SEMANTIC MATCHING WITH S-MATCH

Pavel Shvaiko, Fausto Giunchiglia, and Mikalai Yatskevich

May 2009

Technical Report # DISI-09-030

Also: to appear in "Semantic Web Information Management: a model based perspective", Roberto De Virgilio, Fausto Giunchiglia, and Letizia Tanca (eds.), Springer, 2009

Chapter 1
Semantic matching with S-Match*

Pavel Shvaiko, Fausto Giunchiglia, and Mikalai Yatskevich

Abstract

We view *matching* as an operation that takes two graph-like structures (e.g., lightweight ontologies) and produces an alignment between the nodes of these graphs that correspond semantically to each other. *Semantic matching* is based on two ideas: (*i*) we discover an alignment by computing *semantic relations* (e.g., equivalence, more general); (*ii*) we determine semantic relations by analyzing the *meaning* (concepts, not labels) which is codified in the entities and the structures of ontologies. In this chapter we first overview the state of the art in the ontology matching field. Then, we present basic and optimized algorithms for semantic matching as well as their implementation within the S-Match system. Finally, we evaluate S-Match against state of the art systems, thereby justifying empirically the strength of the approach.

1.1 Introduction

Matching is an important operation in many applications, such as ontology integration, data warehouses, peer-to-peer data sharing, etc. The matching operation takes two graph-like structures (e.g., lightweight ontologies [14]) and produces an alignment, that is a set of mapping elements (or correspondences), between the nodes of the graphs that correspond semantically to each other.

There exist various solutions of matching, see [7, 37, 40, 41] for recent surveys. In turn, some recent examples of individual matching approaches can be

Pavel Shvaiko
TasLab, Informatica Trentina S.p.A., Trento, Italy
e-mail: `pavel.shvaiko@infotn.it`

Fausto Giunchiglia
DISI, University of Trento, Trento, Italy
e-mail: `fausto@dit.unitn.it`

Mikalai Yatskevich
Faculté de Médecine, Université Rennes 1, Rennes, France
e-mail: `mikalai.yatskevich@univ-rennes1.fr`

* The first and the third authors performed this work while at the University of Trento.

found in [2, 3, 6, 8, 10–12, 23, 27, 32, 38][1]. We concentrate on a schema-based solution, i.e., a matching system that exploits only the schema information and does not consider the instance information. We follow an approach called *semantic matching* [15]. This approach is based on two key ideas. The first is that we calculate correspondences between ontology entities by computing *semantic relations* (e.g., equivalence, more general, disjointness), instead of computing coefficients rating match quality in the [0,1] range, as it is the case in most previous approaches, see, for example, [6, 10, 30, 33]. The second idea is that we determine semantic relations by analyzing the *meaning* (concepts, not labels) which is codified in the entities and the structures of ontologies. In particular, labels at nodes, written in natural language, are automatically translated into propositional formulas which explicitly codify the labels' intended meaning. This allows us to translate the matching problem into a propositional validity problem, which can then be efficiently resolved using (sound and complete) state of the art propositional satisfiability (SAT) deciders, e.g., [28].

A vision for the semantic matching approach and some of its implementation within S-Match were reported in [15–17, 19, 22]. In turn, the works in [18, 21, 23, 43] focused on the following aspects of S-Match: (*i*) algorithms and implementation, (*ii*) discovering missing background knowledge in matching tasks, (*iii*) explanation of matching results, and (*iv*) large-scale evaluation. This chapter builds on top of the above mentioned works and provides a summative account for the semantic matching approach (hence, the key algorithms are identical to those in the above mentioned works).

The rest of the chapter is organized as follows. Section 1.2 overviews the state of the art in the ontology matching field. Section 1.3 presents the semantic matching approach. Section 1.4 introduces the optimizations that allow improving efficiency of the basic version of Section 1.3. Section 1.5 outlines the S-Match architecture. The evaluation results are presented in Section 1.6. Finally, Section 1.7 provides conclusions and discusses future work.

1.2 State of the art

A good survey and a classification of ontology[2] matching approaches up to 2001 was provided in [40], a semantics driven extension of its schema-based part and a user-centric classification of matching systems was provided in [41], while the work in [9] considered both [40, 41] as well as some other classifications.

[1] See `www.OntologyMatching.org` for a complete information on the topic.

[2] An ontology typically provides a vocabulary that describes a domain of interest and a specification of the meaning of terms used in the vocabulary. Depending on the precision of this specification, the notion of ontology includes various data and conceptual models [9]. The term ontology is used here in a wide sense, and, hence, encompasses, e.g., sets of terms, classifications, database schemas, thesauri, or fully axiomatized theories.

In particular, for individual matchers, [41] introduced the following criteria which allow for detailing further the element and structure level of matching: *syntactic* techniques (these interpret their input as a function of their sole structures following some clearly stated algorithms, e.g., iterative fix point computation for matching graphs), *external* techniques (these exploit external resources of knowledge, e.g., WordNet [35]), and *semantic* techniques (these use formal semantics, e.g., model-theoretic semantics) in order to interpret the input and justify their results.

The distinction between the hybrid and composite matching algorithms of [40] is useful from an architectural perspective. The work in [41] extended this view by taking into account how the systems can be distinguished in the matter of considering the alignments and the matching task, thus representing the end-user perspective. In this respect, the following criteria were used: *alignments as solutions* (these systems consider the matching problem as an optimization problem and the alignment is a solution to it, e.g., [10, 33]); *alignments as theorems* (these systems rely on semantics and require the alignment to satisfy it, e.g., the approach discussed in this chapter); *alignments as likeness clues* (these systems produce only reasonable indications to a user for selecting the alignment, e.g., [5, 30]).

So far there have been developed more than 50 various matching approaches, such as Harmony [36], Falcon [26], RiMOM [29], Sambo [27], to name a few, see [9] for the detailed comparison of the state of the art systems. Here, we only consider the closest to S-Match schema-based systems in light of the above mentioned criteria.

Rondo implements the Similarity Flooding (SF) [33] approach, and utilizes a hybrid matching algorithm based on the ideas of similarity propagation [34]. Schemas are presented as directed labeled graphs. The algorithm exploits only syntactic techniques at the element and structure level. It starts from the string-based comparison, such as common prefixes and suffixes tests, of the nodes' labels to obtain an initial alignment which is further refined within the fix-point computation. Rondo considers the alignments as a solution to a clearly stated optimization problem.

Cupid implements a hybrid matching algorithm comprising syntactic techniques at the element (e.g., common prefixes test) and structure (e.g., tree matching weighted by leaves) levels [30]. It also exploits external resources, such as a precompiled thesaurus. Cupid falls into the alignments as likeness clues category.

COMA implements a composite matching approach which exploits syntactic and external techniques [5, 6]. It provides a library of matching algorithms; a framework for combining obtained results, and a platform for the evaluation of the effectiveness of the different matchers. The matching library is extensible and contains six elementary matchers, five hybrid matchers and one reuse-oriented matcher. Most of them implement string-based techniques, such as n-gram and edit distance; others share techniques with Cupid, such as tree matching weighted by leaves; reuse-oriented matcher tries to reuse previously obtained results for entire new ontologies or for their fragments. Specialties of COMA with respect to Cupid include a more flexible architecture and a possibility of performing iterations in the matching process. COMA falls into the alignments as likeness clues category.

Semantic heterogeneity is typically reduced in two steps. We have focused so far only on the first step, i.e., on establishing an alignment between semantically related entities of ontologies. The second step includes interpreting an alignment according to application needs, such as data translation or query answering. Here, alignments are taken as input and are analyzed in order to generate, e.g., query expressions, that automatically translate/exchange data instances between the information sources. Notice that taking as input semantic relations, instead of coefficients in the [0,1] range, enables, e.g., data translation systems, to produce better results, since, e.g., in such systems as Clio [25], the first step is to interpret the correspondences by giving them a clear semantics.

1.3 Semantic Matching

We assume that all the data and conceptual models (e.g., lightweight ontologies [14]) can be generally represented as graphs (see [15] for a detailed discussion). This allows for the statement and solution of a *generic (semantic) matching problem* independently of specific conceptual or data models, very much along the lines of what is done in Cupid [30] and COMA [5]. We focus on tree-like structures or such types of ontologies as classifications and XML schemas. Real-world schemas are seldom trees, however, there are (optimized) techniques, transforming a graph representation of a schema into a tree representation, e.g., the graph-to-tree operator of Protoplasm [1]. From now on we assume that a graph-to-tree transformation can be done by using existing systems, and therefore, we focus on other issues instead.

The semantic matching approach is based on two key notions, namely:

- *Concept of a label*, which denotes the set of documents (data instances) that one would classify under a *label* it encodes;
- *Concept at a node*, which denotes the set of documents (data instances) that one would classify under a node, given that it has a certain *label* and that it is in a certain *position* in a tree.

Our approach can discover the following semantic relations between the *concepts at nodes* of two ontologies: *equivalence* ($\equiv$); *more general* ($\sqsupseteq$); *less general* ($\sqsubseteq$); *disjointness* ($\perp$). When none of the relations holds, the special *idk* (i do not know) relation is returned[3]. The relations are ordered according

[3] Notice *idk* is an explicit statement that the system is unable to compute any of the declared (four) relations. This should be interpreted as either there is not enough background knowledge,

to decreasing binding strength, i.e., from the strongest ($\equiv$) to the weakest (idk), with more general and less general relations having equal binding power. Notice that the strongest semantic relation always exists since, when holding together, more general and less general relations are equivalent to equivalence. These relations have the obvious set-theoretic semantics.

A *mapping element* is a 4-tuple $\langle ID_{ij}, a_i, b_j, R\rangle$, i =1,...,N_A; j =1,...,N_B; ID_{ij} is a unique identifier of the given mapping element; a_i is the i-th node of the first tree, N_A is the number of nodes in the first tree; b_j is the j-th node of the second tree, N_B is the number of nodes in the second tree; and R specifies a semantic relation which may hold between the concepts at nodes a_i and b_j. *Semantic matching* can then be defined as the following problem: given two trees T_A and T_B, compute the $N_A \times N_B$ mapping elements $\langle ID_{ij}, a_i, b_j, R'\rangle$, with $a_i \in T_A$, i=1,..., N_A; $b_j \in T_B$, j =1,..., N_B; and R' is the strongest semantic relation holding between the concepts at nodes a_i and b_j. Since we look for the $N_A \times N_B$ correspondences, the cardinality of mapping elements we are able to determine is 1:N. Also, these, if necessary, can be decomposed straightforwardly into mapping elements with the 1:1 cardinality.

1.3.1 The Tree Matching Algorithm

We discuss the semantic matching algorithm with the help of an example, see Figure 1.1. Here, numbers are the unique identifiers of nodes. We use "C" for concepts of labels and concepts at nodes. For instance, in the tree A, $C_{History}$ and C_4 are, respectively, the concept of the label $History$ and the concept at node 4. To simplify the presentation, whenever it is clear from the context we assume that the concept of a label can be represented by the label itself. In this case, for example, $C_{History}$ becomes denoted as $History$. Finally, we sometimes use subscripts to distinguish between trees in which the given concept of a label occurs. For instance, $History_A$, means that the concept of the label $History$ belongs to the tree A.

The algorithm discussed below was first published in [16] and later updated in [17, 23]. It takes as input two ontologies and computes as output a set of mapping elements in four macro steps:

- *Step 1*: for all labels L in two trees, compute concepts of labels, C_L.
- *Step 2*: for all nodes N in two trees, compute concepts at nodes, C_N.
- *Step 3*: for all pairs of labels in two trees, compute relations among C_L's.
- *Step 4*: for all pairs of nodes in two trees, compute relations among C_N's.

The first two steps represent the preprocessing phase, while the third and the fourth steps are the element level and structure level matching respectively. It is important to notice that *Step 1* and *Step 2* can be done once,

and therefore, the system cannot explicitly compute any of the declared relations or, indeed, none of those relations hold according to an application.

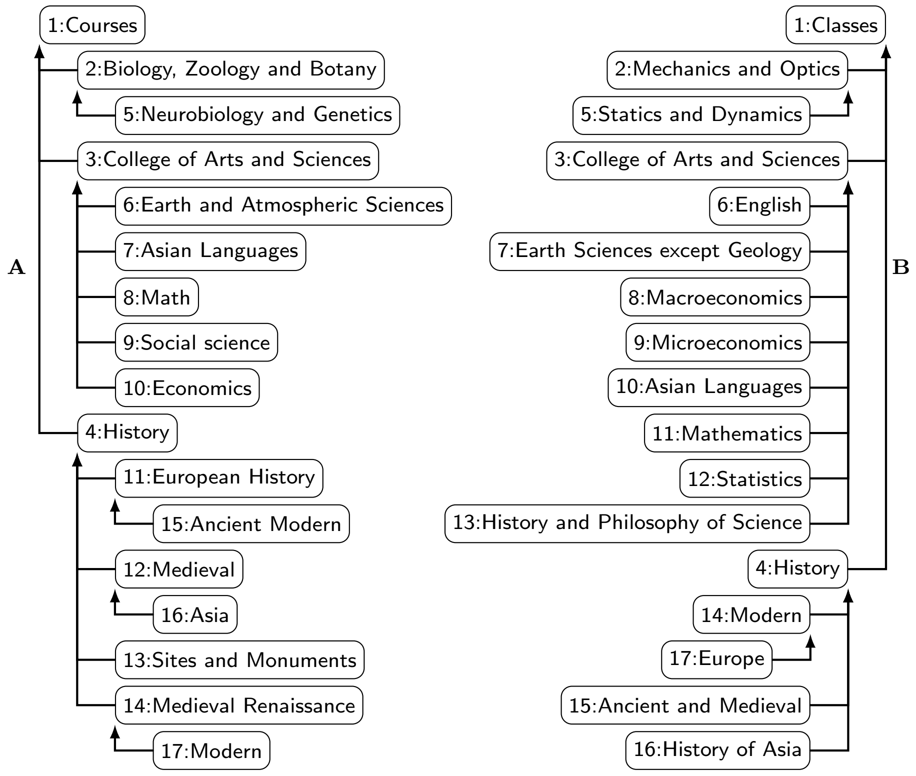


Fig. 1.1: Fragments of two classifications devoted to academic courses.

independently of the specific matching problem. *Step 3* and *Step 4* can only be done at run time, once two trees which must be matched have been chosen.

Step 1. For all labels L in two trees, compute *concepts of labels*. We view labels of nodes as concise descriptions of the data that is stored under the nodes. Here, we compute the meaning of a label at a node (in isolation) by taking as input a *label*, by analyzing its real-world semantics (e.g., using WordNet [35]), and by returning as output a *concept of the label*. For example, by writing $C_{History}$ we move from the natural language label $History$ to the concept $C_{History}$, which codifies explicitly its intended meaning, namely the data (documents) which are about history.

From a technical viewpoint, the main goal of *Step 1* is to automatically translate ambiguous natural language labels taken from the entities' names into an internal logical language, which is a propositional description logic language. First, we chunk labels into tokens, e.g., *Earth and Atmospheric Sciences* becomes $\langle earth\ sciences, and, atmospheric, sciences\rangle$. Tokens of labels are further lemmatized, namely they are morphologically analyzed in order to find all their possible basic forms. For instance, *sciences* is associated with its singular form, *science*. WordNet is queried to obtain the senses of lemmas identified during the previous phase. For example, the label *sciences* has the only one token *sciences*, and one lemma *science*. From WordNet we find out that *science* has two senses as a noun. Atomic formulas are WordNet *senses* of lemmas obtained from single words (e.g., science) or multiwords (e.g., earth sciences). Complex formulas are built by combining atomic formulas using the connectives of set theory. For example, the concept of label *History and Philosophy of Science* is computed as $C_{History\ and\ Philosophy\ of\ Science} = (C_{History} \sqcup C_{Philosophy}) \sqcap C_{Science}$, where $C_{Science} = \langle science, \{senses_{WN\#2}\}\rangle$ is taken to be the union of two WordNet senses, and similarly for *history* and *philosophy*. Notice that natural language *and* is converted into logical disjunction, rather than into conjunction (see [31] for detailed discussion and justification for this choice).

Step 2. For all nodes *N* in two trees, compute *concepts of nodes*. Here, we analyze the meaning of the *positions* that the labels of nodes have in a tree. By doing this we *extend* concepts of labels to *concepts at nodes*. This is required to capture the knowledge residing in the structure of a tree, namely the *context* in which the given concept of label occurs [13]. For example, in the tree A, when we write C_4 we mean the concept describing all the documents of the (academic) courses, which are about history.

From a technical viewpoint, concepts of nodes are written in the same propositional logical language as concepts of labels. Classifications and XML schemas are hierarchical structures where the path from the root to a node uniquely identifies that node and also its meaning. Thus, following an access criterion semantics [24], the logical formula for a concept at node is defined as a conjunction of concepts of labels located in the path from the given node to the root. For example, in the tree A, the concept at node four is computed as follows: $C_4 = C_{Courses} \sqcap C_{History}$.

Step 3: For all pairs of labels in two trees, compute *relations* among *concepts of labels* (label matching). Relations between concepts of labels are computed with the help of a library of element level semantic matchers [20]. These matchers take as input two concepts of labels and produce as output a semantic relation (e.g., equivalence, more/less general) between them. Some of them are reimplementations of the well-known matchers used in Cupid [30] and COMA [5]. The most important difference is that our matchers ultimately return a semantic relation, rather than an affinity level in the [0,1] range, although sometimes using customizable thresholds.

The label matchers are briefly summarized in Table 1.1. 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 matchers' approximation level. The relations produced by a matcher with the first approximation level are always correct. For example, *name* $\sqsupseteq$ *brand* as returned by the *WordNet* matcher. In fact, according to WordNet *name* is a hypernym (superordinate word) of *brand*. Notice that *name* has 15 senses and *brand* has 9 senses in WordNet. We use some sense filtering techniques to discard the irrelevant senses [23]. The relations produced by a matcher with the sec-

Table 1.1: Element level semantic matchers [17, 23].

Matcher name	**Execution order**	**Approximation level**	**Matcher type**	**Schema info**
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
Text corpus	13	3	String-based	Labels + corpus
WordNet	1	1	Sense-based	WordNet senses
Hierarchy distance	6	3	Sense-based	WordNet senses
WordNet gloss	7	3	Gloss-based	WordNet senses
Extended WordNet gloss	8	3	Gloss-based	WordNet senses
Gloss comparison	9	3	Gloss-based	WordNet senses
Extended gloss comparison	10	3	Gloss-based	WordNet senses
Semantic gloss comparison	11	3	Gloss-based	WordNet senses
Extended semantic gloss comparison	12	3	Gloss-based	WordNet senses

ond approximation level are likely to be correct (e.g., *net* ≡ *network*, but *hot* ≡ *hotel* by *Prefix*). The relations produced by a matcher with the third approximation level depend heavily on the context of the matching task (e.g., *cat* ≡ *dog* by *Extended gloss comparison* in the sense that they are both pets). Notice that by default matchers are executed following the order of increasing approximation level. The fourth column reports the matchers' type. The fifth column describes the matchers' input.

We have three main categories of matchers. *String-based* matchers have two labels as input (with exception of *Text corpus* which takes in input also a text corpus). These compute only equivalence relations (e.g., equivalence holds if the weighted *distance* between the input strings is lower than an empirically established threshold). *Sense-based* matchers have two WordNet senses in input. The *WordNet* matcher computes equivalence, more/less general, and disjointness relations; while *Hierarchy distance* computes only the equivalence relation. *Gloss-based* matchers also have two WordNet senses as input, however they exploit techniques based on comparison of textual definitions (*glosses*) of the words whose senses are taken in input. These compute, depending on a particular matcher, the equivalence, more/less general relations, see for details [18, 20].

The result of *Step 3* is a matrix of relations holding between atomic concepts of labels; Table 1.2 shows a part of it for the example of Figure 1.1.

Table 1.2: Matrix of semantic relations holding between concepts of labels (*ClabsMatrix*).

B / A	*Classes*	*History*	*Modern*	*Europe*
Courses	=	*idk*	*idk*	*idk*
History	*idk*	=	*idk*	*idk*
Medieval	*idk*	*idk*	⊥	*idk*
Asia	*idk*	*idk*	*idk*	⊥

Step 4: For all pairs of nodes in two trees, compute *relations* among *concepts of nodes* (node matching). Here, we initially reformulate the tree matching problem into a set of node matching problems (one problem for each pair of nodes). Then, we translate each node matching problem into a propositional validity problem.

The tree matching algorithm is concerned with the decomposition of the tree matching task into a set of node matching tasks. It takes as input two preprocessed trees obtained as a result of *Step 1*, *Step 2* and a matrix of semantic relations holding between the atomic concepts of labels in both trees obtained as a result of *Step 3*. It produces as output the matrix of semantic relations holding between concepts at nodes in both trees. The pseudo code in Algorithm 1 illustrates the tree matching algorithm.

Algorithm 1 The pseudo code of the tree matching algorithm [23].

```
900.   String[ ][ ] treeMatch(Tree of Nodes source, target, String[ ][ ] cLabsMatrix)
910.     Node sourceNode, targetNode;
920.     String[ ][ ] cNodesMatrix, relMatrix;
930.     String axioms, $context_A$, $context_B$;
940.     int i, j;
960.     for each sourceNode $\in$ source
970.       i = getNodeId(sourceNode);
980.       $context_A$ = getCnodeFormula(sourceNode);
990.       for each targetNode $\in$ target
1000.        j = getNodeId(targetNode);
1010.        $context_B$ = getCnodeFormula(targetNode);
1020.        relMatrix = extractRelMatrix(cLabsMatrix, sourceNode, targetNode);
1030.        axioms = mkAxioms(relMatrix);
1040.        cNodesMatrix[i][j] = nodeMatch(axioms, $context_A$, $context_B$);
1050.  return cNodesMatrix;
```

In particular, treeMatch takes two trees of Nodes (source and target) and the matrix of relations holding between atomic concepts of labels (cLabsMatrix) as input. It starts from two loops over all the nodes of source and target trees in lines 960-1040 and 990-1040. The node matching problems are constructed within these loops. For each node matching problem we take a pair of propositional formulas encoding concepts at nodes and relevant relations holding between the atomic concepts of labels using the getCnodeFormula and extractRelMatrix functions respectively. The former are memorized as context_A and context_B in lines 980 and 1010. The latter are memorized in relMatrix in line 1020. In order to reason about relations between concepts at nodes, we build the premises (axioms) in line 1030. These are a conjunction of the concepts of labels which are related in relMatrix. For example, the semantic relations in Table 1.2, which are considered when we match C_4 in the tree A and C_4 in the tree B are $Classes_B \equiv Courses_A$ and $History_B \equiv History_A$. In this case axioms is $(Classes_B \equiv Courses_A) \sqcap (History_B \equiv History_A)$. Finally, in line 1040, the semantic relations holding between the concepts at nodes are calculated by nodeMatch and are reported as a bidimensional array (cNodesMatrix); Table 1.3 shows a part of it for the example of Figure 1.1.

Table 1.3: Matrix of relations among the concepts at nodes (*cNodesMatrix*).

B / A	C_1	C_4	C_{14}	C_{17}
C_1	$=$	$\sqsupseteq$	$\sqsupseteq$	$\sqsupseteq$
C_4	$\sqsubseteq$	$=$	$\sqsupseteq$	$\sqsupseteq$
C_{12}	$\sqsubseteq$	$\sqsubseteq$	$\perp$	$\perp$
C_{16}	$\sqsubseteq$	$\sqsubseteq$	$\perp$	$\perp$

1.3.2 Node matching algorithm

Each node matching problem is converted into a propositional validity problem. Semantic relations are translated into propositional connectives using the rules described in Table 1.4 (second column).

Table 1.4: The relationship between semantic relations and propositional formulas [17, 23].

$rel(a,b)$	Translation of $rel(a,b)$ into propositional logic	Translation of Eq. 1.2 into Conjunctive Normal Form
$a \equiv b$	$a \leftrightarrow b$	N/A
$a \sqsubseteq b$	$a \rightarrow b$	$axioms \wedge context_A \wedge \neg context_B$
$a \sqsupseteq b$	$b \rightarrow a$	$axioms \wedge context_B \wedge \neg context_A$
$a \perp b$	$\neg(a \wedge b)$	$axioms \wedge context_A \wedge context_B$

The criterion for determining whether a relation holds between concepts of nodes is the fact that it is entailed by the premises. Thus, we have to prove that the following formula:

$$axioms \longrightarrow rel(context_A, context_B) \tag{1.1}$$

is valid, namely that it is true for all the truth assignments of all the propositional variables occurring in it. *axioms*, $context_A$, and $context_B$ are the same as they were defined in the tree matching algorithm. *rel* is the semantic relation that we want to prove holding between $context_A$ and $context_B$. The algorithm checks the validity of Eq. 1.1 by proving that its negation, i.e., Eq. 1.2, is unsatisfiable.

$$axioms \wedge \neg rel(context_A, context_B) \tag{1.2}$$

Table 1.4 (third column) describes how Eq. 1.2 is translated before testing each semantic relation. Notice that Eq. 1.2. is in Conjunctive Normal Form (CNF), namely it is a conjunction of disjunctions of atomic formulas. The check for equivalence is omitted in Table 1.4, since $A \equiv B$ holds if and only if $A \sqsubseteq B$ and $A \sqsupseteq B$ hold, i.e., both $axioms \wedge context_A \wedge \neg context_B$ and $axioms \wedge context_B \wedge \neg context_A$ are unsatisfiable formulas.

Let us consider the pseudo code of a basic node matching algorithm, see Algorithm 2. In line 1110, nodeMatch constructs the formula for testing disjointness. In line 1120, it converts the formula into CNF, while in line 1130, it checks the CNF formula for unsatisfiability. If the formula is unsatisfiable the disjointness relation is returned. Then, the process is repeated for the less and more general relations. If both relations hold, then the equivalence relation is returned (line 1220). If all the tests fail, the *idk* relation is returned (line 1280). In order to check the unsatisfiability of a propositional formula in a basic version of our NodeMatch algorithm we use the standard DPLL-based SAT solver [28].

Algorithm 2 The pseudo code of the node matching algorithm [23].

```
1100. String nodeMatch(String axioms, context_A, context_B)
1110.    formula = And(axioms, context_A, context_B);
1120.    formulaInCNF = convertToCNF(formula);
1130.    boolean isOpposite = isUnsatisfiable(formulaInCNF);
1140.    if (isOpposite)
1150.       return "$\perp$";
1160.    String formula = And(axioms, context_A, Not(context_B));
1170.    String formulaInCNF = convertToCNF(formula);
1180.    boolean isLG = isUnsatisfiable(formulaInCNF)
1190.    formula = And(axioms, Not(context_A), context_B);
1200.    formulaInCNF = convertToCNF(formula);
1210.    boolean isMG = isUnsatisfiable(formulaInCNF);
1220.    if (isMG && isLG)
1230.       return "=";
1240.    if (isLG)
1250.       return "$\sqsubseteq$";
1260.    if (isMG)
1270.       return "$\sqsupseteq$";
1280.    return "idk";
```

From the example in Figure 1.1, trying to prove that C_4 in the tree B is less general than C_4 in the tree A, requires constructing the following formula:

$$((Classes_B \leftrightarrow Courses_A) \wedge (History_B \leftrightarrow History_A)) \wedge \\ (Classes_B \wedge History_B) \wedge \neg(Courses_A \wedge History_A)$$

The above formula turns out to be unsatisfiable, and therefore, the less general relation holds. Notice, if we test for the more general relation between the same pair of concepts at nodes, the corresponding formula would be also unsatisfiable. Thus, the final relation returned by the NodeMatch algorithm for the given pair of concepts at nodes is the equivalence.

1.4 Efficient Semantic Matching

The node matching problem in semantic matching is a CO-NP hard problem, since it is reduced to the validity problem for the propositional calculus. In this section we present a set of optimizations for the node matching algorithm. In particular, we show that when dealing with conjunctive concepts at nodes, i.e., the concept at node is a conjunction (e.g., C_7 in the tree A in Figure 1.1

is defined as $Asian_A \wedge Languages_A$), the node matching tasks can be solved in linear time. When we have disjunctive concepts at nodes, i.e., the concept at node contains both conjunctions and disjunctions in any order (e.g., C_3 in the tree B in Figure 1.1 is defined as $College_B \wedge (Arts_B \vee Sciences_B)$), we use techniques allowing us to avoid the exponential space explosion which arises due to the conversion of disjunctive formulas into CNF. This modification is required since all state of the art SAT deciders take CNF formulas in input.

1.4.1 Conjunctive concepts at nodes

Let us make some observations with respect to Table 1.4. The first observation is that the *axioms* part remains the same for all the tests, and it contains only clauses with two variables. In the worst case, it contains $2 \times n_A \times n_B$ clauses, where n_A and n_B are the number of atomic concepts of labels occurred in *context*$_A$ and *context*$_B$, respectively. The second observation is that the formulas for testing less and more general relations are very similar and they differ only in the negated context formula (e.g., in the test for less general relation *context*$_B$ is negated). This means that Eq. 1.2 contains one clause with n_B variables plus n_A clauses with one variable. In the case of disjointness test *context*$_A$ and *context*$_B$ are not negated. Therefore, Eq. 1.2 contains $n_A + n_B$ clauses with one variable. Let us consider tests for more/less general relations, see [22, 23] for details on the other tests.

Tests for less and more general relations. Using the above observations concerning Table 1.4, Eq. 1.2 with respect to the tests for less/more general relations can be represented as follows:

$$\overbrace{\bigwedge_{q=0}^{n*m}(\neg A_s \vee B_t) \wedge \bigwedge_{w=0}^{n*m}(A_k \vee \neg B_l) \wedge \bigwedge_{v=0}^{n*m}(\neg A_p \vee \neg B_r) \wedge}^{Axioms} \overbrace{\bigwedge_{i=1}^{n} A_i}^{Context_A} \wedge \overbrace{\bigvee_{j=1}^{m} \neg B_j}^{\neg Context_B} \tag{1.3}$$

where n is the number of variables in *context*$_A$, m is the number of variables in *context*$_B$. The A_i's belong to *context*$_A$, and the B_j's belong to *context*$_B$. s, k, p are in the [0..n] range, while t, l, r are in the [0..m] range. q, w and v define the number of particular clauses. *Axioms* can be empty. Eq. 1.3 is composed of clauses with one or two variables plus one clause with possibly more variables (the clause corresponding to the negated context). The key observation is that the formula in Eq. 1.3 is Horn, i.e., each clause contains at most one positive literal. Therefore, its satisfiability can be decided in linear time by the *unit resolution rule* [4]. Notice, that DPLL-based SAT solvers require quadratic time in this case.

In order to understand how the linear time algorithm works, let us prove the unsatisfiability of Eq. 1.3 in the case of matching C_{16} in the tree A and C_{17} in the tree B in Figure 1.1. In this case, Eq. 1.3 is as follows:

$$
\begin{aligned}
&(\neg \boldsymbol{course}_A \vee class_B) \wedge (\boldsymbol{course}_A \vee \neg class_B) \wedge (\neg \boldsymbol{history}_A \vee history_B) \wedge \\
&(\boldsymbol{history}_A \vee \neg history_B) \wedge (\neg \boldsymbol{medieval}_A \vee modern_B) \wedge (\neg \boldsymbol{asia}_A \vee \neg europe_B) \wedge \\
&\boldsymbol{course}_A \wedge \boldsymbol{history}_A \wedge \boldsymbol{medieval}_A \wedge \boldsymbol{asia}_A \wedge \\
&(\neg class_B \vee \neg history_B \vee \neg modern_B \vee \neg europe_B)
\end{aligned} \tag{1.4}
$$

In Eq. 1.4, the variables from $context_A$ are written in bold face. First, we assign *true* to all unit clauses occurring in Eq. 1.4 positively. Notice these are all and only the clauses in $context_A$. This allows us to discard the clauses where $context_A$ variables occur positively (in this case: $\boldsymbol{course}_A \vee \neg class_B$, $\boldsymbol{history}_A \vee \neg history_B$). The resulting formula is as follows:

$$
\begin{aligned}
&class_B \wedge history_B \wedge \neg modern_B \wedge \neg europe_B \wedge \\
&(\neg class_B \vee \neg history_B \vee \neg modern_B \vee \neg europe_B)
\end{aligned} \tag{1.5}
$$

Eq. 1.5 does not contain any variable derived from $context_A$. Notice that, by assigning *true* to $class_B$, $history_B$ and *false* to $modern_B$, $europe_B$ we do not derive a contradiction. Therefore, Eq. 1.5 is satisfiable. In fact, a (Horn) formula is unsatisfiable if and only if the empty clause is derived (and it is satisfiable otherwise).

Let us consider again Eq. 1.5. For this formula to be unsatisfiable, all the variables occurring in the negation of $context_B$ ($class_B \vee \neg history_B \vee \neg modern_B \vee \neg europe_B$ in our example) should occur positively in the unit clauses obtained after resolving *axioms* with the unit clauses in $context_A$ ($class_B$ and $history_B$ in our example). For this to happen, for any B_j in $context_B$ there must be a clause of form $\neg A_i \vee B_j$ in *axioms*, where A_i is a formula of $context_A$. Formulas of form $\neg A_i \vee B_j$ occur in Eq. 1.3 if and only if we have the axioms of form $A_i \equiv B_j$ and $A_i \sqsubseteq B_j$. These considerations suggest the following macro steps for testing satisfiability (see [23] for details):

- Step 1. Create an array of size m. Each entry in the array stands for one B_j in Eq. 1.3.
- Step 2. For each axiom of type $A_i \equiv B_j$ and $A_i \sqsubseteq B_j$ mark the corresponding B_j.
- Step 3. If all the B_j's are marked, then the formula is unsatisfiable.

1.4.2 Disjunctive concepts at nodes

Now, we allow for the concepts of nodes to contain conjunctions and disjunctions in any order. As from Table 1.4, *axioms* is the same for all the tests. However, $context_A$ and $context_B$ may contain any number of disjunctions. Some of them are coming from the concepts of labels, while others may appear from the negated $context_A$ or $context_B$ (e.g., see less/more generality tests). With disjunctive concepts at nodes, Eq. 1.1 is a full propositional formula, and hence, no hypothesis can be made on its structure. Thus, its satisfiability must be tested by using a standard SAT decider.

In order to avoid the exponential space explosion, which may arise when converting Eq. 1.1 into CNF, we apply a set of structure preserving transformations [39]. The main idea is to replace disjunctions occurring in the original formula with newly introduced variables and to explicitly state that these variables imply the subformulas they substitute. Therefore, the size of the propositional formula in CNF grows linearly with respect to the number of disjunctions in the original formula. Thus, nodeMatch (see Algorithm 2) should be optimized by replacing all the calls to convertToCNF with calls to optimizedConvertToCNF.

1.5 The S-Match architecture

S-Match was designed and developed (in Java) as a platform for semantic matching, i.e., a modular system with the core of computing semantic relations where single components can be plugged, unplugged or suitably customized, see Figure 1.2. It is a sequential system with a parallel composition at the element level.

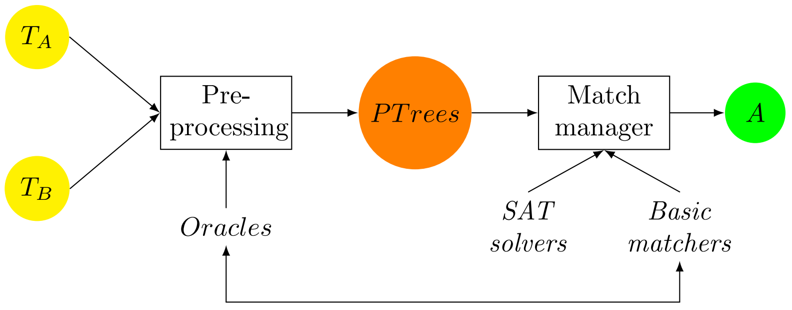


Fig. 1.2: The S-Match architecture (adapted from [9]).

The input tree-like structures (T_A and T_B) are codified in a standard internal XML format. The module taking input ontologies performs *preprocessing* (*Steps 1,2*) with the help of *oracles*, such as WordNet. The output of the module is an enriched tree. These enriched trees are stored in an internal database (*PTrees*) where they can be browsed, edited and manipulated. The *Match manager* coordinates the matching process. S-Match libraries contain *basic element level matchers* of Table 1.1 (*Step 3*) and structure level matchers that include *SAT solvers* [28] and ad hoc reasoning methods [22] (*Step 4*). Finally, the Match manager outputs the computed alignment A.

1.6 Evaluation

Let us discuss the performance and quality evaluation of S-Match. In particular, we evaluate basic and optimized versions of our system, called (S-Match$_B$) and (S-Match) respectively, against three state of the art systems, such as Cupid [30], COMA [5], and SF [33] as implemented in Rondo [34]. All the systems under consideration are fairly comparable because they are all schema-based. They differ in the specific matching techniques they use and in the way they compute alignments (see §1.2).

1.6.1 Evaluation set up

The evaluation was performed on six matching tasks from different application domains, see Table 1.5[4].

Table 1.5: Some indicators of the complexity of the test cases [23].

#	matching task	max depth	#nodes	#labels per tree	concepts of nodes
(a)	Cornell *vs.* Washington	3/3	34/39	62/64	Conjunctive Disjunctive
(b)	CIDX *vs.* Excel	3/3	34/39	56/58	Conjunctive Disjunctive
(c)	Looksmart *vs.* Yahoo	10/8	140/74	222/101	Conjunctive Disjunctive
(d)	Yahoo *vs.* Standard	3/3	333/115	965/242	Conjunctive Disjunctive
(e)	Google *vs.* Yahoo	11/11	561/665	722/945	Conjunctive Disjunctive
(f)	Google *vs.* Looksmart	11/16	706/1081	1048/1715	Conjunctive Disjunctive

There is one matching task from the academy domain: (a). It describes in a minimal way courses taught at the Cornell University and at the University of Washington. There are two tasks from the business domain: (b) and (d), e.g., BizTalk[5] purchase order schemas CIDX and Excel. Finally, there are three matching tasks on general topics: (c), (e), (f) as represented by the well-known web directories, such as Google[6], Yahoo[7], and Looksmart[8].

The reference alignments for the tasks (a) and (b) were established manually. Then, the results computed by the systems have been compared with the reference alignments. In this evaluation study we focus mostly on the per-

[4] Source files and description of the ontologies tested can be found at the Knowdive project web-site: `http://www.dit.unitn.it/~knowdive/description_SMatch_experiments.php`

[5] `http://www.microsoft.com/biztalk`

[6] `http://www.google.com/Top/`

[7] `http://dir.yahoo.com/`

[8] `http://www.looksmart.com/`

formance characteristics of S-Match, involving large matching tasks, namely ontologies with hundreds and thousands of nodes. Quality characteristics of the S-Match results which are presented here address only medium size ontologies; see [18, 21] for a large-scale quality evaluation.

There are three further observations that ensure a fair (qualitative) comparative study. The first observation is that Cupid, COMA, and Rondo can discover only the correspondences which express similarity between entities. Instead, S-Match, among others, discovers the disjointness relation which can be interpreted as strong dissimilarity in terms of other systems under consideration. Therefore, we did not take into account the disjointness relations when specifying the reference alignments. The second observation is that, since S-Match returns a matrix of relations, while all other systems return a list of the best correspondences, we used some filtering rules. More precisely we have the following two rules: (*i*) discard all the correspondences where the relation is *idk*; (*ii*) return always the *core* relations, and discard relations whose existence is implied by the core relations. Finally, whether S-Match returns the equivalence or subsumption relations does not affect the quality indicators. What only matters is the presence of the correspondences standing for those relations.

As match quality measures we have used the following indicators: *precision*, which is a correctness measure, *recall*, which is a completeness measure, *overall*, which is an estimate of the post match efforts needed for adding false negatives and removing false positives, and *F-measure*, computed as the harmonic mean of precision and recall, see for details [9]. As a performance measure we have used *time*. It estimates how fast systems are when producing alignments fully automatically. Time is important, since it shows the ability of matching systems to scale up.

In our experiments each test has two degrees of freedom: *directionality* and *use of oracles*. By directionality we mean here the direction in which correspondences have been computed: from the first ontology to the second one (forward direction), or vice versa (backward direction). We report the best results obtained with respect to directionality, and use of oracles allowed. We were not able to plug a thesaurus in Rondo, since the version we have is standalone, and it does not support the use of external thesauri. Thesauri of S-Match, Cupid, and COMA were expanded with terms necessary for a fair competition (e.g., expanding *uom* into *unitOfMeasure*, a complete list is available at the URL in footnote 4).

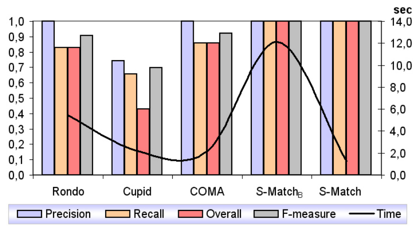
All the tests have been performed on a P4-1700, with 512 MB of RAM, with the Windows XP operating system, and with no applications running but a single matching system. The systems were limited to allocate no more than 512 MB of memory. All the tuning parameters (e.g., thresholds, combination strategies) of the systems were taken by default (e.g., for COMA we used *NamePath* and *Leaves* matchers combined in the *Average* strategy) for all the tests. S-Match was also used in default configuration, e.g., threshold for string-based matchers was 0.6.

1.6.2 Evaluation results

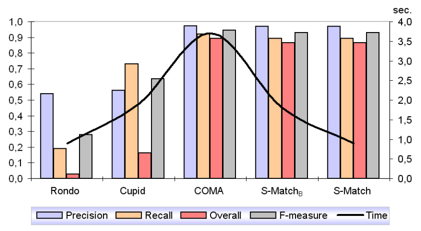
We present the time performance results for all the tasks of Table 1.5, while quality results, as from the previous discussion are presented for the tasks (a) and (b).

The evaluation results were first published in [23] and are shown in Figure 1.3. For example, on task (a), since all the labels at nodes in the given test case were correctly encoded into propositional formulas, all the quality measures of S-Match reach their highest values. In fact, as discussed before, the propositional SAT solver is correct and complete. This means that once the element level matchers have found all and only the correspondences, S-Match will return all of them and only the correct ones. In turn, on task (b), S-Match performs as good as COMA and outperforms other systems in terms of quality indicators. Also, the optimized version of S-Match works more than 4 times faster than COMA, more than 2 times faster than Cupid, and as fast as Rondo.

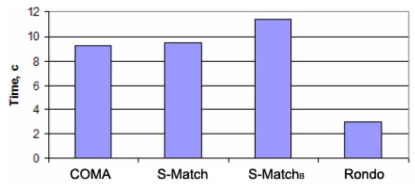
For what concerns the other tasks, whenever a tested system (e.g., Cupid) went out of memory, its results were not reported. On the task (d), S-Match



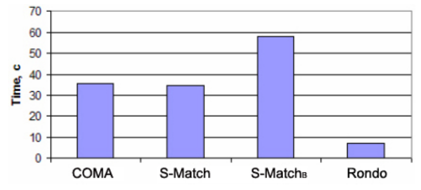
(a) Cornell vs. Washington



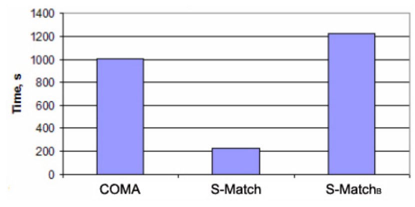
(b) CIDX vs. Excel



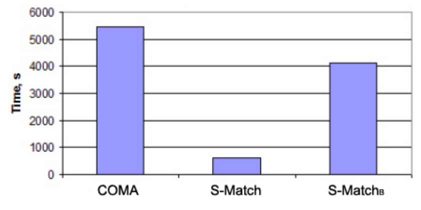
(c) Looksmart vs. Yahoo



(d) Yahoo vs. Standard



(e) Google vs. Yahoo



(f) Google vs. Looksmart

Fig. 1.3: The evaluation results.

works about 40% faster than S-Match$_B$. It performs 1% faster than COMA and about 5 times slower than Rondo. The relatively small improvement in this case can be explained by noticing that the maximum depth in both trees is 3 and that the average number of labels at nodes is about 2. Hence, here the optimizations cannot significantly influence the system performance.

In the case of task (e), S-Match is more than 6 times faster than S-Match$_B$. COMA performs about 5 times slower than S-Match. Finally, in the case of the biggest matching task, i.e., (f), S-Match performs about 9 times faster than COMA, and about 7 times faster than S-Match$_B$.

Having considered matching tasks of Table 1.5, we conclude that our system performs (in terms of execution time) slightly slower than COMA and Rondo on the ontologies with one up to three hundred of nodes. At the same time, it is considerably faster on the ontologies with more than five hundreds nodes, thereby indicating for system scalability.

1.7 Conclusions and future work

In this chapter we have presented a semantic matching approach to ontology matching. Our solution builds on top of the past approaches at the element level and uses model-based techniques at the structure level. We implemented our approach within the S-Match system and conducted a comparative evaluation of S-Match against three state of the art systems with encouraging results.

Let us now provide an outlook for the ontology matching field in terms of the future challenges to be addressed and how the work on S-Match is positioned with respect to those challenges. In particular, the work in [42] articulated ten challenges for ontology matching, including: (*i*) large-scale evaluation, (*ii*) performance of ontology matching techniques, (*iii*) discovering missing background knowledge, (*iv*) uncertainty in ontology matching, (*v*) matcher selection and self-configuration, (*vi*) user involvement, (*vii*) explanation of matching results, (*viii*) social and collaborative ontology matching, (*ix*) alignment management: infrastructure and support, and (*x*) reasoning with alignments.

Some of these challenges have already been tackled within the work on S-Match. Specifically, a contribution to the large-scale evaluation (challenge (*i*)) has been developed in [21] by providing a testing methodology which is able to estimate quality of the alignments between ontologies with hundreds and thousands of nodes. Performance of S-Match (challenge (*ii*)) has been discussed in this chapter, see also [22, 23]. In turn, the work in [18] introduced an automatic approach to discover the missing background knowledge (challenge (*iii*)) in matching tasks by using semantic matching iteratively. Finally, explanations (challenge (*vii*)) of the S-Match results has been tackled in [43].

Future work includes user involvement (challenge (*vi*)), e.g., through the development of an interactive semantic matching system. It will improve the quality of the alignments by focusing user's attention on the critical points where his/her input is maximally useful. Moreover, users have to be provided with the system that is configurable and customizable, such that they themselves can improve it, thereby arriving to the exact solution that fits best their needs and preferences.

References

1. P. Bernstein, S. Melnik, M. Petropoulos, and C. Quix. Industrial-strength schema matching. *ACM SIGMOD Record*, 2004.
2. P. Bouquet, L. Serafini, and S. Zanobini. Semantic coordination: A new approach and an application. In *Proceedings of ISWC*, 2003.
3. X. Chai, M. Sayyadian, A. Doan, A. Rosenthal, and L. Seligman. Analyzing and revising mediated schemas to improve their matchability. In *Proceedings of VLDB*, 2008.
4. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 1960.
5. H. Do and E. Rahm. COMA – a system for flexible combination of schema matching approaches. In *Proceedings of VLDB*, 2002.
6. H. Do and E. Rahm. Matching large schemas: Approaches and evaluation. *Information Systems*, 2007.
7. A. Doan and A. Halevy. Semantic integration research in the database community: A brief survey. *AI Magazine*, 2005. Special issue on Semantic integration.
8. M. Ehrig, S. Staab, and Y. Sure. Bootstrapping ontology alignment methods with APFEL. In *Proceedings of ISWC*, 2005.
9. J. Euzenat and P. Shvaiko. *Ontology matching*. Springer, 2007.
10. J. Euzenat and P. Valtchev. Similarity-based ontology alignment in OWL-lite. In *Proceedings of ECAI*, 2004.
11. S. Falconer and M. Storey. A cognitive support framework for ontology mapping. In *Proceedings of ISWC/ASWC*, 2007.
12. A. Gal, A. Anaby-Tavor, A. Trombetta, and D. Montesi. A framework for modeling and evaluating automatic semantic reconciliation. *The VLDB Journal*, 2005.
13. F. Giunchiglia. Contextual reasoning. *Epistemologia*, 1993.
14. F. Giunchiglia, M. Marchese, and I. Zaihrayeu. Encoding classifications into lightweight ontologies. *Journal of Data Semantics*, 2007.
15. F. Giunchiglia and P. Shvaiko. Semantic matching. *Knowledge Engineering Review*, 2003.
16. F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-Match: an algorithm and an implementation of semantic matching. In *Proceedings of ESWS*, 2004.
17. F. Giunchiglia, P. Shvaiko, and M. Yatskevich. Semantic schema matching. In *Proceedings of CoopIS*, 2005.
18. F. Giunchiglia, P. Shvaiko, and M. Yatskevich. Discovering missing background knowledge in ontology matching. In *Proceedings of ECAI*, 2006.
19. F. Giunchiglia, P. Shvaiko, and M. Yatskevich. Semantic matching. *Encyclopedia of Database Systems*, 2009, to appear.
20. F. Giunchiglia and M. Yatskevich. Element level semantic matching. In *Proceedings of the workshop on Meaning Coordination and Negotiation at ISWC*, 2004.

21. F. Giunchiglia, M. Yatskevich, P. Avesani, and P. Shvaiko. A large scale dataset for the evaluation of ontology matching systems. *Knowledge Engineering Review*, 2009.
22. F. Giunchiglia, M. Yatskevich, and E. Giunchiglia. Efficient semantic matching. In *Proceedings of ESWC*, 2005.
23. F. Giunchiglia, M. Yatskevich, and P. Shvaiko. Semantic matching: Algorithms and implementation. *Journal on Data Semantics*, 2007.
24. N. Guarino. The role of ontologies for the semantic web (and beyond). Technical report, Laboratory for Applied Ontology, Institute for Cognitive Sciences and Technology (ISTC-CNR), 2004.
25. L. Haas, M. Hernández, H. Ho, L. Popa, and M. Roth. Clio grows up: from research prototype to industrial tool. In *Proceedings of SIGMOD*, 2005.
26. W. Hu, Y. Qu, and G. Cheng. Matching large ontologies: A divide-and-conquer approach. *Data and Knowledge Engineering*, 2008.
27. P. Lambrix and H. Tan. SAMBO - a system for aligning and merging biomedical ontologies. *Journal of Web Semantics*, 2006.
28. D. Le Berre. Sat4j: A satisfiability library for Java. http://www.sat4j.org/, 2004.
29. J. Li, J. Tang, Y. Li, and Q. Luo. RiMOM: A dynamic multi-strategy ontology alignment framework. *Transaction on Knowledge and Data Engineering*, 2008.
30. J. Madhavan, P. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of VLDB*, 2001.
31. B. Magnini, L. Serafini, and M. Speranza. Making explicit the semantics hidden in schema models. In *Proceedings of the Workshop on Human Language Technology for the Semantic Web and Web Services at ISWC*, 2003.
32. R. McCann, W. Shen, and A. Doan. Matching schemas in online communities: A web 2.0 approach. In *Proceedings of ICDE*, 2008.
33. S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: a versatile graph matching algorithm. In *Proceedings of ICDE*, 2002.
34. S. Melnik, E. Rahm, and P. Bernstein. Developing metadata-intensive applications with Rondo. *Journal of Web Semantics*, 2003.
35. G. Miller. WordNet: A lexical database for english. *Communications of the ACM*, 1995.
36. P. Mork, L. Seligman, A. Rosenthal, J. Korb, and C. Wolf. The Harmony integration workbench. *Journal on Data Semantics*, 2008.
37. N. Noy. Semantic integration: A survey of ontology-based approaches. *ACM SIGMOD Record*, 2004.
38. N. Noy and M. Musen. The PROMPT suite: interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 2003.
39. D. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 1986.
40. E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 2001.
41. P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics*, 2005.
42. P. Shvaiko and J. Euzenat. Ten challenges for ontology matching. In *Proceedings of ODBASE*, 2008.
43. P. Shvaiko, F. Giunchiglia, P. Pinheiro da Silva, and D. McGuinness. Web explanations for semantic heterogeneity discovery. In *Proceedings of ESWC*, 2005.