuses and produces a semantic relation. It employs several heuristics in order to choose the right relation from the input numbers. By default, the relation with maximum number of occurrences in the corpuses is returned. **MakeRelation** takes two semantic relations as input (the first is used to obtain extended gloss, the second is obtained from corpuses comparison) and uses rules depicted in Table 16 to produce an output relation.

Especially interesting seems the application of the gloss comparison techniques to the matching of big ontologies equipped with descriptions of concepts. In this case gloss comparison methods can provide more useful element level mappings than traditional element level approaches.

7. Conclusion

In this paper we have presented the library of element level semantic matchers implemented within the S-Match system. We have implemented three kinds of matchers: *string*, *sense* and *gloss based*.

A lot of work still needs to be done in order to identify their performance and how they can be successfully exploited in the many, very diverse, matching tasks.

References

1. E. Agirre and G. Rigau. Word sense disambiguation using conceptual density. In Proceedings of the 16th International Conference on Computational Linguistics, Copenhagen, 1996.
2. S. Banerjee and T. Pedersen. An adapted Lesk algorithm for word sense disambiguation using Word-Net. In Proceedings of the 3rd International Conference on Intelligent Text Processing and Computational Linguistics, Mexico City, 2002.
3. S. Banerjee and T. Pedersen. Extended gloss overlaps as a measure of semantic relatedness. In Proceedings of the 18th International Conference on Artificial Intelligence (IJCAI-03), Acapulco, Mexico, 2003.
4. P. Bouquet, L. Serafini, S. Zanobini. Semantic Coordination: A new approach and an application. In Proceedings of ISWC 2003.
5. H. Do, E. Rahm. COMA - A system for Flexible Combination of Schema Matching Approaches, In Proceedings of VLDB 2002
6. F. Giunchiglia, P. Shvaiko. Semantic Matching. In The Knowledge Engineering Review Journal, 18(3) 2004.
7. F. Giunchiglia, P. Shvaiko, M. Yatskevich. S-Match: An algorithm and an implementation of semantic matching. In Proceedings of ESWS'04.
8. F. Giunchiglia, I. Zaihrayeu. Making peer databases interact - a vision for an architecture supporting data coordination. In Proceedings of the Conference on Information Agents, Madrid, September (2002)
9. J. Jiang and D. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In Proceedings on International Conference on Research in Computational Linguistics, Taiwan, 1997.
10. P. Hall, G. Dowling. Approximate String Matching. In the Computing Survey 12: 4, 1980
11. M. Hearst. "Automated Discovery of WordNet Relations" in WordNet: An Electronic Lexical Database, Christiane Fellbaum (ed.), MIT Press, 1998.
12. M. Lesk. Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from a ice cream cone. In Proceedings of SIGDOC 1986.
13. J. Madhavan, P.A. Bernstein, E. Rahm. Generic Schema Matching with Cupid. VLDB 2001
14. A. Miller. Wordnet: A lexical database for english. In Communications of the ACM, number 38(11) 1995.
15. P. Mitra, G. Wiederhold. Resolving Terminological Heterogeneity in Ontologies Proceedings ECAI-02 Workshop on Ontologies and Semantic Interoperability, 2002
16. S. Patwardhan, S. Banerjee, and T. Pedersen. Using measures of semantic relatedness for word sense disambiguation. In Proceedings of the 4th International Conference on Intelligent Text Processing and Computational Linguistics (CICLING-03), Mexico City, 2003.
17. R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. In IEEE Transactions on Systems, Man and Cybernetics, 19(1), 1989.
18. E. Rahm, P.A. Bernstein. A Survey of Approaches to Automatic Schema Matching. In VLDB Journal 10: 4, 2001

19. P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal, August 1995.
20. WordNet a lexical database for the English language. http://www.cogsci.princeton.edu/~wn