PhD Dissertation

International Doctorate School in Information and Communication Technology

DIT - University of Trento

SEMANTIC MATCHING ALGORITHMS AND IMPLEMENTATION

Mikalai Yatskevich

Advisor:

Prof. Fausto Giunchiglia

Università degli Studi di Trento

March 2008

Abstract

Match is a critical operator in many well-known metadata intensive applications, such as schema/ontology integration, data warehouses, data integration, e-commerce, etc. The match operator takes two graph-like structures and produces a mapping between the nodes of the graphs that correspond semantically to each other. This dissertation focuses only on the task of development of the novel algorithms implementing the match operator.

Many various solutions of matching have been proposed so far. This work concentrates on a schema-based solution, namely a solution exploiting only the schema information, and not considering instance information. To ground the choice of the solution, this thesis provides a comprehensive coverage of the schema-based approaches used in ontology matching as well as their applications by reviewing state of the art in the field in a uniform way. It also points out how the approach developed in the thesis fits in with existing work.

The thesis proposes the matching algorithm implementing the so-called *semantic matching* approach. This approach is based on two key ideas. The first is that correspondences are calculated between entities of ontologies by computing logical relations (e.g., equivalence, subsumption, disjointness), instead of computing coefficients rating match quality in the [0 1] range, as it is the case in many other approaches. The second idea is

that the relations are determined by analyzing the meaning which is codified in the elements 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 the translation of the matching problem into a propositional validity problem, which can then be efficiently resolved using sound and complete state of the art propositional satisfiability deciders.

The basic, efficient and structure preserving semantic matching algorithms have been designed and developed. The approach has been evaluated on various real world test cases with encouraging results, thus, proving empirically its benefits.

Keywords

Ontology matching, schema matching, ontology alignment, semantic heterogeneity, semantic matching, element level semantic matching, matching evaluation

Wishes

Acknowledgments

I would like to express my gratitude to my supervisor, Fausto Giunchiglia, whose expertise, understanding, and patience, added considerably to my graduate experience. I appreciate his vast knowledge and skill in many areas (e.g., vision, aging, ethics, interaction with participants), and his assistance in writing as well as being supportive in my work, and guidance in my entrance into AI domain and life in general. Also, I am thankful for his insightful suggestions that helped me made the right strategic choices at many crucial decision points along these years.

I thank all my friends and everyone who have contributed to this thesis through many fruitful discussions, technical advice, encouraging words and in many other ways.

I would also like to thank my family for the support they provided me through my entire life and in particular, I must acknowledge my wife and best friend, Tanya, without whose love, encouragement and editing assistance, I would not have finished this thesis.

This thesis is dedicated to my grandmother Polina, without whom none of this would have been even possible. Many thanks for endless hours she spent teaching me math and fostering my computer science skills.

Wishes

Contributions and publications

This work has been developed in collaboration with various people (as the publications indicate) and in particular with: Fausto Giunchiglia, Pavel Schvaiko, Paolo Avesani and Jérôme Euzenat.

This thesis makes the following contributions:

- Design and development of a new semantic schema matching algorithm, which guarantees correctness and completeness of its results at the structure level;
- A library of element level matchers which builds on the advances of the previous solutions at the element level;
- An extension of the semantic matching approach for efficiently handling the most frequent classes of the matching tasks;
- Design and development of the structure preserving semantic matching algorithm for computing the similarity of the two graph-like structures;
- Creation of a large scale real world data set from the Google, Yahoo! and Looksmart web directories for the evaluation of quality results of matching systems;
- Empirical evaluation of the semantic matching approach on various data sets;

Part of the material of the thesis has been published in various conferences and journals (in order of appearance):

- [62]: Fausto Giunchiglia and Mikalai Yatskevich. Element level semantic matching. In *Proceedings of Meaning Coordination and Negotiation workshop at ISWC*, 2004.

- [58]: Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. S-Match: an algorithm and an implementation of semantic matching. In *Proceedings of the European Semantic Web Symposium (ESWS)*, 2004.

- [63]: Fausto Giunchiglia, Mikalai Yatskevich, and Enrico Giunchiglia. Efficient Semantic Matching. In *Proceedings of 2nd European Semantic Web Conference (ESWC)*, 2005.

- [59]: Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. Semantic schema matching. In *Proceedings of the International Conference on Cooperative Information Systems (CoopIS)*, 2005.

- [6]: Paolo Avesani, Fausto Giunchiglia, and Mikalai Yatskevich A large scale taxonomy mapping evaluation. In *Proceedings of International Semantic Web Conference (ISWC)*, 2005.

- [45]: Jérôme Euzenat, Heiner Stuckenschmidt, and Mikalai Yatskevich Introduction to the ontology alignment evaluation 2005. In *Proceedings of K-CAP 2005 Workshop on Integrating Ontologies*, 2005.

- [43]: Jérôme Euzenat, Malgorzata Mochol, Pavel Shvaiko, Heiner Stuckenschmidt, Ondřej Šváb, Vojtěch Svátek, Willem Robert van Hage, and Mikalai Yatskevich. Results of the ontology alignment evaluation initiative 2006. In *Proceedings of the International Workshop*

on Ontology Matching (OM) at the International Semantic Web Conference (ISWC), 2006.

- [60]: Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. Discovering missing background knowledge in ontology matching. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 2006.

- [64]: Fausto Giunchiglia, Mikalai Yatskevich, and Pavel Shvaiko. Semantic matching: algorithms and implementation. *Journal on Data Semantics (JoDS)*, IX, 2006. to appear.

Whenever results of any of these works are reported, proper citations are made in the body of the thesis.

Contents

List of Figures

List of Tables

Introduction

Motivating example

To motivate the matching problem, let us use two simple XML schemas that are shown in Figure 1 and exemplify one of the possible situations which arise, for example, when resolving a schema integration task.

Figure 1: Two XML schemas

Suppose an e-commerce company needs to finalize a corporate acquisition of another company. To complete the acquisition we have to integrate databases of the two companies. The documents of both companies are stored according to XML schemas O1 and O2, respectively. Numbers in boxes are the unique identifiers of the XML elements. A first step in integrating the schemas is to identify candidates to be merged or to have

taxonomic relationships under an integrated schema. This step refers to a process of schema matching. For example, the elements with labels *Office_Products* in O1 and in O2 are the candidates to be merged, while the element with label *Digital_Cameras* in O2 should be subsumed by the element with label *Photo_and_Cameras* in O1. Once the correspondences between two schemas have been determined, the next step has to generate query expressions that automatically translate data instances of these schemas under an integrated schema.

Structure of the thesis

The thesis is organized in five parts.

Part one is dedicated to the definition of the ontology matching problem. It technically defines the ontology matching process and its result: the alignment.

Part two provides a comprehensive coverage of the schema-based approaches used for ontology matching. Chapter 2 defines a classification of the matching approaches, presents some basic methods and matching strategies used for designing an ontology matching system. Chapter 3 presents a large number of state of the art schema-based matching systems, discussed in light of the classifications, methods and strategies of the previous chapter. It also points out how the approach further developed in this thesis fits in with existing work.

Part three is devoted to the semantic matching algorithms proposed in this thesis. Chapter 4 introduces semantic matching approach. It also discusses in detail the main macro steps realizing the semantic matching algorithm. Chapter 5 discusses efficiency improvements to the semantic matching algorithm. Chapter 6 presents an extension of the semantic matching approach to deal with the complex structures.

Part four is devoted to the test beds generation and evaluation of ontology matching and semantic matching in particular. Chapter 7 presents a methods for the large scale dataset generation. Chapter 8 reports the results of the conducted experiments.

Finally, part five concludes. Chapter 9 summarizes the work done in the thesis.

Part I

The matching problem

Chapter 1

The matching problem

There have been different formalizations of matching and its result, see, for example, [13, 81, 74, 15, 136]. We provide here a general definition, following the work in [129].

The *matching* operation determines the alignment A' for a pair of ontologies $\mathsf{O1}$ and $\mathsf{O2}$. There are some other parameters which can extend the definition of the matching process, namely: (i) the use of an input alignment A, which is to be completed by the process; (ii) the matching parameters, p, e.g., weights, thresholds; and (iii) external resources used by the matching process, r, e.g., common knowledge and domain specific thesauri. Technically, this process can be defined as follows.

The matching process can be viewed as a function f which, from a pair of ontologies $O1$ and $O2$ to match, an input alignment A, a set of parameters p and a set of oracles and resources r, returns a new alignment A' between these ontologies:

$$A' = f(O1, O2, A, p, r)$$

The multiple matching process can be viewed as a function f which, from

a set of ontologies to match $\{O1, \ldots On\}$*, an input multi-alignment* A*, a set of parameters* p *and a set of oracles and resources* r*, returns a new multi-alignment* A' *between these ontologies:*

$$A' = f(O1, \ldots On, A, p, r)$$

The matching process is the main subject of this thesis. However, before discussing its internals, let us first consider what it provides: the alignment.

Alignments express the correspondences between entities belonging to different ontologies. We focus here on matching between two ontologies. In case of multiple matching, the definitions can be straightforwardly extended by using n-ary correspondences. A correspondence must express the two corresponding entities and the relation that is supposed to hold between them. We provide the definition of the alignment following the work in [42, 15].

Given two ontologies, a correspondence is a 5-tuple:

$$\langle id, e_1, e_2, n, R \rangle,$$

such that

- *id is a unique identifier of the given correspondence;*
- e_1 *and* e_2 *are the entities (e.g., tables, XML elements, properties, classes) of the first and the second ontology, respectively;*
- n *is a confidence measure (typically in the [0 1] range) holding for the correspondence between the entities* e_1 *and* e_2*;*
- R *is a relation (e.g., equivalence* $(=)$*, more general* $(\sqsupseteq)$*, disjointness* $(\perp)$*, overlapping* $(\sqcap)$*) holding between the entities* e_1 *and* e_2*.*

The correspondence $\langle id, e_1, e_2, n, R \rangle$ asserts that the relation R holds between the ontology entities e_1 and e_2 with confidence n. The usage of confidences is that the higher the degree, the most likely the relation holds.

Given two ontologies O1 and O2, an alignment is made up of a set of correspondences between pairs of entities belonging to O1 and O2, respectively.

For example, in Figure 1 (p.xxi), according to some matching algorithm based on linguistic and structure analysis, the confidence measure (for the fact that the equivalence relation holds) between entities with labels *Photo_and_Cameras* in O1 and *Cameras_and_Photo* in O2 could be 0.67. Suppose that this matching algorithm uses a threshold of 0.55 for determining the resulting alignment, i.e., the algorithm considers all the pairs of entities with a confidence measure higher than 0.55 as correct correspondences. Thus, our hypothetical matching algorithm should return to the user the following correspondence:

$$\langle id_{5,4}, Photo\_and\_Cameras, Cameras\_and\_Photo, 0.67, = \rangle.$$

However, the relation between the same pair of entities, according to another matching algorithm which is able to determine that both entities mean the same thing, could be exactly the equivalence relation (without computing the confidence measure). Thus, returning to the user

$$\langle id_{5,4}, Photo\_and\_Cameras, Cameras\_and\_Photo, n/a, = \rangle.$$

By analogy with mathematical functions, it is useful to define some properties of the alignments. These apply when the only considered relation is equality (=). One can ask for a total alignment with regard to one ontology, i.e., all the entities of one ontology must be successfully mapped to the other one. This property is purposeful whenever thoroughly transcribing knowledge from one ontology to another is the goal: there is no entity that cannot be translated.

One can also require the mapping to be injective with regard to one ontology, i.e., all the entities of the other ontology is part of at most one correspondence. Injectivity is useful in ensuring that entities that are distinct in one ontology remain distinct in the other one. In particular, this contributes to the reversebility of alignments.

Usual mathematical properties apply to these alignments. In particular, a total alignment from O1 to O2 is a surjective alignment from O2 to O1. A total alignment from both O1 and O2 which is injective from one of them is a bijection. In mathematical English, an injective function is said to be *one-to-one* and a surjective function to be *onto*. Due to the wide use among matching practitioners of the term *one-to-one* for a bijective, i.e., both injective and surjective, alignment, we will only use one-to-one for bijective.

In conceptual models and databases, the terms multiplicity or cardinality denote the constraints on a relation. Usual notations are 1:1, 1:m, n:1 or n:m. If we consider only total and injective property, denoted as 1 for injective and total, ? for injective, + for total and * for none, and the two possible orientations of the alignments, from O1 to O2 and from O2 to O1, the multiplicities become: ?:?, ?:1, 1:?, 1:1, ?:+, +:?, 1:+, +:1, +:+, ?:*, *:?, 1:*, *:1, +:*, *:+, *:* [41].

Part II

State of the art: ontology matching approaches

Chapter 2

Ontology matching techniques

Having defined what the matching problem is, we overview some classifications of the techniques that can be used for solving this problem. In particular, surveys on the topic through the recent years have been provided in [124, 135, 74]; while the major contributions of the previous decades are presented in [8, 78, 130, 75, 118]. The work presented in [74] focuses on current state of the art in ontology matching. Authors review recent approaches, techniques and tools. The survey of [135] concentrates on approaches to ontology-based information integration and discusses general matching approaches that are used in information integration systems. However, none of the above mentioned works provide a comparative review of the existing ontology matching techniques and systems. On the contrary, the survey of [124] is devoted to a classification of database schema matching approaches and a comparative review of matching systems. Notice that these three works address the matching problem from different perspectives (artificial intelligence, information systems, databases) and analyze disjoint sets of systems. [129] have attempted at considering the above mentioned works together, focusing on schema-based matching methods, and aiming to provide a common conceptual basis for their analysis. In this chapter we

rely on [129] while discussing the classification of the matching approaches.

2.1 A classification of matching techniques

In this section we discuss only schema-based elementary matchers. Therefore, only schema level information is considered, not instance data[1].

For classifying elementary schema-based matching techniques, two synthetic classifications are presented in [129](see Figure 2.1). These two classifications are presented as two trees sharing their leaves. The leaves represent classes of elementary matching techniques and their concrete examples. Two synthetic classifications are:

- *Granularity/Input Interpretation* classification is based on (*i*) granularity of match, i.e., element- or structure-level, and then (*ii*) on how the techniques generally interpret the input information;
- *Kind of Input* classification is based on the kind of input which is used by elementary matching techniques.

The overall classification of Figure 2.1 can be read both in descending (focusing on how the techniques interpret the input information) and ascending (focusing on the kind of manipulated objects) manner in order to reach the *Basic Techniques* layer. Let us discuss in turn *Granularity/Input Interpretation*, *Basic Techniques*, *Kind of Input* layers together with supporting arguments for the categories/classes introduced at each layer.

Elementary matchers are distinguished by the *Granularity/Input interpretation* layer according to the following classification criteria:

- *Element-level vs structure-level.* Element-level matching techniques compute correspondences by analyzing entities in isolation, ignoring

[1]Prominent solutions of instance-based ontology matching can be found in [33, 35].

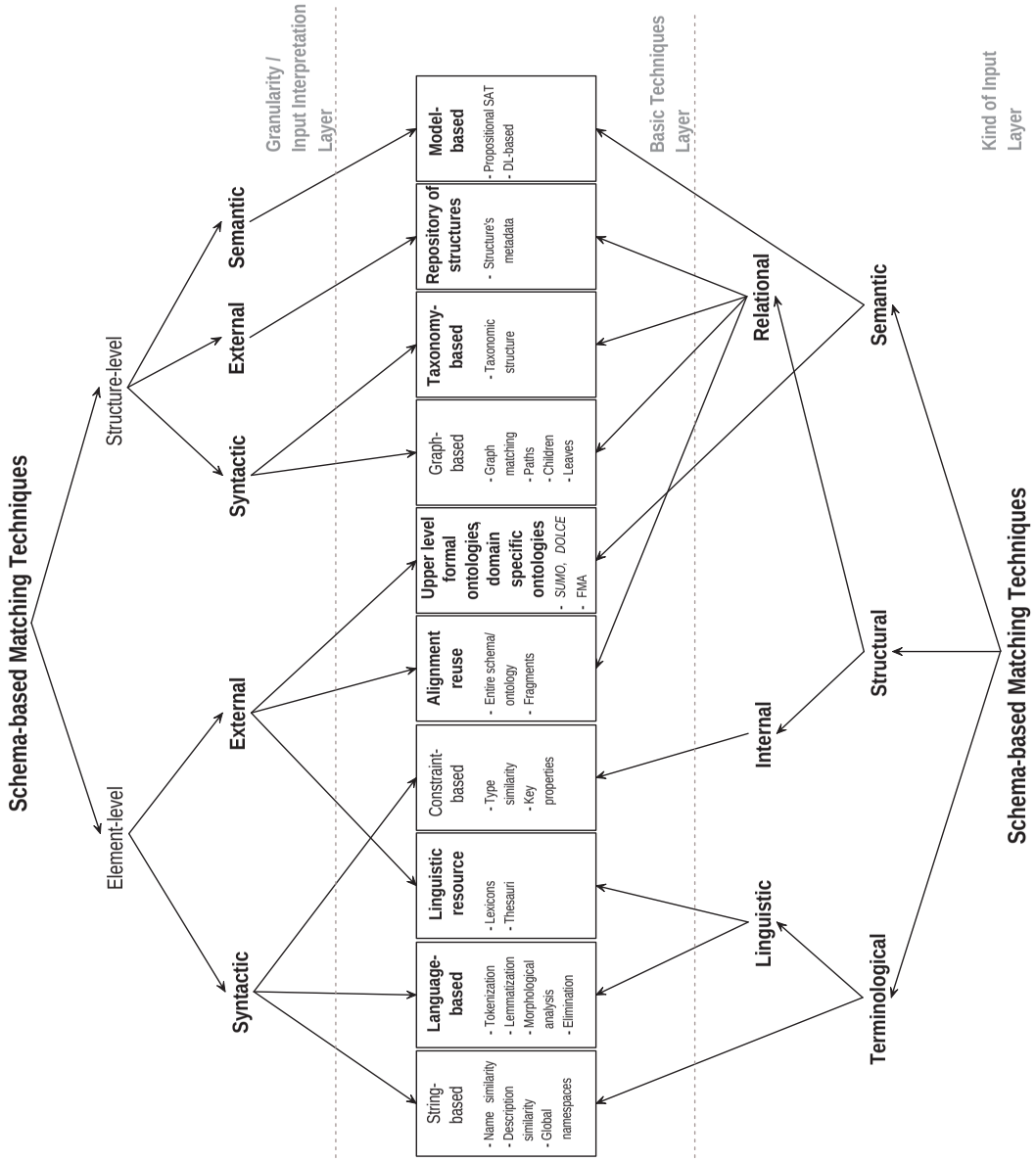

Figure 2.1: A retained classification of elementary schema-based matching techniques

their relations with other entities. Structure-level techniques compute correspondences by analyzing how entities appear together in a structure. This criterion is the same as first introduced in [124].

- *Syntactic vs external vs semantic.* The key characteristic of the syntactic techniques is that they interpret the input as a function of its sole structure following some clearly stated algorithm. External are the techniques exploiting auxiliary (external) resources of a domain and common knowledge in order to interpret the input. These resources might be human input or some thesaurus expressing the relationships between terms. The key characteristic of the semantic techniques is that they use some formal semantics (e.g., model-theoretic semantics) to interpret the input and justify their results. In case of a semantic based matching system, exact algorithms are complete (i.e., they guarantee a discovery of all the possible alignments) while approximate algorithms tend to be incomplete.

Distinctions between classes of elementary matching techniques in the *Basic Techniques* layer of the classification are motivated by the way a matching technique interprets the input information in each concrete case. In particular, a label can be interpreted as a string (a sequence of letters from an alphabet) or as a word or a phrase in some natural language, a hierarchy can be considered as a graph (a set of nodes related by edges) or a taxonomy (a set of concepts having a set-theoretic interpretation organized by a relation which preserves inclusion). Thus, the following classes of elementary ontology matching techniques are introduced at the element-level: *string-based*, *language-based*, *based on linguistic resources*, *constraint-based*, *alignment reuse*, and *based on upper level and domain specific formal ontologies*. At the structure-level we distinguish between: *graph-based*, *taxonomy-based*, *based on repositories of structures*, and *model-based tech-*

niques.

The *Kind of Input* layer classification is concerned with the type of input considered by a particular technique:

- The first level is categorized depending on which kind of data the algorithms work on: strings (*terminological*), structure (*structural*) or models (*semantics*). The two first ones are found in the ontology descriptions, the last one requires some semantic interpretation of the ontology and usually uses some semantically compliant reasoner to deduce the correspondences.

- The second level of this classification decomposes further these categories if necessary: *terminological* methods can be *string-based* (considering the terms as sequences of characters) or based on the interpretation of these terms as linguistic objects (*linguistic*). The structural methods category is split into two types of methods: those which consider the *internal* structure of entities (e.g., attributes and their types) and those which consider the relation of entities with other entities (*relational*).

The order follows that of the *Granularity/Input Interpretation* classification and these techniques are divided in two sections concerning element-level techniques (§2.1.1) and structure-level techniques (§2.1.2).

2.1.1 Element-level techniques

String-based techniques

These techniques are often used in order to match names and name descriptions of ontology entities. They consider strings as sequences of letters in an alphabet. They are typically based on the following intuition: the more similar the strings, the more likely they denote the same concepts.

A comparison of different string matching techniques, from *distance* like functions to *token-based distance* functions can be found in [26]. Usually, distance functions map a pair of strings to a real number, where a smaller value of the real number indicates a greater similarity between the strings. Some examples of string-based techniques which are extensively used in matching systems are *prefix*, *suffix*, *edit distance*, and *n-gram* (see Section 4.4 for more detail).

Language-based techniques

These techniques consider names as words in some natural language (e.g., English). They are based on Natural Language Processing (NLP) techniques exploiting morphological properties of the input words. The prominent examples of them are tokenization, lemmatization and elimination. Usually, the above mentioned techniques are applied to names of entities before running string-based or lexicon-based techniques in order to improve their results. However, we consider these language-based techniques as a separate class of matching techniques, since they can be naturally extended, for example, in a distance computation (by comparing the resulting strings or sets of strings).

Constraint-based techniques

These are algorithms which deal with the internal constraints being applied to the definitions of entities, such as types, cardinality of attributes, and keys. We omit here a discussion of matching keys as these techniques appear in our classification without changes with respect to the original publication [124].

Linguistic resources

Linguistic resources, such as common knowledge or domain specific thesauri are used in order to match words (in this case names of ontology entities are considered as words of a natural language) based on linguistic relations between them (e.g., synonyms, hyponyms).

Alignment reuse

These techniques represent an alternative way of exploiting external resources, which record alignments of previously matched ontologies. For instance, when we need to match ontology o' and o'', given the alignments between o and o', and between o and o'' from the external resource, storing previous match operations results. The alignment reuse is motivated by the intuition that many ontologies to be matched are similar to already matched ontologies, especially if they are describing the same application domain. These techniques are particularly promising when dealing with large ontologies consisting of hundreds and thousands of entities. In these cases, first, large match problems are decomposed into smaller subproblems, thus generating a set of ontology fragments matching problems. Then, reuse of previous match results can be more effectively applied at the level of ontology fragments rather than at the level of entire ontologies. The approach was first introduced in [124] and later was implemented as two matchers, i.e., (*i*) reuse alignments of entire ontologies, or (*ii*) their fragments [31, 5, 125].

Upper level and domain specific formal ontologies

These techniques use as external sources of knowledge upper level and domain specific formal ontologies. Examples of the upper level ontologies are the Suggested Upper Merged Ontology (SUMO) [107] and Descriptive

Ontology for Linguistic and Cognitive Engineering (DOLCE) [50]. The key characteristic of these ontologies is that they are logic-based systems, and therefore, matching techniques exploiting them can be based on the analysis of interpretations. Thus, these are semantic techniques. For the moment, we are not aware of any matching systems which use these kind of techniques. However, it is quite reasonable to assume that this will happen in the near future. In fact, for example, the DOLCE ontology aims at providing a formal specification (axiomatic theory) for the top level part of WordNet. Therefore, systems exploiting WordNet now in their matching process (and aware of some of its limitations [51]) might also consider using DOLCE as a potential extension.

Domain specific formal ontologies can also be used as external sources of background knowledge. Such ontologies are focusing on a particular domain and use terms in a sense that is relevant only to this domain and which is not related to similar concepts in other domains. For example, in the anatomy domain, an ontology such as The Foundational Model of Anatomy (FMA)[2] can be used as the context for the other medical ontologies to be matched (as long as it is known that the reference ontology covers the ontologies to be matched). This can be particularly useful for providing the missing structure when matching poorly structured resources [1].

2.1.2 Structure-level techniques

Graph-based techniques

These are graph algorithms which consider the input as labeled graphs. The applications (e.g., database schemas, or ontologies) are viewed as graph-like structures containing terms and their inter-relationships. Finding the correspondences between elements of such graphs corresponds to

[2] http://sig.biostr.washington.edu/projects/fm/AboutFM.html

solving a form of the graph homomorphism problem [52]. Usually, the similarity comparison between a pair of nodes from the two ontologies is based on the analysis of their positions within the graphs. The intuition behind is that, if two nodes from two ontologies are similar, their neighbors might also be somehow similar. Below, we present some particular matchers representing this intuition.

Taxonomy-based techniques

These are also graph algorithms which consider only the specialization relation. The intuition behind taxonomic techniques is that *is-a* links connect terms that are already similar (being a subset or superset of each other), therefore their neighbors may be also somehow similar.

Repository of structures

Repositories of structures store ontologies and their fragments together with pairwise similarity measure, e.g., coefficients in the [0 1] range between them. Notice that unlike the alignment reuse, repository of structures stores only similarities between ontologies, not alignments. In the following, to simplify the presentation, we call ontologies or their fragments as structures. When new structures are to be matched, they are first checked for similarity against the structures which are already available in the repository. The goal is to identify structures which are sufficiently similar to be worth matching in more detail, or reusing already existing alignments, thus, avoiding the match operation over the dissimilar structures. Obviously, the determination of similarity between structures should be computationally cheaper than matching them in full detail. The approach of [125], to matching two structures proposes to use some metadata describing these structures, such as structure name, root name, number of nodes, maximal path length, etc. These indicators are then analyzed and

are aggregated into a single coefficient, which estimates similarity between them. For example, two ontologies may be found as an appropriate match if they both have the same number of nodes.

Model-based

These are algorithms which handle the input based on its semantic interpretation (e.g., model-theoretic semantics). Thus, they are well grounded deductive methods. Examples are propositional satisfiability (SAT) and description logics (DL) reasoning techniques.

2.2 Summary

There is a variety of techniques that can be used for ontology matching. The classification discussed in this chapter provides a common conceptual basis, and, hence, can be used for comparing (analytically) different existing ontology matching systems as well as for designing a new one, taking advantages of state of the art solutions. Also, classifications of matching methods provide some guidelines which help in identifying families of matching techniques.

This chapter provided two such classifications based on granularity and input interpretation on one side and the kind of input on the other side.

Chapter 3

Overview of matching systems

This chapter is devoted to an overview of existing matching systems which have emerged during the last decade. There have already been done some comparisons of a number of matching systems, in particular in [119, 124, 30, 74, 109, 34, 129]. Our purpose here is not to compare them in full detail, although we give some comparisons, but rather to show their variety, in order to demonstrate in how many different ways the methods presented in the previous chapter have been practically exploited. We present the matching systems in light of the classifications of Chapter 2. We also point to concrete basic matchers and matching strategies used in the considered systems.

In order to facilitate the presentation we follow two rules. First, the year of the system appearance is considered. Then, if there are some evolutions of the system or very similar systems, these are discussed close to each other. Since the main focus of this thesis is on schema-based matching, instance-based systems (e.g., LSD [33], GLUE [35], Automatch [12], sPLMap [108]) as well as meta-matching systems (APFEL [38], eTuner [127]) were excluded from the consideration, see [44] for an overview. We have also excluded from consideration the systems which assume that align-

ments have already been established, and use this assumption as a prerequisite of running the actual system. These approaches include such information integration systems as: Tsimmis [25], Observer [97], SIMS [2], Kraft [123], Picsel [66], DWQ [20], AutoMed [19], and InfoMix [82].

The structure of this chapter is as follows. We first describe systems which focus on schema-level information (§3.1). Then, we present systems which exploit both schema-level and instance-level information (§3.2).

3.1 Schema-based systems

Schema-based systems, according to the classification of Chapter 2, are those which rely mostly on schema-level input information for performing ontology matching.

3.1.1 Hovy (University of Southern California)

[72] describes a number of heuristics used to match large-scale ontologies, such as *Sensus* and *Mikrokosmos*, in order to combine them in a single reference ontology. In particular, were used three types of matchers based on (*i*) concept names, (*ii*) concept definitions, and (*iii*) taxonomy structure. For example, the name matcher splits composite-word names into separate words and then compares substrings in order to produce a similarity score. Specifically, the name matcher score is computed as the sum of the square of the number of letters matched, plus 20 points if words are exactly equal or 10 points if end of match coincides. For instance, using this strategy, the comparison between *Free World* and *World* results in 35 points score, while the comparison between *cuisine* and *vine* results in 19 points score. The definition matcher compares the English definitions of two concepts. Here, both definitions are first separated into individual words. Then, the number and the ratio of shared words in two definitions is computed in

order determine the similarity between them. Finally, results of all the matchers are combined based on experimentally obtained formula. The combined scores between concepts from two ontologies are sorted in descending order and are presented to the user for establishing a cutoff value as well as for approving or discarding operations, results of which are saved for later reuse.

3.1.2 TransScm (Tel Aviv University)

TransScm [101] provides data translation and conversion mechanisms between input schemas based on schema matching. First, by using rules, the alignment is produced in a semi-automatic way. Then, this alignment is used to translate data instances of the source schema to instances of the target schema. Input schemas are internally encoded as labeled graphs, where some of the nodes may be ordered. Nodes of the graph represent schema elements, while edges stand for the relations between schema elements or their components. Matching is performed between nodes of the graphs top-down and in one-to-one fashion. Matchers are viewed as rules. For example, (according to the *identical* rule) two nodes match if their labels are found to be synonyms based on the built-in thesaurus; see for a list of the available rules [137]. The system combines rules sequentially based on their priorities. It tries to find for the source node a unique *best* matching target node, or determine a mismatch. In case (*i*) there are a number of matching candidates, among which the system cannot choose the best one, or (*ii*) if the system cannot match or mismatch a source node to a target node with the given set of rules, user involvement is required. In particular, users with the help of a graphic user interface can add, disable or modify rules to obtain the desired matching result. Then, instances of the source schema are translated to instances of the target schema according to the match rules. For the example of the *identical* rule, translation

includes copying the source node components.

3.1.3 DIKE (Università di Reggio Calabria, Università di Calabria)

DIKE (Database Intensional Knowledge Extractor) is a system supporting the semi-automatic construction of cooperative information systems (CISs) from heterogeneous databases [117, 115, 116, 114]. It takes as input a set of databases belonging to the CIS. It builds a kind of mediated schema (called data repository or global structured dictionary) in order to provide a user-friendly integrated access to the available data sources. DIKE focuses on entity-relationship schemas. The matching step is called the extraction of inter-schema knowledge. It is performed in a semi-automatic way. Some examples of inter-schema properties that DIKE can find are *terminological properties*, such as synonyms, homonyms among objects, namely entities and relationships, or type conflicts, e.g., similarities between different types of objects, such as entities, attributes, relationships; *structural properties*, such as object inclusion; *subschema similarities*, such as similarities between schema fragments. With each kind of property is associated its plausibility coefficient in the [0 1] range. The properties with a lower plausibility coefficient than a dynamically derived threshold are discarded, whereas others are accepted. DIKE works by computing sequentially the above mentioned properties. For example, synonyms and homonyms are determined based on information from external resources, such as WordNet, and by analyzing the distances of objects in the neighborhood of the objects under consideration. Also, some weights are used to produce a final coefficient. Then, type conflicts are analyzed and resolved by taking as input the results of synonyms and hyponyms analysis.

3.1.4 SKAT and ONION (Stanford University)

SKAT (Semantic Knowledge Articulation Tool) is a rule-based system that semi-automatically discovers mappings between two ontologies [103]. Internally, input ontologies are encoded as graphs. Rules are provided by domain experts and are encoded in first order logic. In particular, experts specify initially desired matches and mismatches. For example, a rule *President* = *Chancellor*, indicates that we want *President* to be an appropriate match to *Chancellor*. Apart from declarative rules, experts can specify matching procedures that can be used to generate the new matches. Thus, experts have to approve or reject the automatically suggested matches, thereby producing the resulting alignment. Matching procedures are applied sequentially. Some examples of these procedures are: string-based matching, e.g., two terms match if they are spelled similarly, and structure matching, e.g., structural graph slices matching, such as considering nodes near the root of the first ontology against nodes near the root of the second ontology.

ONION (ONtology compositION) is a successor system. It discovers mappings between multiple ontologies semi-automatically. The ultimate goal of matching is to enable a unified query answering over the involved ontologies [104]. Input ontologies (the system handles RDF files) are internally represented as labeled graphs. The alignment is viewed as a set of *articulation rules*. The semi-automated algorithm for resolving the terminological heterogeneity of [102] forms the basis of the *articulation generator*, ArtGen, for the ONION system. ArtGen, in turn, can be viewed as an evolution of the SKAT system with some added matchers. Thus, it executes a set of matchers and suggests articulation rules to the user. A human expert can either accept, modify or delete the suggestions. The expert can also indicate the new matches that the articulation generator might have

missed. ArtGen works sequentially, first by performing linguistic matching and then structure-based matching. During the linguistic matching phase, concept names are represented as sets of words. The linguistic matcher compares all possible pairs of words from any two concepts of both ontologies and assigns a similarity score in [0 1] to each pair. The matcher uses a word similarity table generated by a thesaurus-based or corpus-based matcher called the *word relator* to determine the similarity between pairs of words. The similarity score between two concepts is the average of the similarity scores (ignoring scores of zero) of all possible pairs of words in their names. If this score is higher than a given threshold, ArtGen generates a match candidate. Structure-based matching is performed based on the results of the linguistic matching. It looks for structural isomorphism between subgraphs of the ontologies, taking into account some linguistic clues (see also §3.1.9 for a similar technique). The structural matcher tries to match only the unmatched pairs from the linguistic matching, thereby complementing its results.

3.1.5 Artemis (Università di Milano, Università di Modena e Reggio Emilia)

Artemis (Analysis of Requirements: Tool Environment for Multiple Information Systems) [21] was designed as a module of the MOMIS mediator system [11, 10] for creating global views. It performs affinity-based analysis and hierarchical clustering of source schema elements. Affinity-based analysis represents the matching step: in a sequential manner it calculates the name, structural and global affinity coefficients exploiting a common thesaurus. The common thesaurus is built with the help of ODB-Tools [9], WordNet or manual input. It represents a set of intensional and a set of extensional relationships which depict intra- and inter-schema knowledge about classes and attributes of the input schemas. Based on global

affinity coefficients, a hierarchical clustering technique categorizes classes into groups at different levels of affinity. For each cluster it creates a set of global attributes and the global class. Logical correspondence between the attributes of a global class and source schema attributes is determined through a mapping table.

3.1.6 H-Match (Università degli Studi di Milano)

H-Match [23] is an automated ontology matching system. It was designed to enable knowledge discovery and sharing in the settings of open networked systems, in particular within the Helios peer-to-peer framework [22]. The system handles ontologies specified in OWL. Internally, these are encoded as graphs using the H-model representation [22]. H-Match inputs two ontologies and outputs (one-to-one or one-to-many) correspondences between concepts of these ontologies with the same or closest intended meaning. The approach is based on a similarity analysis through affinity metrics, e.g., term to term affinity, datatype compatibility, and thresholds. H-Match computes two types of affinities (in the [0 1] range), namely *linguistic* and *contextual* affinity. These are then combined by using weighting schemas, thus yielding a final measure, called *semantic* affinity. Linguistic affinity builds on top of a thesaurus-based approach of the Artemis system (§3.1.5). In particular, it extends the Artemis approach (*i*) by building a common thesaurus involving such relations among WordNet synsets as meronymy or coordinate terms, and (*ii*) by providing an automatic handler of compound terms (i.e., those composed by more than one token) that are not available from WordNet. Contextual affinity requires consideration of the neighbor concepts, e.g., linked via taxonomical or mereological relations, of the actual concept.

One of the major characteristics of H-Match is that it can be dynamically configured for adaptation to a particular matching task. Notice that

in dynamic settings complexity of a matching task is not known in advance. This is achieved by means of four matching models. These are: *surface*, *shallow*, *deep*, and *intensive*, each of which involves different types of constructs of the ontology. Computation of a linguistic affinity is a common part of all the matching models. In case of the surface model, linguistic affinity is also the final affinity, since this model considers only names of ontology concepts. All the other three models take into account various contextual features and therefore contribute to the contextual affinity. For example, the shallow model takes into account concept properties, whereas the deep and the intensive models extend previous models by including relations and property values, respectively. Each concept involved in a matching task can be processed according to its own model, independently from the models applied to the other concepts within the same task. Finally, by applying thresholds, correspondences with semantic (final) affinity higher than the cut-off threshold value are returned in the final alignment.

3.1.7 Anchor-Prompt (Stanford Medical Informatics)

Anchor-Prompt [113] is an extension of Prompt, also formerly known as SMART, and is an ontology merging and alignment tool with a sophisticated prompt mechanism for possible matching terms [111]. Prompt handles ontologies expressed in such knowledge representation formalisms as OWL and RDF Schema. Anchor-Prompt is a sequential matching algorithm that takes as input two ontologies, internally represented as graphs and a set of anchors-pairs of related terms, which are identified with the help of string-based techniques, such as edit-distance, or defined by a user or another matcher computing linguistic similarity. Then the algorithm refines them by analyzing the paths of the input ontologies limited by the anchors in order to determine terms frequently appearing in similar posi-

tions on similar paths. Finally, based on the frequencies and user feedback, the algorithm determines matching candidates.

3.1.8 OntoBuilder (Technion Israel Institute of Technology)

OntoBuilder is a system for information seeking on the web [105]. A typical situation the system deals with is, for example, when a user is searching for a car to be rented. Obviously, the user would like to compare prices from multiple providers in order to make an informed decision. Thus, the same input information has to be typed in many times. OntoBuilder operates in two phases, namely: (*i*) ontology creation (the so called *training* phase) and (*ii*) ontology adaptation (the so called *adaptation* phase). During the training phase an initial ontology (in which a user's data needs are encoded) is created by extracting it from a visited web-site of, e.g., *AVIS* car rental company. The adaptation phase includes on-the-fly matching and interactive merging operations of the related ontologies with the actual (initial) ontology. Ontology creation is out of the scope of this thesis. Hence, we concentrate only on the ontology adaptation phase. During the adaptation phase the user suggests the web sites (s)he would like to further explore, e.g., the *Hertz* car rental company. Each such a site goes through the ontology extraction process, thus, resulting in a candidate ontology, which is then merged into the actual ontology. To support this, the best match for each existing term in the actual ontology (to terms from the candidate ontology) is selected. Selection strategy employs thresholds. The matching algorithm works in a term to term fashion. It sequentially executes a number of matchers. Some examples of the matchers used here are removing noisy characters and stop terms and substring matching. If all else fails, a thesaurus look-up is performed. Finally, mismatched terms are presented to the user for manual matching. Some further matchers such as those for precedence matching were introduced in a later work in

[49]. Also top-k mappings as alternative for single best matching (i.e., top-1 category) was proposed in [48].

3.1.9 Cupid (University of Washington, Microsoft Corporation, University of Leipzig)

Cupid [88] implements a sequential algorithm comprising linguistic and structural schema matching techniques, and computing similarity coefficients with the assistance of domain specific thesauri. Input schemas are encoded as graphs. Nodes represent schema elements and are traversed in a combined bottom-up and top-down manner. The matching algorithm consists of three phases and operates only with tree-structures, to which non-tree cases are reduced. The first phase (linguistic matching) computes linguistic similarity coefficients between schema element names (labels) based on morphological normalisation, categorization, string-based techniques, such as common prefix, suffix tests, and thesauri look-up. The second phase (structural matching) computes structural similarity coefficients weighted by leaves which measure the similarity between contexts in which elementary schema elements occur. The third phase (mapping elements generation) aggregates the results of the linguistic and structural matching through a weighted sum and generates a final alignment by choosing pairs of schema elements with weighted similarity coefficients which are higher than a threshold.

3.1.10 COMA and COMA++ (University of Leipzig)

COMA (COmbination of MAtching algorithms) [31] is a schema matching tool based on parallel composition of matchers. It provides an extensible library of matching algorithms, a framework for combining obtained results, and a platform for the evaluation of the effectiveness of the different matchers. As in [31], COMA contains six elementary matchers, five hybrid (i.e., combinations of elementary methods) matchers, and one reuse-oriented matcher. Most of them implement string-based techniques, such as affix, n-gram, edit distance; others share techniques with Cupid (thesauri look-up, etc.). Reuse-oriented is an original matcher, which tries to reuse previously obtained results for entire new schemas or for its fragments. Schemas are internally encoded as directed acyclic graphs, where elements are the paths. This aims at capturing contexts in which the elements occur. Distinct features of the COMA tool in respect to Cupid are a more flexible architecture and the possibility of performing iterations in the matching process. It presumes interaction with users who approve obtained matches and mismatches to gradually refine and improve the accuracy of match. COMA++ is built on top of COMA by elaborating in more detail the alignment reuse operation, provides a more efficient implementation of the COMA algorithms and a graphical user interface [31, 29].

3.1.11 Similarity Flooding (Stanford University, University of Leipzig)

The Similarity Flooding (SF) [94] approach is based on the ideas of similarity propagation. Schemas are presented as directed labeled graphs; grounding on the OIM specification [92]. The algorithm manipulates them in an iterative fix-point computation to produce an alignment between the nodes of the input graphs. The technique starts from string-based

comparison, such as common prefix, suffix tests, of the vertices labels to obtain an initial alignment which is refined within the fix-point computation. The basic concept behind the similarity flooding algorithm is the similarity spreading from similar nodes to the adjacent neighbors through propagation coefficients. From iteration to iteration the spreading depth and a similarity measure are increasing till the fix-point is reached. The result of this step is a refined alignment which is further filtered to finalize the matching process.

3.1.12 CtxMatch and CtxMatch2 (University of Trento, ITC-IRST)

CtxMatch [17, 16, 18] represents the first instantiation of the semantic matching approach [56]. It translates the ontology matching problem into the logical validity problem and computes logical relations, such as equivalence, subsumption between concepts and properties. CtxMatch is a sequential system. At the element level it uses only WordNet to find initial matches for classes as well as for properties. At the structure level, it exploits description logic reasoners, such as Pellet[1] or FaCT[2] to compute the final alignment in a way similar to what is presented in Chapter 2 when discussing methods based on description logics.

3.1.13 DCM framework (University of Illinois at Urbana-Champaign)

MetaQuerier [24] is a middleware system that assists users in finding and quering multiple databases on the web. It exploits the Dual Correlation Mining (DCM) matching framework to facilitate source selection according to user search keywords [70]. Unlike other works, the given approach takes

[1] http://www.mindswap.org/2003/pellet/

[2] http://www.cs.man.ac.uk/~horrocks/FaCT

as input multiple schemas and returns alignments between all of them. This setting is called *holistic* schema matching. DCM automatically discovers complex mappings, e.g., {*author*} corresponds to {*first name, last name*}, between attributes of the web query interfaces in the same domain of interest, e.g., books. As the name (DCM) indicates, schema matching is viewed as *correlation mining*. The idea is that co-occurrence patterns often suggest complex matches. That is, *grouping attributes*, such as *first name* and *last name*, tend to co-occur in query interfaces. Technically, this means that those attributes are positively correlated. Contrary, attribute names which are synonyms, e.g., *quantity* and *amount*, rarely co-occur, thus representing an example of negative correlation between them. Matching is performed in two phases. During the first phase (matching discovery), a set of matching candidates is generated by mining first positive and then negative correlations among attributes and attribute groups. Also, some thresholds and a specific correlation measure such as the H-measure are used. During the second phase (matching construction), by applying some strategies of ranking, e.g., scoring function, rules, and selection, such as iterative greedy selection, the final alignment is produced.

3.2 Mixed systems

The following systems take advantage of both schema-level and instance-level input information if they are both available.

3.2.1 SEMINT (Northwestern University, NEC, The MITRE Corporation)

SEMantic INTegrator (SEMINT) is a tool based on neural networks to assist in identifying attribute correspondences in heterogeneous databases [83, 84]. It supports access to a variety of database systems and utilizes

both schema- and instance-level information to produce rules for matching corresponding attributes automatically. The approach works as follows. First, it extracts from two databases all the necessary information (features or discriminators) which is potentially available and useful for matching. This includes normalized schema information, e.g., field specifications, such as datatypes, length, constraints, and statistics about data values, e.g., character patterns, such as ratio of numerical characters, ratio of white spaces, and numerical patterns, such as mean, variance, standard deviation. Second, by using a neural network as a classifier (self-organizing map algorithm), it groups the attributes based on similarity of the features for a single (the first) database. Then, it uses a back-propagation neural network for learning and recognition. Based on the previously obtained clusters, the learning is performed. Finally, using a trained neural network on the first database features and clusters, the system recognizes and computes similarities between the categories of attributes from the first database and the features of attributes from the second database, thus, generating a list of match candidates, which are to be inspected and confirmed or discarded by a human user.

3.2.2 Clio (IBM Almaden, University of Toronto)

Clio is a system for managing and facilitating data transformation and integration tasks within heterogeneous environments [99, 100, 106, 69]. Clio handles relational and XML schemas. As a first step, the system transforms the input schemas into an internal representation, which is a nested relational model. The Clio approach is focused on making the alignment operational. It is assumed that the matching step (namely, identification of the so-called *value correspondences*) is performed with the help of a schema matching component or manually by the user. The built-in schema matching algorithm of Clio combines in a sequential manner instance-based

attribute classification via a variation of a Naive Bayes classifier [67, 36, 90] and string matching between elements names, e.g., by using edit distance measure. Then, taking the n:m value correspondences (the alignment) together with constraints coming from the input schemas, Clio compiles these into an internal query graph representation. In particular, an interpretation of the input correspondences is given. Thus, a set of logical mappings with formal semantics is produced. To this end, Clio also supports mappings composition [47]. Finally, the query graph can be serialized into different query languages, e.g., SQL, XSLT, XQuery, thus enabling actual data to be moved from a source to a target, or to answer queries. The system, besides trivial transformations, aims at discovering complex ones, such as the generation of keys, references, join conditions.

3.2.3 NOM and QOM (University of Karlsruhe)

NOM (Naive Ontology Mapping) [40] and QOM (Quick Ontology Mapping) [39] are components of the FOAM framework [37].

NOM adopts the idea of parallel composition of matchers from COMA (§3.1.10). Some innovations with respect to COMA are in the set of elementary matchers based on rules, exploiting explicitly codified knowledge in ontologies, such as information about super- and sub-concepts, super- and sub-properties, etc. At present the system supports 17 rules. For example, a rule states that if super-concepts are the same, the actual concepts are similar to each other. These rules use many terminological and structural techniques.

QOM (Quick Ontology Mapping) [39] is a variation of the NOM system dedicated to improve the efficiency of the system. The approach is based on the idea that the loss of quality in matching algorithms is marginal (to a standard baseline); however improvement in efficiency can be tremendous. This fact allows QOM to produce correspondences fast, even for large-

size ontologies. QOM is grounded on matching rules of NOM. However, for the purpose of efficiency the use of some rules, e.g., the rules that traverse the taxonomy, have been restricted. QOM avoids the complete pairwise comparison of trees in favor of a (n incomplete) top-down strategy, thereby focusing only on promising matching candidates. The similarity measures produced by basic matchers (matching rules) are refined by using a sigmoïd function, thereby emphasizing high individual similarities and de-emphasizing low individual similarities. They are then aggregated through weighted average. Finally, with the help of thresholds, the final alignment is produced.

3.2.4 OLA (INRIA Rhône-Alpes, Université de Montréal)

OLA (OWL Lite Aligner) [46] is an ontology matching system which is designed with the idea of balancing the contribution of each of the components that compose an ontology, e.g., classes, constraints, data instances. OLA handles ontologies in OWL. It first compiles the input ontologies into graph structures, unveiling all relationships between entities. These graph structures produce the constraints for expressing a similarity between the elements of the ontologies. The similarity between nodes of the graphs follows two principles: (i) it depends on the category of node considered, e.g., class, property, and (ii) it takes into account all the features of this category, e.g., superclasses, properties.

The distance between nodes in the graph are expressed as a system of equations based on string-based, language-based and structure-based similarities. These distances are almost linearly aggregated (they are linearly aggregated modulo local matches of entities). For computing these distances, the algorithm starts with base distance measures computed from labels and concrete datatypes. Then, it iterates a fix-point algorithm until no improvement is produced. From that solution, an alignment is gener-

ated which satisfies some additional criterion on the alignment obtained and the distance between matched entities.

3.2.5 Corpus-based matching (University of Washington, Microsoft Research, University of Illinois at Urbana-Champaign)

[87] proposed an approach to schema matching which, besides input information available from schemas under consideration, also exploits some domain specific knowledge via an external corpus of schemas and mappings. The intuition behind the approach is based on the use of corpus in information retrieval, where similarity between queries and concepts is determined based on analyzing large corpora of text. In schema matching settings, such a corpus can be initialized with a small number of schemas obtained, for example, by using available standard schemas in the domain of interest (see, for instance, XML.org[3] and OASIS.org[4]) and should eventually evolve in time with new matching tasks.

Since the corpus is intended to have a number of different representations of each concept in the domain, it should facilitate learning these variations in the elements and their properties. The corpus is exploited in two ways. First, to obtain an additional evidence about each element being matched by including evidence from similar elements in the corpus. Second, in the corpus, similar elements are clustered and some statistics for clusters are computed, such as neighborhood and ordering of elements. These statistics are ultimately used to build constraints that facilitate selection of the correspondences in the resulting alignment.

The approach handles web forms and relational schemas and focuses on one-to-one alignments. It works in two logical phases. Firstly, schemas

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

[4] http://www.oasis-open.org/

under consideration are matched against the corpus, thereby augmenting these with possible variations of their elements based on knowledge available from the corpus. Secondly, augmented schemas are matched against each other. In both cases the same set of matchers is applied. In particular, basic matchers, called learners, include: (*i*) a *name learner*, (*ii*) a *text learner*, (*iii*) a *data instance learner*, and (*iv*) a *context learner*. These matchers mostly follow the ideas of techniques used in LSD [32] and Cupid (§3.1.9). For example, the *name learner* exploits names of elements. It applies tokenization and n-grams to the names in order to create training examples. The matcher itself is a text classifier, such as Naive Bayes. In addition, the name learner, in order to determine similarity between element names string, uses edit distance. The *data instance learner* determines whether the values of instances share common patterns, same words, etc. Also, a matcher for an automatic combination of the results produced by basic matchers, called *metalearner*, uses *logistic regression* with the help of *stacking* technique [133] in order to learn its parameters. Finally, by using constraints obtained based on the statistics from the corpus, some filtering of the candidate correspondences is performed in order to produce the final alignment.

3.3 Summary

The panorama of systems considered in this chapter has multiplied the diversity of basic techniques by the variety of strategies for combining them introduced in the previous chapter. However, there are a number of constant features that are shared by the majority of systems. Also, usually each individual system innovates on a particular aspect. Let us summarize some global observations concerning the presented systems:

- Most of the systems under consideration deal with particular ontol-

ogy types, such as DTDs, relational schemas and OWL ontologies. Only a small number of systems aim at being generic, i.e., handle multiple types of ontologies. Some examples include Cupid (§3.1.9), COMA and COMA++ (§3.1.10), Similarity Flooding (§3.1.11) and the approach proposed in this thesis.

- Most of the approaches take as input a pair of ontologies, including the approach proposed in this thesis, while only a small number of systems take as input multiple ontologies, e.g., DCM (§3.1.13).

- Most of the approaches handle only tree-like structures, including the approach proposed in this thesis, while only a small number of systems handle graphs. Some examples of the latter include Cupid (§3.1.9), COMA and COMA++ (§3.1.10), and OLA (§3.2.4).

- Most of the systems focus on discovery of one-to-one alignments, while only a small number of systems have tried to address the problem of discovering more complex correspondences, such as one-to-many, e.g., the approach proposed in this thesis, and many-to-many, e.g., DCM (§3.1.13).

- Most of the systems focus on computing confidence measures in [0 1] range, most often standing for the fact that the equivalence relation holds between ontology entities. Only a small number of systems compute logical relations between ontology entities, such as equivalence, subsumption. Some examples of the latter include CtxMatch (§3.1.12) and the approach proposed in this thesis.

Table 3.1 summarizes how some of the matching systems considered in this chapter cover the solution space in terms of the classification of Chapter 2.

Table 3.1: Basic matchers used by different systems

	Element-level		Structure-level	
	Syntactic	**External**	**Syntactic**	**Semantic**
Hovy §3.1.1	string-based, language-based	-	taxonomic structure	-
TranScm §3.1.2	string-based	built-in thesaurus	taxonomic structure, matching of neighbourhood	-
DIKE §3.1.3	string-based, domain compatibility	WordNet	matching of neighbourhood	-
SKAT §3.1.4	string-based	auxiliary thesaurus, corpus-based	taxonomic structure, matching of neighbourhood	-
Artemis §3.1.5	domain compatibility, language-based	common thesaurus (CT)	matching of neighbours via CT, clustering	-
H-Match §3.1.6	domain compatibility, language-based, domains and ranges	common thesaurus (CT)	matching of neighbours via CT, relations	-
Anchor-Prompt §3.1.7	string-based, domains and ranges	-	bounded paths matching: (arbitrary links), taxonomic structure	-
OntoBuilder §3.1.8	string-based, language-based	thesaurus look up	-	-
Cupid §3.1.9	string-based, language-based, datatypes, key properties	auxiliary thesauri	tree matching weighted by leaves	-
COMA & COMA++ §3.1.10	string-based, language-based, datatypes	auxiliary thesauri, alignment reuse, repository of structures	DAG (tree) matching with a bias towards various structures (e.g., leaves)	-
SF §3.1.11	string-based, datatypes, key properties	-	iterative fix-point computation	-
CtxMatch §3.1.12	string-based, language-based	WordNet	-	based on description logics
DCM §3.1.13	-	-	correlation mining	-
SEMINT §3.2.1	neural network, datatypes, value patterns	-	-	-
Clio §3.2.2	string-based, language-based, Naive Bayes	-	-	-
NOM & QOM §3.2.3	string-based, domains and ranges	application-specific vocabulary	matching of neighbours, taxonomic structure	-
OLA §3.2.4	string-based, language-based, datatypes	WordNet	iterative fix-point computation, matching of neighbours, taxonomic structure	-

For example as from Table 3.1, OLA (§3.2.4) exploits string-based element-level matchers, a matcher based on WordNet, iterative fix-point computation, etc. Table 3.1 also testifies that ontology matching research so far was mainly focused on syntactic and external techniques. In fact, many systems rely on the same string-based techniques. Similar observation can be also made concerning the use of WordNet as an external resource of common knowledge. In turn, semantic techniques have rarely been exploited, this is only done by the approach proposed in this thesis and CtxMatch (§3.1.12).

Having considered some of the recent schema-based matching systems, it is important to notice that the matching operation typically constitutes only one of the steps towards the ultimate goal of, e.g., ontology integration, data integration, and web service composition. To this end, we would like to mention some existing infrastructures, which use matching as one of its integral components. Some examples include: Chimaera [91], MAFRA [27, 86], Rondo and Moda [96, 95, 93], Prompt Suite [112, 110], Alignment API [42], GeRoMe [76], Protoplasm [14], COMA++ [31, 29] and ModelGen [3, 4]. The goal of such infrastructures is to enable a user with a possibility of performing such high-level tasks, e.g., given a product request expressed in terms of the catalog $C1$, return the products satisfying the request from the marketplaces $MP1$ and $MP2$. Moreover, use matching component $M5$, and translate instances by using component $T3$.

Part III

Semantic matching algorithms

Chapter 4

S-Match algorithm

This chapter presents the semantic matching algorithm and expands on the technical details of its steps.

Material presented in this chapter has been developed in collaboration with Pavel Shvaiko and published in [57, 64, 58]. The semantic matching approach presented in this chapter have been developed by Pavel Shvaiko. See [56] for details.

In this chapter we first discuss the semantic matching by intuitions and examples as well as we state the problem technically (§4.1). Then we provide the main macro steps of the algorithm realizing the semantic matching approach (§4.2-4.5).

4.1 Semantic matching

In our approach, we assume that all the data and conceptual models (e.g., classifications, database schemas, ontologies) can be generally represented as graphs (see [56] 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 [88] and COMA [31]. We focus on tree-like structures, e.g., 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 [14]. 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 schemas: *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[1]. The relations are ordered according 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. The semantics of the above relations are 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 where ID_{ij} is a unique identifier of the given mapping element; a_i is the

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

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.

Let us summarize the algorithm for semantic matching via a running example. We consider small academic courses classifications shown in Figure 4.1.

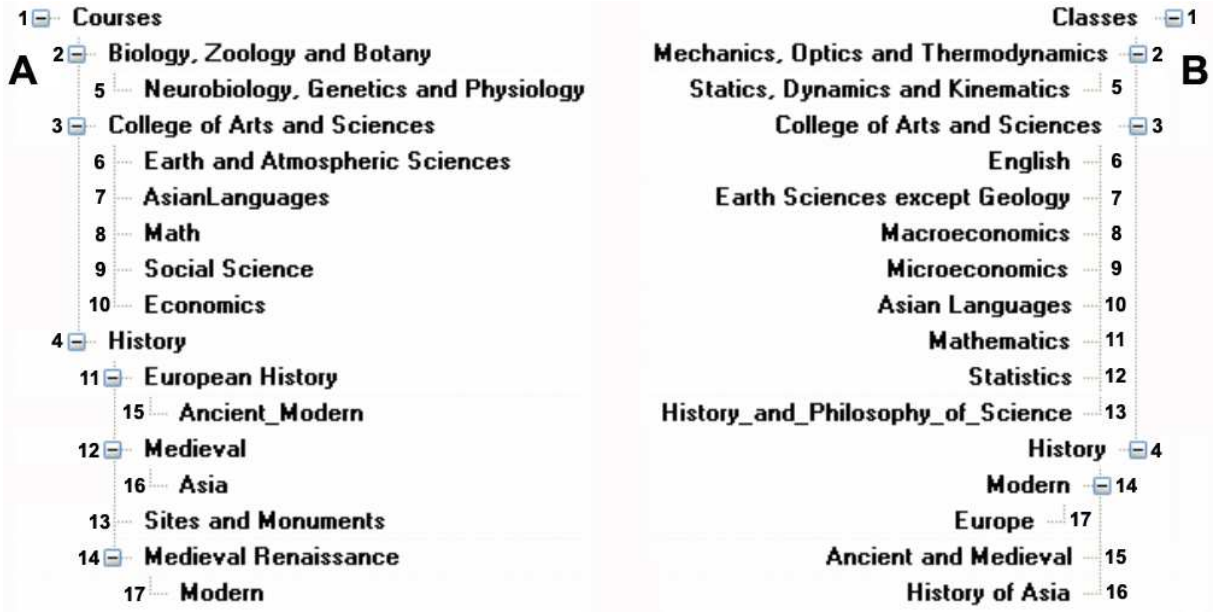

Figure 4.1: Parts of two classifications devoted to academic courses

Let us introduce some notation (see also Figure 4.1). Numbers are the unique identifiers of nodes. We use "C" for concepts of labels and

concepts at nodes. Thus, for example, in the tree A, $C_{History}$ and C_4 are, respectively, the concept of the label $History$ and the concept at node 4. Also, 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 takes as input two schemas 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, independently of the specific matching problem. *Step 3* and *Step 4* can only be done at run time, once the two trees which must be matched have been chosen. We also refer in the remainder of the thesis to the element level matching (Step 3) as *label matching* and to the structure level matching (Step 4) as *node matching*.

We view labels of nodes as concise descriptions of the data that is stored under the nodes. During *Step 1*, 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 [98][2]), and by returning as output a *concept of the label*. Thus, for example, by writing $C_{History}$ we move from the natural language ambiguous label $History$ to the concept $C_{History}$, which codifies explicitly its intended meaning, namely the data (documents) which are about history.

During *Step 2* 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 [54]. Thus, 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.

Step 3 is concerned with acquisition of "world" knowledge. Relations between concepts of labels are computed with the help of a library of element level semantic matchers. These matchers take as input two concepts of labels and produce as output a semantic relation (e.g., equivalence, more/less general) between them. For example, from WordNet [98] we can derive that *course* and *class* are synonyms, and therefore, $C_{Courses} \equiv C_{Classes}$.

Step 4 is concerned with the computation of the relations between concepts at nodes. This problem cannot be resolved by exploiting static knowledge sources only. We have (from Step 3) background knowledge, codified as a set of relations between concepts of labels occurring in two trees. This knowledge constitutes the background theory (axioms) within which we reason. We need to find a semantic relation (e.g., equivalence, more/less general) between the concepts at any two nodes in two trees. However,

[2]WordNet is a lexical database for English. It is based on synsets (or senses), namely structures containing sets of terms with synonymous meanings.

these are usually complex concepts obtained by suitably combining the corresponding concepts of labels. For example, suppose we want to find a relation between C_4 in the tree A (which, intuitively, stands for the concept of courses of history) and C_4 in the tree B (which, intuitively, stands for the concept of classes of history). In this case, we should realize that they have the same extension, and therefore, that they are equivalent.

4.2 Step 1: Concepts of labels computation

Technically, the main goal of Step 1 is to automatically translate ambiguous natural language labels taken from the schema elements' names into an internal logical language. We use a propositional description logic language (L^C) for several reasons. First, given its set-theoretic interpretation, it "maps" naturally to the real world semantics. Second, natural language labels, e.g., in classifications, are usually short expressions or phrases having simple structure. These phrases can often be converted into a formula in L^C with no or little loss of meaning [55]. Third, a formula in L^C can be converted into an equivalent formula in a propositional logic language with boolean semantics. Apart from the atomic propositions, the language L^C includes logical operators, such as conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$). There are also comparison operators, namely more general ($\sqsupseteq$), less general ($\sqsubseteq$), and equivalence ($\equiv$). The interpretation of these operators is the standard set-theoretic interpretation. We compute concepts of labels according to the following four logical phases, being inspired by the work in [89].

1. *Tokenization.* Labels of nodes are parsed into tokens by a tokenizer which recognizes punctuation, cases, digits, stop characters, etc. Thus, for instance, the label History and Philosophy of Science becomes his-

tory, and, philosophy, of, science. The multiword concepts are then recognized. At the moment the list of all multiword concepts in WordNet [98] is exploited here together with a heuristic which takes into account the natural language connectives, such as and, or, etc. For example, Earth and Atmospheric Sciences becomes earth sciences, and, atmospheric, sciences since WordNet contains senses for earth sciences, but not for atmospheric sciences.

2. *Lemmatization.* Tokens of labels are further lemmatized, namely they are morphologically analyzed in order to find all their possible basic forms. Thus, for instance, sciences is associated with its singular form, science. Also here we discard from further considerations some predefined meaningless (in the sense of being useful for matching) words, articles, numbers, and so on.

3. *Building atomic concepts.* 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.

4. *Building complex concepts.* When existing, all tokens that are prepositions, punctuation marks, conjunctions (or strings with similar roles) are translated into logical connectives and used to build complex concepts out of the atomic concepts constructed in the previous phase. Thus, for instance, commas and conjunctions are translated into logical disjunctions, prepositions, such as of and in, are translated into logical conjunctions, and words like except, without are translated into negations. Thus, for example, the concept of label *History and Philosophy of Science* is computed as $C_{History\ and\ Philosophy\ of\ Science} = (C_{History} \vee C_{Philosophy}) \wedge C_{Science}$, where $C_{Science} = \langle science, senses_{WN}\#2 \rangle$ is taken to be the union of two WordNet senses, and similarly for *his-*

tory and *philosophy*. Notice that natural language *and* is converted into logical disjunction, rather than into conjunction (see [89] for detailed discussion and justification for this choice).

The result of Step 1 is the logical formula for concept of label. It is computed as a full propositional formula were literals stand for atomic concepts of labels.

In Algorithm 1 we present the pseudocode which provides an algorithmic account of how concepts of labels are built. In particular, the *buildCLab* function takes the tree of nodes context and constructs concepts of labels for each node in the tree. The nodes are preprocessed in the main loop in lines 220-350. Within this loop, first, the node label is obtained in line 240. Then, it is tokenized and lemmatized in lines 250 and 260, respectively. The (internal) loop on the lemmas of the node (lines 270-340) starts from stop words test in line 280. Then, WordNet is queried. If the lemma is in WordNet, its senses are extracted. In line 300, atomic concept of label is created and attached to the node by the *addACOLtoNode* function. In the case when WordNet returns no senses for the lemma, the special identifier *SENSES_NOT_FOUND* is attached to the atomic concept of label[3]. The propositional formula for the concept of label is iteratively constructed by *constructcLabFormula* (line 340). Finally, the logical formula is attached to the concept at label (line 350) and some sense filtering is performed by *elementLevelSenseFiltering*[4].

The pseudo code of Algorithm 2 illustrates the sense filtering technique. It is used in order to filter out the irrelevant (for the given matching task) senses from concepts of labels. In particular, we look whether the senses of atomic concepts of labels within each concept of a label are con-

[3]This identifier is further used by element level semantic matchers in Step 3 of the matching algorithm in order to determine the fact that the label (lemma) under consideration is not contained in WordNet, and therefore, there are no senses in WordNet for a given concept.

[4]The sense filtering problem is also known under the name of word sense disambiguation (WSD).

Algorithm 1 Concept of label construction pseudo code

```
Node: struct of
        int nodeId;
        String label;
        String cLabel;
        String cNode;
        AtomicConceptAtLabel[ ] ACOLs;
AtomicConceptOfLabel struct of
        int id;
        String token;
        String[ ] wnSenses;
200. void buildCLab(Tree of Nodes context)
210. String[ ] wnSenses;
220. for each node ∈ context
230.    String cLabFormula=" ";
240.    String nodeLabel=getLabel(node);
250.    String[ ] tokens=tokenize(nodeLabel);
260.    String[ ] lemmas=lematize(tokens);
270.    for each lemma ∈ lemmas
280.      if (isMeaningful(lemma))
290.          if (!isInWordnet(lemma))
300.          addACOLtoNode(node, lemma, SENSES_NOT_FOUND);
310.      else
320.          wnSenses= getWNSenses(token);
330.          addACOLtoNode(node, lemma, wnSenses);
340.    cLabFormula=constructcLabFormula(cLabFormula, lemma);
350. setcLabFormula(node, cLabFormula);
360. elementLevelSenseFiltering(node);
```

nected by any relation in WordNet. If so, we discard all other senses from atomic concept of label. Otherwise we keep all the senses. For example, for the concept of label *Sites and Monuments* before the sense filtering step we have $Sites, senses_{WN}\#4 \wedge Monuments, senses_{WN}\#3$. Since the second sense of *monument* is a hyponym of the first sense of *site*, notationally $Monument\#2 \sqsubseteq Site\#1$, all the other senses are discarded. Therefore, as a result of this sense filtering step we have $Sites, senses_{WN}\#1 \wedge Monuments, senses_{WN}\#1$.

elementLevelSenseFiltering takes the node structure as input and discards the irrelevant senses from atomic concepts of labels within the node. In particular, it executes two loops on atomic concept of labels (lines 30-120 and 50-120). WordNet senses for the concepts are acquired in lines 40 and 70. Then two loops on the WordNet senses are executed in lines 80-120 and 90-120. Afterwards, checking whether the senses are connected by a WordNet relation is performed in line 100. If so, the senses are added to a special set, called refined senses set (lines 110, 120). Finally, the WordNet senses are replaced with the refined senses by *saveRefinedSenses*.

4.3 Step 2: Concepts at nodes computation

Concepts at nodes are written in the same propositional description logic 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 [68], 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: $C_4 = C_{Courses} \wedge C_{History}$.

Algorithm 2 The pseudo code of element level sense filtering technique

```
10. void elementLevelSenseFiltering(Node node)
20.   AtomicConceptOfLabel[ ] nodeACOLs=getACOLs(node);
30.   for each nodeACOL ∈ nodeACOLs
40.     String[ ] nodeWNSenses=getWNSenses(nodeACOL);
50.     for each ACOL ∈ nodeACOLs
60.       if (ACOL!=nodeACOL)
70.         String[ ] wnSenses=getWNSenses(ACOL);
80.           for each nodeWNSense ∈ nodeWNSenses
90.             for each wnSense ∈ wnSenses
100.              if (isConnectedbyWN(nodeWNSense, focusNodeWNSense))
110.                addToRefinedSenses(nodeACOL,nodeWNSense);
120.                                addToRefinedSenses(focusNodeACOL, focusNodeWNSense);
130.          saveRefinedSenses(context);

140.  void saveRefinedSenses(context)
150.    for each node ∈ context
160.      AtomicConceptOfLabel[ ] nodeACOLs=getACOLs(node);
170.      for each nodeACOL ∈ NodeACOLs
180.        if (hasRefinedSenses(nodeACOL))
190.          //replace original senses with refined
```

Further in the thesis we require the concepts at nodes to be consistent (satisfiable). The reasons for their inconsistency are negations in atomic concepts of labels. For example, natural language label *except_geology* is translated into the following logical formula $C_{except\_geology} = \neg C_{geology}$. Therefore, there can be a concept at node represented by a formula of the following type $C_{geology} \wedge \ \wedge \neg C_{geology}$, which is inconsistent. In this case the user is notified that the concept at node formula is unsatisfiable and asked to decide a more important branch, i.e., (s)he can choose what to delete from the tree, namely $C_{geology}$ or $C_{except\_geology}$. Notice that this does not sacrifice the system performance since this check is made within the preprocessing (i.e., offline, when the tree is edited)[5]. Let us consider the following example: $C_N = C_{Medieval} \wedge C_{Modern}$. Here, concept at node formula contains two concepts of labels, which are as from WordNet disjoint. Intuitively, this means that the context talks about either *Medieval* or *Modern* (or there is implicit disjunction in the concept at node formula). Therefore, in such cases, the formula for concept at node is rewritten in the following way: $C_N = (C_{Medieval} \vee C_{Modern})$....

The pseudo code of the second step is presented as Algorithm 3. The *buildCNode* function takes as an input the tree of nodes with precomputed concepts of labels and computes as output the concept at node for each node in the tree. The sense filtering (line 620) is performed by *structureLevelSenseFiltering* in the way similar to the sense filtering approach used at the element level (as discussed in Algorithm 2). Then, the formula for the concept at node is constructed within *buildcNodeFormula* as conjunction of concepts of labels attached to the nodes in the path to the root. Finally, the formula is checked for unsatisfiability (line 640). If so, user is asked about the possible modifications in the tree structure or they are

[5]In general case the reasoning is as costly as in the case of propositional logic (i.e., deciding unsatisfiability of the concept is co-NP hard). In many real world cases (see [63] for more details) the corresponding formula is Horn. Thus, its satisfiability can be decided in linear time.

applied automatically, specifically implicit disjunctions are added between disjoint concepts (line 650).

Algorithm 3 Concepts at nodes construction pseudo code

```
600. void buildCNode(Tree of Node context)
610.   for each node ∈ context
620.     structureLevelSenseFiltering(node,context);
630.     String cNodeFormula=buildcNodeFormula(node, context);
640.     if (isUnsatisifiable(cNodeFormula))
650.       updateFormula(cNodeFormula);
```

Let us discuss how the structure level sense filtering operates. As noticed before, this technique is similar to the one described in Algorithm 2. The major difference is that the senses now are filtered not within the node label but within the tree structure. For all concepts of labels we collect all their ancestors and descendants. We call them a focus set. Then, all WordNet senses of atomic concepts of labels from the focus set are compared with the senses of the atomic concepts of labels of the concept. If a sense of atomic concept of label is connected by a WordNet relation with the sense taken from the focus set, then all other senses of these atomic concepts of labels are discarded. Therefore, as a result of sense filtering step we have (i) the WordNet senses which are connected with any other WordNet senses in the focus set or (ii) all the WordNet senses otherwise. After this step the meaning of concept of labels is reconciled with respect to the knowledge residing in the tree structure. The pseudo code of Algorithm 4 provides an algorithmic account of the structure level sense filtering procedure.

The *structureLevelSenseFiltering* function takes a node and a tree of nodes as input and refines the WordNet senses within atomic concepts of labels in the node with respect to the tree structure. First, atomic concepts at labels from the ancestor and descendant nodes are gathered into the focus set (line 420). Then, a search for pairwise relations between the senses attached to the atomic concepts of labels is performed (lines

440-520). These senses are added to the refined senses set (lines 530-540) and further *saveRefinedSenses* from Algorithm 4 is applied (line 550) in order to save the refined senses.

Algorithm 4 The pseudo code of structure level sense filtering technique

```
400.void structureLevelSenseFiltering(Node node, Tree of Nodes context)
410.    AtomicConceptOfLabel[ ] focusNodeACOLs;
420.    Node[ ] focusNodes=getFocusNodes(node, context);
430.    AtomicConceptOfLabel[ ] nodeACOLs=getACOLs(node);
440.    for each nodeACOL ∈ nodeACOLss
450.      String[ ] nodeWNSenses=getWNSenses(nodeACOL);
460.      for each nodeWNSense ∈ nodeWNSenses
470.        for each focusNode ∈ focusNodes
480.          focusNodeACOLs=getACOLs(focusNode);
490.          for each focusNodeACOL ∈ focusNodeACOLs
500.            String[ ] fNodeWNSenses=getWNSenses(focusNodeACOL);
510.            for each fNodeWNSense ∈ nodeWNSenses
520.              if (isConnectedbyWN(nodeWNSense, fNodeWNSense))
530.                addToRefinedSenses(nodeACOL,nodeWNSense);
540.                addToRefinedSenses(focusNodeACOL, focusNodeWNSense);
550.    saveRefinedSenses(context);
```

4.4 Step 3: Label Matching

4.4.1 A library of label matchers

Include detailed description of element level matchers Relations between concepts of labels are computed with the help of a library of element level semantic matchers [62]. These matchers take as input two atomic concepts of labels and produce as output a semantic relation between them. Some of them are reimplementations of well-known matchers used in Cupid [88] and COMA [31]. 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.

Table 4.1: Element level semantic matchers.

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

Our label matchers are briefly summarized in Table 4.1. 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 WordNet. 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 sense filtering techniques to discard the irrelevant senses, see Sections 4.2 and 4.3 for details. The relations produced by a matcher with the second approximation level are likely to be correct (e.g., $net \equiv network$, but $hot \equiv hotel$ by Prefix). The relations produced by a matcher with the third approximation level depend heavily on the context of the matching task (e.g., $cat \equiv dog$ by Extended gloss comparison in the sense that they are both pets). Note, matchers by default 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-*, *sense-* and *gloss-based matchers*. String-based matchers exploit string comparison tech-

niques in order to produce the semantic relation, while sense-based exploit the structural properties of the WordNet hierarchies and gloss-based compare two textual descriptions (glosses) of WordNet senses. Below, we discuss in detail some matchers from each of these categories.

4.4.2 Prefix

Prefix is a string based matcher. It 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 are summarized in Table 4.2. Prefix is efficient in matching cognate words and similar acronyms

Table 4.2: Semantic relations produced by prefix matcher

Source label	Target label	Semantic relation
net	network	=
hot	hotel	=
cat	core	Idk

(e.g., *RDF* and *RDFS*) but often syntactic similarity does not imply semantic relatedness. Consider the examples in Table 4.2. 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*.

4.4.3 Suffix

Suffix is a string based matcher. It checks whether one input label ends with the other. It returns the equivalence relation in this case and Idk otherwise. The results produced by Suffix are summarized in Table 4.3. 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*).

Table 4.3: Semantic relations produced by suffix matcher

Source label	Target label	Semantic relation
phone	telephone	=
word	sword	=
door	floor	Idk

4.4.4 Edit Distance

Edit distance is a string based matcher. It 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. Edit Distance is

Table 4.4: Semantic relations produced by edit distance matcher

Source label	Target label	Semantic relation
street	street1	=
proper	propel	=
owe	woe	Idk

useful with some unknown 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.

4.4.5 nGram

NGram is a string based matcher. It 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 are summarized in Table 4.5.

Table 4.5: Semantic relations produced by nGram matcher

Source label	Target label	Semantic relation
address	address1	=
behavior	behaviour	=
door	floor	Idk

4.4.6 WordNet

WordNet [98] is a lexical database which is available online 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*. The relations of WordNet 2.0 are presented on Table 4.6.

Figure 4.2 shows an example of nouns taxonomy.

WordNet matcher is a knowledge based matcher. It translates the relations provided by WordNet to semantic relations according to the following rules:

- $A \sqsubseteq B$ if A is a hyponym, meronym or troponym of B;

Table 4.6: Possible relationships in WordNet

Relation	Description	Example
Hypernym	is a generalization of	*motor vehicle* is a hypernym of *car*
Hyponym	is a kind of	*car* is a hyponym of *motor vehicle*
Meronym	is a part of	*lock* is a meronym of *door*
Holonym	contains part	*door* is a holonym of *lock*
Troponym	is a way to	*fly* is a troponym of *travel*
Antonym	opposite of	*stay in place* is an antonym of *travel*
Attribute	attribute of	*fast* is an attribute of *speed*
Entailment	entails	*calling on the phone* entails *dialing*
Cause	cause to	*to hurt* causes *to suffer*
Also See	related verb	*to lodge* is related to *reside*
Similar to	similar to	*evil* is similar to *bad*
Participle of	is participle of	*stored* is the participle of *to store*
Pertainym	pertains to	*radial* pertains to *radius*

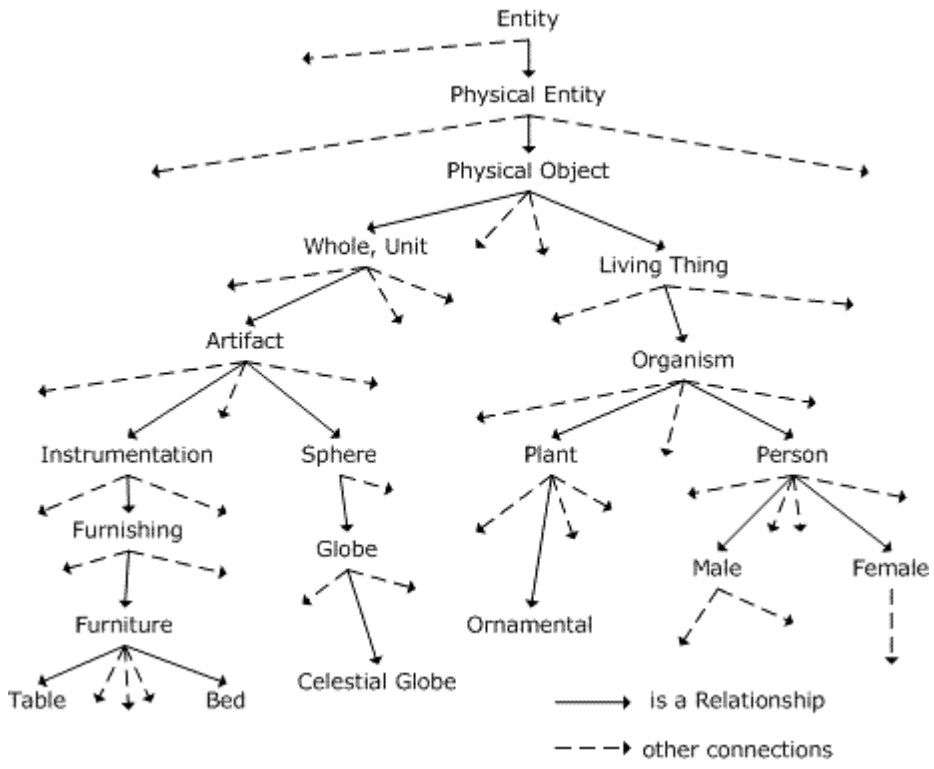

Figure 4.2: An example of WordNet nouns taxonomy

- $A \sqsupseteq B$ 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.

Notice that hyponymy, meronymy, troponymy, hypernymy and holonymy relations are transitive. Therefore, for example, from Figure 4.2 we can derive that $Person \sqsubseteq LivingThing$.

If none of the abovementioned relations holds among the two input synsets Idk relation is returned.

Table 4.7 illustrates WordNet matcher results.

Table 4.7: Semantic relations produced by WordNet matcher

Source label	Target label	Semantic relation
car	minivan	$\sqsupseteq$
car	auto	$=$
tail	dog	$\sqsubseteq$
red	pink	Idk

4.4.7 Leacock Chodorow Matcher

Leacock Chodorow matcher is a knowledge based matcher. It exploits Leacock Chodorow semantic similarity measure. It returns $\equiv$ if the measure exceeds the given threshold and Idk otherwise. The measure is based on counting the number of links between two input synsets. Intuitively, the shorter the path, the more related are the concepts under consideration. Leacock and Chodorow[80] considered the noun *is a* hierarchy. They proposed the following formula for estimating the similarity of two synsets:

$$sim_{lc}(c_1, c_2) = -\ln\left(\frac{spath\,(c_1, c_2)}{2 \cdot D}\right) \tag{4.1}$$

where $spath(s_1, s_2)$ is the length of the shortest path between the two synsets c_1 and c_2 and D is the depth of the tree.

The measure has a lower bound of 0 and upper bound $U_b = -\ln(1/(2 \cdot maxDepth))$, where $maxDepth$ is a maximum depth of the taxonomy.

Table 4.8 illustrates Leacock Chodorow matcher results with 3.0 threshold.

Table 4.8: Semantic relations produced by Leacock Chodorow matcher

Source synset	Target synset	Semantic relation
autograph	signature	=
actor	actress	=
dog	cat	Idk
sky	atmosphere	Idk

4.4.8 Resnik Matcher

Resnik matcher is a knowledge based matcher. It exploits Resnik semantic similarity measure. It returns $\equiv$ if the measure exceeds the given threshold and Idk otherwise. This measure is based on the concept of *information content*[126]. Information content defines the generality or specificity of a concept in a certain topic.

Information content of the given concept is calculated as follows. Firstly the frequency[6] of concept occurrences F_C in *text corpus* is calculated. Then the frequencies of all subsuming concepts are calculated and added to F_C. Thus the root concept will count the occurrences of all the concepts in its taxonomy. In the case of WordNet synsets the frequency counts are precomputed for wide range of large scale corpora. In our preliminary experiments we exploited *Brown corpus of standard american english* [77].

[6]Here and further in the paper following to NL tradition we treat frequency as count (i.e., frequency of concept occurrences is a number of times the given concept occurs in the corpora).

Information content of a concept c is defined as:

$$IC(c) = -\ln\left(\frac{freq(c)}{freq(root)}\right) \tag{4.2}$$

where $freq(c)$ and $freq(root)$ are, respectively, the frequencies of the concept c and the *root* of the taxonomy. Note that the fraction represents the probability of occurrence of the concept in a large corpus.

Resnik defines the semantic similarity of the two concepts as the amount of information they share in common. To be more precise, the amount of information two concepts share in common is equal to the value of information content of their *lowest common subsumer*, that is the lowest node in the taxonomy that subsumes both concepts. For example the lowest common subsumer of *cat* and *dog* is *carnivore*. Therefore Resnik measure is defined as:

$$sim_{res}(c_1, c_2) = IC\left(lcs\left(c_1, c_2\right)\right) \tag{4.3}$$

where IC is the information content of a concept and $lcs(c_1, c_2)$ is the lowest common subsumer of concepts c_1 and c_2.

This measure has a lower bound of 0 and no upper bound.

Table 4.9 illustrates Resnik matcher results with 10.0 threshold.

Table 4.9: Semantic relations produced by Resnik matcher

Source synset	Target synset	Semantic relation
robot	android	=
actor	actress	Idk
dog	cat	Idk

4.4.9 Jiang Conrath Matcher

Jiang Conrath matcher is a knowledge based matcher. It exploits Jiang Conrath semantic similarity measure. It returns $\equiv$ if the measure exceeds the given threshold and Idk otherwise. This measure [73] incorporates both

information content of the concepts and the information content of their lowest common subsumer. Originally Jiang Conrath defined the *distance* between two concepts as:

$$distance_{jc}(c_1, c_2) = IC(c_1) + IC(c_2) - 2 \cdot IC\left(lcs\left(c_1, c_2\right)\right) \tag{4.4}$$

where IC is the information content of a concept and lcs finds the lowest common subsumer of two given concepts.

Therefore the similarity of two concepts can be represented as

$$sim_{jc}(c_1, c_2) = \frac{1}{distance_{jc}(c_1, c_2)} \tag{4.5}$$

The formula has two special cases:

- In the first case all information content values are 0:

$$IC(c_1) = IC(c_2) = IC\left(lcs\left(c_1, c_2\right)\right) = 0 \tag{4.6}$$

  This happens when both concepts and their lowest common subsumer are either the *root node* or have a frequency count of 0. In both cases 0 similarity is returned.

- The second case is when

$$IC(c_1) + IC(c_2) = 2 \cdot IC\left(lcs\left(c_1, c_2\right)\right) \tag{4.7}$$

  which usually happens when

$$IC(c_1) = IC(c_2) = IC\left(lcs\left(c_1, c_2\right)\right) \tag{4.8}$$

  In this case c_1 and c_2 are the same concept and so we would like to return a maximum value of relatedness.

This measure has a lower bound of 0 and the upper bound $U_b = \frac{1}{-\ln((f_{root}-1)/f_{root})}$,

where f_{root} is the frequency of the taxonomy root.

The pseudo code for the algorithm is presented below.

Algorithm 5 The pseudo code of Jiang Conrath matcher

```
1. struct Synset
2.    String ID;
3.    String[] lemmas;
4.    String gloss;
5.    String[] relations;

6. float match( Synset synset1, Synset synset2 )
7.    String shortestPathId = getShortestPath( synset1, synset2 );
8.    Synset lcs = getLowCommonSubsumer( shortestPathId );
9.    float ic_lcs = getInformationContent( lcs );
10.   float ic_s1 = getInformationContent( synset1 );
11.   float ic_s2 = getInformationContent( synset2 );
12.   if ( ic_s1 == ic_s2 && ic_s1 == ic_lcs && ic_lcs == 0)
13.      return 0;
14.   if ( ic_s1 + ic_s2 == 2*ic_lcs )
15.      return maxValue;
16.   float distance = ic_s1 + ic_s2 - 2*ic_lcs;
17.   return 1/distance;
```

where **ID** (line 2) is the unique identifier, **lemmas** (line 3) is the list of synonyms that represent this synset, **gloss** (line 4) is the definition associated to that synset and **relations** (line 5) is a list of pointers to other synsets connected to this by a WordNet relation. **maxValue** (line 12) is the upper bound.

Firstly the shortest path between two synsets is computed (line 7). Then the lowest common subsumer of the input synsets is obtained (line 8). The information content values are computed for both synsets and lowest common subsumer in lines 6-8. Finally after handling the special cases (lines 12-15) the **distance** (line 16) and similarity (line 17) are computed.

Table 4.10 illustrates Jiang Conrath matcher results with 1.0 threshold.

Table 4.10: Semantic relations produced by Jiang Conrath matcher

Source synset	Target synset	Semantic relation
trip	hallucination	=
actor	actress	=
dog	cat	Idk

4.4.10 Lin Matcher

Lin matcher is a knowledge based matcher. It exploits Lin semantic similarity measure. It returns ≡ if the measure exceeds the given threshold and Idk otherwise. This measure is also based on information content[85]. It is defined as follows:

$$sim_{lin}(c_1, c_2) = \frac{2 \cdot IC\left(lcs\left(c_1, c_2\right)\right)}{IC(c_1) + IC(c_2)} \tag{4.9}$$

In the case of $IC(c_1) = 0$ and $IC(c_2) = 0$ 0 similarity is returned.

Table 4.11 illustrates Lin matcher results with 0.9 threshold.

Table 4.11: Semantic relations produced by Lin matcher

Source synset	Target synset	Semantic relation
robot	android	Idk
actor	actress	=
dog	cat	Idk

4.4.11 Hirst-St.Onge Matcher

Hirst-St.Onge matcher is a knowledge based matcher. It exploits Hirst-St.Onge semantic similarity measure. It returns ≡ if the measure exceeds the given threshold and Idk otherwise. This measure in contrast to information content based measures is not restricted to noun hierarchies.

Hirst and St.Onge [71] classified links in WordNet in 3 different categories:

- *Upward* (e.g. hypernym);

- *Downward* (e.g. holonym);
- *Horizontal* (e.g. antonym);

According to them two words are connected by a *strong* relation if:

- they both belong to the same synset;
- they belong to synsets connected by a *horizontal* link;
- one is a compound word, the second one is substring of the first while their synsets are connected by an *is a* relation;

For strongly related words relatedness is computed as $2 \cdot C$, where C is a constant used in the formula for medium-strong relations. Its value is 8, so the coefficient is equal to 16.

A *medium-strong* relation exists if the two synsets are connected by a *valid path* in WordNet. A path is considered valid if it is not longer than 5 links and conforms to one of the eight predefined patterns. The relatedness between two words connected by a medium-strong relation corresponds to the *weight* of the path which is given by the following formula:

$$Weight = C - Pathlength - k \cdot Changesindirection \tag{4.10}$$

where C and k are constants and, in our case they are assumed to be equal 8 and 1 respectively. The pseudocode below illustrates the algorithm. In order to compute the relatedness of two concepts first the type of relation holding between them is determined (line 2). Then the given relation is compared with existing patterns (line 4).

checkStrongRelationship (line 2) takes in input the two synsets and returns true if they fulfill one of the requirements of *Strong relations*. False otherwise.

getMedStrongWeight (line 4) takes in input two synsets and three zero values, that are respectively defined as **state**, **distance** and **chdir**, and

Algorithm 6 The pseudo code of Hirst-St.Onge matcher

```
1. float match( Synset synset1, Synset synset2 )
2.   if ( checkStrongRelationship( synset1, synset2 ))
3.     return 2*C;
4.   int weight = getMedStrongWeight( 0, 0, 0, synset1, synset2 );
5.   return weight;
```

returns the weight as stated in the above formula. Basically it searches recursively for a path from **synset1** to **synset2**, taking trace of the **distance**, changes in direction (**chdir**) so far and giving the **state** value for the next call. There are 8 possible states (from 0 to 7) in which the function may find itself, every state defines the rules the path has to follow. In particular, they specify the categories of links used in the last iteration, the ones that are allowed in the current step and the ones to use as next, taking into account the possible changes in direction.

This measure has a lower bound of 0 and an upper bound of 16.

Table 4.12 illustrates Hirst-St.Onge matcher results with 4.0 threshold.

Table 4.12: Semantic relations produced by Hirst-St.Onge matcher

Source synset	Target synset	Semantic relation
school	private school	=
actor	actress	=
dog	cat	Idk
sky	atmosphere	Idk

4.4.12 Context Vectors Matcher

Context Vectors Matcher is a knowledge based matcher. It exploits context vectors semantic similarity measure. It returns $\equiv$ if the measure exceeds the given threshold and Idk otherwise. This measure is based on context vector notion introduced by Schütze in [128]. Originally exploited for *word sense disambiguation* context vectors was adapted for semantic similarity

computation exploiting WordNet in [121].

The context vectors computation process starts from selection of the highly topical words which will define the dimensions of our *word space*. For our experiments we used WordNet glosses as a corpus. We stemmed all the words in the glosses and filtered out the function words. Then we counted the frequencies of the words in the corpus. Then we cut off the words with frequencies lower then 5 and higher then 1000. This allowed us to keep the most informative words. We also added a *tf-idf*[7] cutoff with an upper bound of 1500. This allowed to perform additional filtering of the frequent words. Further we will call the remaining words content words.

Afterwards we have created word vectors for all content words w as follows:

1. Initialize a vector $\vec{w}$ to zero
2. Find every occurrence of w in WordNet glosses
3. For each occurrence, search that gloss for words in the word space and increment the dimensions of $\vec{w}$ that correspond to those words.

The basic idea here is to have a matrix of word vectors, where every row corresponds to a word in the content words list and every column corresponds to the respective frequencies of each word in the word space.

The final step is to calculate gloss vectors for every synset in WordNet, this is done by adding the word vectors for each content word in the gloss. For example, if we want the gloss vector of *clock* we have to consider its gloss: *a timepiece that shows the time of day* and add the word vectors of *timepiece*, *shows*, *time* and *day*. Notice that this is a simplified example because for our experiments we use *extended glosses*, thus we had to take

[7] tf-idf is a weight used to evaluate how important a word is to a document (or gloss in our case). The formula we used is $tfidf = tf \cdot \ln(idf)$; where tf is the frequency of occurence of the word and $idf = \frac{nr.documents}{docFrequency}$. For our experiments *nr.documents* is the number of glosses and *docFrequency* is the number of glosses in which our word appears.

into account also the glosses of every concept connected to *clock* by a WordNet relation.

As soon as gloss vectors are calculated they are stored in database and the preprocessing phase is finished.

Semantic similarity of two synsets is defined as follows

$$sim_{cv}(c_1, c_2) = \cos(angle(\vec{v_1}, \vec{v_2})) \tag{4.11}$$

where c_1 and c_2 are the concepts, $\vec{v_1}$ and $\vec{v_2}$ are the respective gloss vectors and *angle* is the angle between vectors. This formula can be rewritten using vector products, it becomes:

$$sim_{cv}(c_1, c_2) = \frac{\vec{v_1} \cdot \vec{v_2}}{|v_1|\,|v_2|} \tag{4.12}$$

where at the denominator we have the *magnitude*[8] of the two vectors. It is a dot product[9] between two normalized vectors.

Figure 4.3 illustrates context vectors similarity in 2 dimensional space.

The pseudo code of context vector semantic similarity computation algorithm is as follows:

Algorithm 7 The pseudo code of context vector semantic similarity computation

```
1. float match( Synset synset1, Synset synset2 )
2.     int glossVec1[] = loadGlossVector( synset1 );
3.     int glossVec2[] = loadGlossVector( synset2 );
4.     float normVec1[] = normalizeVec( glossVec1 );
5.     float normVec1[] = normalizeVec( glossVec1 );
6.     return dotProduct( normVec1, normVec2 );
```

loadGlossVector (line 2-3) loads, from database, the precomputed gloss vector for the given synset.

normalizeVec (line 4-5) calculates the normalized form of the given vec-

[8] the magnitude of vector $\vec{v}$ is equal to $\sqrt{\sum_{i=1}^{n} v_i^2}$

[9] the dot product between vector $\vec{v}$ and vector $\vec{w}$ is $\vec{v} \cdot \vec{w} = \sum_{i=1}^{n} v_i w_i$

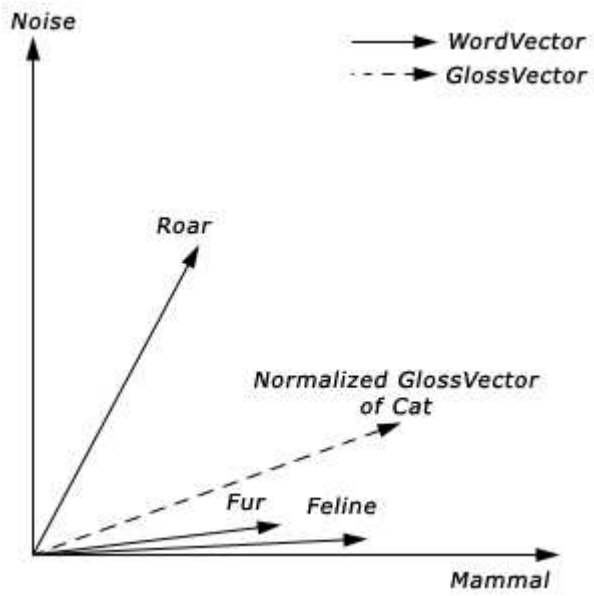

Figure 4.3: An example of a 2-dimensional space for Word and Gloss vectors for "cat - feline mammal usually having thick soft fur and being unable to roar" synset

tor.

dotProduct (line 6) return the dot product between two given vectors. The measure has a lower bound of 0 and an upper bound of 1.

Table 4.13 illustrates context vectors matcher results with 0.3 threshold.

Table 4.13: Semantic relations produced by context vectors matcher

Source synset	Target synset	Semantic relation
autograph	signature	=
actor	actress	=
robot	android	=
fruit	glass	Idk

4.4.13 WordNet gloss

WordNet gloss is a gloss based matcher. It 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, $\sqsubseteq$ 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 4.14 illustrates WordNet gloss matcher results.

Table 4.14: Semantic relations produced by WordNet gloss matcher

Source synset	Target synset	Semantic relation
hound	dog	$\sqsubseteq$
hound	ear	$\sqsubseteq$
dog	car	Idk

4.4.14 WordNet extended gloss

WordNet extended gloss is a gloss based matcher. It 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 [7]. Unlike [7], 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 $\sqsupseteq$ ($\sqsubseteq$) 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* descendant glosses.

Table 4.15 illustrates WordNet extended gloss matcher results.

Table 4.15: Semantic relations produced by WordNet extended gloss matcher

Source synset	Target synset	Semantic relation
dog	breed	$\sqsupseteq$
wheel	mashinery	$\sqsubseteq$
dog	cat	Idk

4.4.15 Gloss comparison

Gloss comparison is a gloss based matcher. Within the matcher 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 4.16 illustrates gloss comparison matcher results. Several modifications of this matcher exist.

Table 4.16: Semantic relations produced by gloss comparison matcher

Source synset	Target synset	Semantic relation
Afghan hound	Maltese dog	=
dog	cat	Idk

One can assign a higher weight to the phrases or particular parts of speech than single words [120]. In the current implementation we have exploited the approach used in [120], but changed the output to be a semantic relation.

4.4.16 Extended Gloss comparison

Extended gloss comparison is a gloss based matcher. It 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 are summarized in Table 4.17.

Table 4.17: Semantic relations produced by extended gloss comparison matcher

Source synset	Target synset	Semantic relation
house	animal	Idk
dog	cat	=

4.4.17 Semantic Gloss comparison

Semantic Gloss comparison is a gloss based matcher. 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 $\sqsupseteq$ ($\sqsubseteq$) 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 4.18 contains the results produced by semantic gloss comparison matcher.

Table 4.18: Semantic relations produced by extended gloss comparison matcher

Source synset	Target synset	Semantic relation
dog	breed	$\sqsupseteq$
dog	cat	Idk
wheel	machinery	$\sqsubseteq$

4.4.18 The label matching algorithm

The pseudo code implementing Step 3 is presented as Algorithm 8. The label matching algorithm produces (with the help of matchers of Table 4.1) a matrix of relations between all the pairs of atomic concepts of labels from both trees.

Algorithm 8 Label matching pseudo code

```
700. String[ ][ ] fillCLabMatrix(Tree of Nodes source,target);
710. String[ ][ ] cLabsMatrix;
720. String[ ] matchers;
730. int i, j;
740. matchers=getMatchers();
750. for each sourceAtomicConceptOfLabel ∈ source
760.    i=getACoLID(sourceAtomicConceptOfLabel);
770.    for each targetAtomicConceptOfLabel ∈ target
780.       j= getACoLID(targetAtomicConceptOfLabel);
790.       cLabsMatrix[i][j]=getRelation(matchers,
                              sourceAtomicConceptOfLa-
bel,targetAtomicConceptOfLabel);
795. return cLabsMatrix;

800. String getRelation(String[ ] matchers, AtomicConceptOfLabel source, target)
810. String matcher;
820. String relation="Idk";
830. int i=0;
840. while ((i<sizeof(matchers))&&(relation=="Idk"))
850.    matcher= matchers[i];
860.    relation=executeMatcher(matcher,source,target);
870.    i++;
880. return relation;
```

fillCLabMatrix takes as input two trees of nodes. It produces as output the matrix of semantic relations holding between the atomic concepts of labels in both trees. First, the element level matchers of Table 4.1, which are to be executed (based on the configuration settings), are acquired in line 740. Then, for each pair of atomic concepts of labels in both trees, semantic relations holding between them are determined by using the *getRelation*

function (line 790).

getRelation takes as input an array of matchers and two atomic concepts of labels. It returns the semantic relation holding between this pair of atomic concepts of labels according to the element level matchers. These label matchers are executed (line 860) until the semantic relation different from *Idk* is produced. Notice that execution order is defined by the matchers array.

The result of Step 3 is a matrix of the relations holding between atomic concepts of labels. A part of this matrix for the example in Figure 4.1 is shown in Table 4.19.

Table 4.19: *ClabsMatrix* matrix of semantic relations holding between concepts of labels.

B / A	*Classes*	*History*	*Modern*	*Europe*
Courses	$=$	*idk*	*idk*	*idk*
History	*idk*	$=$	*idk*	*idk*
Medieval	*idk*	*idk*	$\perp$	*idk*
Asia	*idk*	*idk*	*idk*	$\perp$

4.5 Step 4: Node Matching

During this step, we initially reformulate the tree matching problem into a set of node matching problems (one problem for each pair of nodes). Finally, we translate each node matching problem into a propositional validity problem. Let us first discuss in detail the tree matching algorithm. Then, we consider the node matching algorithm.

4.5.1 Tree matching algorithm

The tree matching algorithm is concerned with 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 Steps 1,2 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 9 illustrates the tree matching algorithm.

Algorithm 9 The pseudo code of the tree matching algorithm

```
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 ∈ source
970.      i=getNodeId(sourceNode);
980.      context_A=getCnodeFormula(sourceNode);
990.      for each targetNode ∈ 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;
```

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 4.19, which are considered when we match C4 in the tree A and C4 in the tree B are $Classes_B \equiv Courses_A$ and $History_B \equiv History_A$. In this case axioms is $(Classes_B \leftrightarrow Courses_A) \wedge (History_B \leftrightarrow History_A)$. 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*). A part of this matrix for the example in Figure 4.1 is shown in Table 4.20.

Table 4.20: *cNodesMatrix*: matrix of relations among the concepts at nodes (matching result).

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

4.5.2 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 4.21 (second column).

Table 4.21: The relationship between semantic relations and propositional formulas.

$rel(a, b)$	Translation of $rel(a, b)$ into propositional logic	Translation of formula (4.13) into Conjunctive Normal Form
$a = b$	$a \leftrightarrow b$	N/A
$a \sqsubseteq b$	$a \rightarrow b$	$axioms \wedge context_A \wedge \neg context_B$
$a \sqsupseteq b$	$b \rightarrow a$	$axioms \wedge context_B \wedge \neg context_A$
$a \bot b$	$\neg(a \wedge b)$	$axioms \wedge context_A \wedge context_B$

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:

$$axioms \longrightarrow rel(context_A, context_B) \tag{4.13}$$

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. 4.13 by proving that its negation, i.e., Eq. 4.14, is unsatisfiable.

$$axioms \wedge \neg rel(context_A, context_B) \tag{4.14}$$

Table 4.21 (third column) describes how Eq. 4.14 is translated before testing each semantic relation. Notice that Eq. 4.14 is in Conjunctive Normal Form (CNF), namely it is a conjunction of disjunctions of atomic formulas. The check for equivalence is omitted in Table 4.21, since $A = B$ 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.

We assume the labels of nodes and the knowledge derived from element level semantic matchers to be all globally consistent. Under this assumption the only reason why we get an unsatisfiable formula is because we have found a match between two nodes. In fact, axioms cannot be inconsistent by construction. Consistency of $context_A$ and $context_B$ is checked in the preprocessing phase (see, Section 4.3 for details). However, axioms and contexts (for example, $axioms \wedge context_A$) can be mutually inconsistent. The situation occurs, for example, when axioms entails negation of the variable occurring in the context. In this case, the concepts at nodes are disjoint. In order to guarantee the correct behavior of the algorithm we

perform the disjointness test first. It does not influence the algorithm correctness in general but allow us to obtain the correct result in this special case.

Let us consider the pseudo code of a basic node matching algorithm, see Algorithm 10. 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 [79].

Algorithm 10 The pseudo code of the node matching algorithm

```
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 "⊥";
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 "⊑";
1260.    if (isMG)
1270.      return "⊒";
1280.    return "Idk";
```

From the example in Figure 4.1, 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.

4.6 Summary

In this chapter we have identified semantic matching as the new approach for performing generic matching. We discussed the key notions of the approach. Then, the main four macro steps of the semantic matching a algorithm has been presented and described with the help of examples and pseudo-code.

Chapter 5

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 chapter 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 4.1 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 4.1 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.

Material presented in this chapter has been developed in collaboration with Enrico Giuchiglia and published in [63, 64].

In this chapter we discuss the optimizations for the matching problems involving conjunctive (§5.1) and disjunctive (§5.2) concepts at nodes.

5.1 Conjunctive concepts at nodes

Let us make some observations with respect to Table 4.21 (Section 4.5.2). 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. 4.14 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, formula Eq. 4.14 contains $n_A + n_B$ clauses with one variable.

5.1.1 The node matching problem by an example

Let us suppose that we want to match C_{16} in the tree A and C_{17} in the tree B in Figure 4.1. The relevant semantic relations between atomic concepts of labels are presented in Table 4.19. Thus, axioms is as follows:

$$\begin{aligned}&(course_A \leftrightarrow class_B) \wedge (history_A \leftrightarrow history_B) \wedge \\ &\neg(medieval_A \wedge modern_B) \wedge \neg(asia_A \wedge europe_B)\end{aligned} \tag{5.1}$$

which, when translated in CNF, becomes:

$$\begin{aligned}&(\neg course_A \vee class_B) \wedge (course_A \vee \neg class_B) \wedge (\neg history_A \vee history_B) \wedge \\ &(history_A \vee \neg history_B) \wedge (\neg medieval_A \vee \neg modern_B) \wedge (\neg asia_A \vee \neg europe_B)\end{aligned} \tag{5.2}$$

As from Step 2, $context_A$ and $context_B$ are constructed by taking the conjunction of the concepts of labels in the path from the node under consideration to the root. Therefore, $context_A$ and $context_B$ are:

$$course_A \wedge history_A \wedge medieval_A \wedge asia_A \tag{5.3}$$

$$class_B \wedge history_B \wedge modern_B \wedge europe_B \tag{5.4}$$

while their negations are:

$$\neg course_A \vee \neg history_A \vee \neg medieval_A \vee \neg asia_A \tag{5.5}$$

$$\neg class_B \vee \neg history_B \vee \neg modern_B \vee \neg europe_B \tag{5.6}$$

So far we have concentrated on atomic concepts of labels. The propositional formulas remain structurally the same if we move to conjunctive concepts at labels. Let consider the following example:

Figure 5.1: Two simple classifications (obtained by modifying, pruning the example in Figure 4.1)

Suppose we want to match C_2 in the tree A and C_2 in the tree B in Figure 5.1. Axioms required for this matching task are as follows: $(course_A \leftrightarrow class_B) \wedge (history_A \leftrightarrow history_B) \wedge (medieval_A \perp modern_B) \wedge (asia_A \perp europe_B)$. If we compare them with those of Eq. 5.1 and Eq. 5.2, which represent axioms for the above considered example in Figure 4.1, we find out that they are the same. Furthermore, as from Step 2, the propositional formulas for $context_A$ and $context_B$ are the same for atomic and for conjunctive concepts of labels as long as they "globally" contain the same

formulas. In fact, concepts at nodes are constructed by taking the conjunction of concepts at labels. Splitting a concept of a label with two conjuncts into two atomic concepts has no effect on the resulting matching formula. The matching result for the matching tasks in Figure 5.1 is presented in Table 5.1.

Table 5.1: The matrix of relations between concepts at nodes (matching result) for Figure 5.1.

A \ B	C_1	C_2
C_1	$=$	$\sqsupseteq$
C_2	$\sqsubseteq$	$\bot$

5.1.2 Optimizations

Tests for less and more general relations.

Using the observations in the beginning of Section 5.1 concerning Table 4.21, Eq. 4.14, with respect to the tests for less/more general relations, can be represented as follows:

$$\overbrace{\bigwedge_{0}^{n*m}(\neg A_s \vee B_t) \wedge \bigwedge_{0}^{n*m}(A_k \vee \neg B_l) \wedge \bigwedge_{0}^{n*m}(\neg A_p \vee \neg B_r)}^{Axioms} \wedge \overbrace{\bigwedge_{i=1}^{n} A_i}^{Context_A} \wedge \overbrace{\bigvee_{j=1}^{m} \neg B_j}^{\neg Context_B} \tag{5.7}$$

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. 5.7 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. 5.7 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 [28]. Notice, that DPLL-based SAT solvers require quadratic time in this case [134].

In order to understand how the linear time algorithm works, let us prove the unsatisfiability of Eq. 5.7 in the case of matching C_{16} in the tree A and C_{17} in the tree B in Figure 4.1. In this case, Eq. 5.7 is as follows:

$$(\neg \boldsymbol{course}_A \vee class_B) \wedge (\boldsymbol{course}_A \vee \neg class_B) \wedge (\neg \boldsymbol{history}_A \vee history_B) \wedge$$
$$(\boldsymbol{history}_A \vee \neg history_B) \wedge (\neg \boldsymbol{medieval}_A \vee modern_B) \wedge (\neg \boldsymbol{asia}_A \vee \neg europe_B) \wedge$$
$$\boldsymbol{course}_A \wedge \boldsymbol{history}_A \wedge \boldsymbol{medieval}_A \wedge \boldsymbol{asia}_A \wedge$$
$$(\neg class_B \vee \neg history_B \vee \neg modern_B \vee \neg europe_B) \tag{5.8}$$

In Eq. 5.8, the variables from $context_A$ are written in bold face. First, we assign true to all unit clauses occurring in Eq. 5.8 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: $course_A \vee \neg class_B$, $history_A \vee \neg history_B$). The resulting formula is as follows:

$$\begin{aligned} &class_B \wedge history_B \wedge \neg modern_B \wedge \neg europe_B \wedge \\ &(\neg class_B \vee \neg history_B \vee \neg modern_B \vee \neg europe_B) \end{aligned} \tag{5.9}$$

Eq. 5.9 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. 5.8 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. 5.9. 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 the form $\neg A_i \vee B_j$ occur in Eq. 5.7 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 algorithm for testing satisfiability:

- Step 1. Create an array of size m. Each entry in the array stands for one B_j in Eq. 5.7.
- 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.

Disjointness test.

Using the same notation as before in this section, Eq. 4.14 with respect to the disjointness test can be represented as follows:

$$\overbrace{\bigwedge_{q=0}^{n*m}(\neg A_s \vee B_t) \wedge \bigwedge_{w=0}^{n*m}(A_k \vee \neg B_l) \wedge \bigwedge_{v=0}^{n*m}(\neg A_p \vee \neg B_r)}^{Axioms} \wedge \overbrace{\bigwedge_{i=1}^{n} A_i}^{Context_A} \wedge \overbrace{\bigwedge_{j=1}^{m} B_j}^{Context_B} \tag{5.10}$$

For example, the formula for testing disjointness between C_{16} in the tree A and C_{17} in the tree B in Figure 4.1 is as follows:

$$(\neg \boldsymbol{course}_A \vee class_B) \wedge (\boldsymbol{course}_A \vee \neg class_B) \wedge (\neg \boldsymbol{history}_A \vee history_B) \wedge$$
$$(\boldsymbol{history}_A \vee \neg history_B) \wedge (\neg \boldsymbol{medieval}_A \vee modern_B) \wedge (\neg \boldsymbol{asia}_A \vee \neg europe_B) \wedge$$
$$\boldsymbol{course}_A \wedge \boldsymbol{history}_A \wedge \boldsymbol{medieval}_A \wedge \boldsymbol{asia}_A \wedge$$
$$class_B \wedge history_B \wedge modern_B \wedge europe_B \tag{5.11}$$

Eq. 5.11 is Horn, and thus, similarly to Eq. 5.8, the satisfiability of this formula can be decided by the unit propagation rule. After assigning true to all the variables in $context_A$ and propagating the results we obtain the following formula:

$$\begin{aligned} &class_B \wedge history_B \wedge \neg modern_B \wedge \neg europe_B \wedge \\ &class_B \wedge history_B \wedge modern_B \wedge europe_B \end{aligned} \tag{5.12}$$

If we further unit propagate $class_B$ and $history_B$ (this means that we assign them to true), then we obtain the contradiction $modern_B \wedge \neg modern_B \wedge europe_B \wedge \neg europe_B$. Therefore, the formula is unsatisfiable. This contradiction arises because $(\neg medieval_A \vee \neg modern_B)$ and $(\neg asia_A \vee \neg europe_B)$ occur in Eq. 5.11, which, in turn, are derived (as from Table 4.21) from the disjointness axioms $modern_B \perp medieval_A$ and $asia_A \perp europe_B$. In fact, all the clauses in Eq. 5.10 contain one positive literal except for the clauses in axioms corresponding to disjointness relations. Thus, the key intuition here is that if there are no disjointness axioms, then Eq. 5.10 is satisfiable. However, if there is a disjointness axiom, atoms occurring there are also ensured to be either in $context_A$ or in $context_B$, hence, Eq. 5.10 is unsatisfiable. Therefore, the optimization consists of just checking the presence/absence of disjointness axioms in axioms.

5.2 Disjunctive concepts at nodes

5.2.1 The node matching problem by an example

Now, we allow for the concepts at nodes to contain conjunctions and disjunctions in any order. Suppose, we want to match C_5 in the tree A and C_5 in the tree B in Figure 4.1. The relevant part of *cLabsMatrix* is shown

in Table 5.2.

Table 5.2: cLabsMatrix: matrix of relations among the atomic concepts of labels.

B / A	*Classes*	*Mechanics*	*Optics*	*Statistics*	*Dynamics*	*Kinematics*
Courses	=	idk	idk	idk	idk	idk
Biology	idk	idk	idk	idk	idk	idk
Zoology	idk	idk	idk	idk	idk	idk
Botany	idk	idk	idk	idk	idk	idk
Neurobiology	idk	idk	idk	idk	idk	idk
Genetics	idk	idk	idk	idk	idk	idk
Physiology	idk	idk	idk	idk	idk	idk

As from Table 4.21, the axioms is as follows:

$$(course_A \leftrightarrow class_B) \tag{5.13}$$

Eq. 5.13 in CNF then becomes:

$$(\neg course_A \vee class_B) \wedge (course_A \vee \neg class_B) \tag{5.14}$$

As from Step 2, $context_A$ and $context_B$ are:

$$\begin{aligned} &class_B \wedge (mechanics_B \vee optics_B \vee thermodynamics_B) \wedge \\ &(statics_B \vee dynamics_B \vee kinematics_B) \end{aligned} \tag{5.15}$$

$$\begin{aligned} &course_A \wedge (biology_A \vee zoology_A \vee botany_A) \wedge \\ &(neurobiology_A \vee genetics_A \vee physiology_A) \end{aligned} \tag{5.16}$$

The negations of $context_A$ and $context_B$, in turn, are:

$$\neg class_B \vee (\neg mechanics_B \wedge \neg optics_B \wedge \neg thermodynamics_B) \vee (\neg statics_B \wedge \neg dynamics_B \wedge \neg kinematics_B) \quad (5.17)$$

$$\neg course_A \vee (\neg biology_A \wedge \neg zoology_A \wedge \neg botany_A) \vee (\neg neurobiology_A \wedge \neg genetics_A \wedge \neg physiology_A) \quad (5.18)$$

The matching result for this task is presented in Table 5.3.

Table 5.3: *cNodesMatrix*: matrix of relations among the concepts at nodes (matching result).

A \ B	C_1	C_2	C_5
C_1	$=$	idk	idk
C_2	idk	idk	idk
C_5	idk	idk	idk

5.2.2 Optimizations

As from Table 4.21, 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 tests for less/more general relations). Thus, for instance, as from Table 4.21 in case of test for less general relation we obtain the following formula:

$$(\neg course_A \vee class_B) \wedge (course_A \vee class_B) \wedge (mechanics_B \vee optics_B \vee thermodynamics_B) \wedge (statics_B \vee dynamics_B \vee kinematics_B) \wedge ((\neg biology_A \wedge \neg zoology_A \wedge \neg botany_A) \vee (\neg neurobiology_A \wedge \neg genetics_A \wedge \neg physiology_A)) \tag{5.19}$$

With disjunctive concepts at nodes, Eq. 4.14 is a full propositional formula and no hypothesis can be made on its structure. As a consequence, its satisfiability must be tested using a standard DPLL SAT solver. Thus, for instance, CNF conversion of Eq. 5.19 is as follows:

$$\begin{aligned}
&(\neg course_A \vee class_B) \wedge (course_A \vee class_B) \wedge \\
&(mechanics_B \vee optics_B \vee thermodynamics_B) \wedge \\
&(statics_B \vee dynamics_B \vee kinematics_B) \wedge \\
&((\neg course_A \vee \neg biology_A \vee \neg neurobiology_A) \wedge \\
&(\neg course_A \vee \neg biology_A \vee \neg genetics_A) \wedge \\
&(\neg course_A \vee \neg biology_A \vee \neg physiology_A) \wedge \\
&(\neg course_A \vee \neg zoology_A \vee \neg neurobiology_A) \wedge \\
&(\neg course_A \vee \neg zoology_A \vee \neg genetics_A) \wedge \\
&(\neg course_A \vee \neg zoology_A \vee \neg physiology_A) \wedge \\
&(\neg course_A \vee \neg botany_A \vee \neg neurobiology_A) \wedge \\
&(\neg course_A \vee \neg botany_A \vee \neg genetics_A) \wedge \\
&(\neg course_A \vee \neg botany_A \vee \neg physiology_A))
\end{aligned} \tag{5.20}$$

In order to avoid the space explosion, which may arise when converting a formula into CNF (see for instance Eq. 5.20), we apply a set of structure preserving transformations [122, 53]. The main idea is to replace disjunc-

tions occurring in the original formula with newly introduced variables and explicitly state that these variables imply the subformulas they substitute. Consider for instance Eq. 5.19. We obtain:

$$\begin{aligned}
&(\neg course_A \vee class_B) \wedge (course_A \vee class_B) \wedge \\
&(mechanics_B \vee optics_B \vee thermodynamics_B) \wedge \\
&(statics_B \vee dynamics_B \vee kinematics_B) \wedge \\
&new_1 \wedge new_2 \wedge (new_1 \rightarrow \neg biology_A \vee \neg zoology_A \vee \neg car_A) \wedge \\
&(new_2 \rightarrow \neg neurobiology_A \vee \neg genetics_A \vee \neg physiology_A)
\end{aligned} \tag{5.21}$$

where new_1 and new_2 stand for newly introduced variables. Eq. 5.16 is converted into CNF as follows:

$$\begin{aligned}
&(\neg course_A \vee class_B) \wedge (course_A \vee class_B) \wedge \\
&(mechanics_B \vee optics_B \vee thermodynamics_B) \wedge \\
&(statics_B \vee dynamics_B \vee kinematics_B) \wedge \\
&new_1 \wedge new_2 \wedge (\neg new_1 \vee \neg biology_A \vee \neg zoology_A \vee \neg car_A) \wedge \\
&(\neg new_2 \vee \neg neurobiology_A \vee \neg genetics_A \vee \neg physiology_A)
\end{aligned} \tag{5.22}$$

Notice that the size of the propositional formula in CNF grows linearly with respect to number of disjunctions in original formula. To account for this optimization in *nodeMatch* all calls to *convertToCNF* are replaced with calls to *optimizedConvertToCNF*, (see Algorithm 11):

Algorithm 11 The CNF conversion optimization pseudo code

```
1120. formulaInCNF=optimizedConvertToCNF(formula);
...
1170. formulaInCNF=optimizedConvertToCNF(formula);
...
1200. formulaInCNF=optimizedConvertToCNF(formula);
```

5.3 Summary

In this chapter we have presented a set of optimizations to the basic semantic matching algorithm and described them with the help of examples and pseudo-code. The key idea is to apply ad hoc reasoning procedures when propositional formulas arising in the matching process belong to the fragments (of propositional logic) for which the more efficient reasoning algorithms are known.

Chapter 6

Beyond semantic matching: structure preserving semantic matching

This chapter presents the structure preserving semantic matching algorithms. In particular the focus is on exact and approximate structure preserving matching algorithms. Material presented in this chapter has been developed in collaboration with Fiona McNeil.

In this chapter we first discuss a motivating scenario for structure preserving semantic matching (§6.1). Section 6.2 is devoted to the exact structure matching algorithm. In Section 6.3 we define the abstraction operations and introduce the correspondence between them and tree edit operations. In Section 6.4 we show how existing tree edit distance algorithms can be exploited for the computation of the global similarity between two web service descriptions. Section 6.5 is devoted to approximate structure matching algorithm.

6.1 A Motivating Example

Figure 6.1 provides an example of exactly matched web service descriptions along with their tree representations (or term trees). Dashed lines stand for the correspondences holding among the nodes of the term trees.

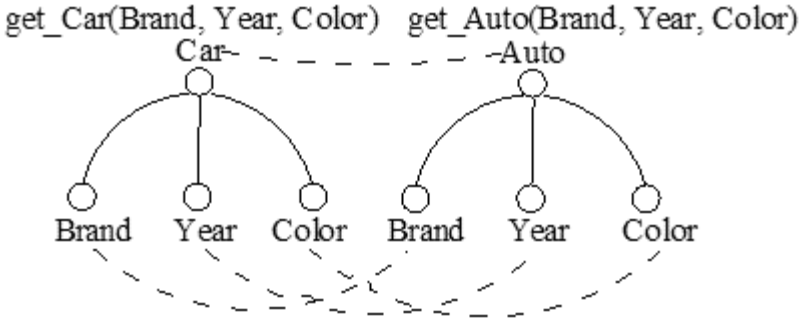

Figure 6.1: Exactly matched web service descriptions and their tree representations.

In particular, in Figure 6.1 we have an exact match, namely the first of the services requires the second to return *Cars* of a given *Brand*, *Year* and *Color* while the other provides *Autos* of a given *Brand*, *Year* and *Color*. Notice that, there are no structural differences and that the only difference is in the function names.

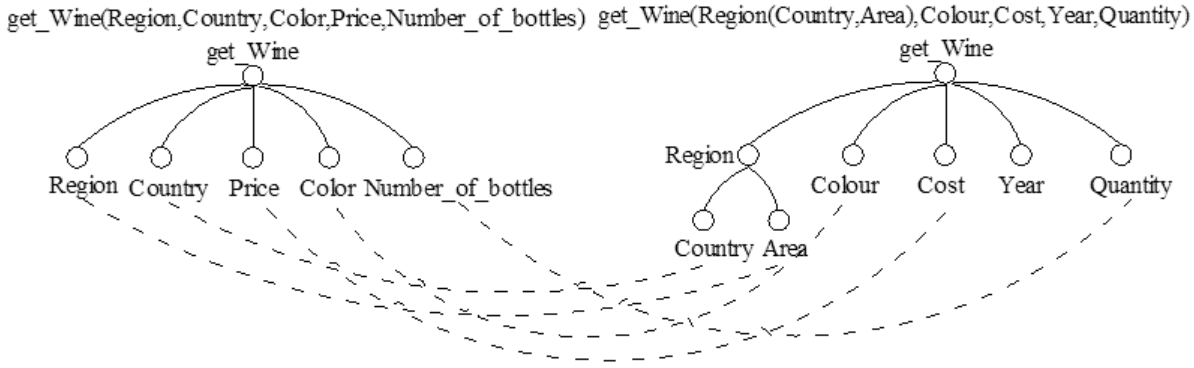

Figure 6.2: Approximately matched web service descriptions and their tree representations.

Consider now Figure 6.2. It provides an example of an approximate match. In this case a more sophisticated data translation is required. For

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

Therefore, in order to guarantee the successful data translation we are interested in the correspondences holding among the nodes of the term trees of the given web service descriptions only in the case when the web service descriptions themselves are "similar enough". At the same time the correspondences have to preserve the certain structural properties of the descriptions being matched. In particular we require the functions to be mapped to functions and variables to variables.

6.2 Exact structure semantic matching

There are two stages in the matching process:

- *Node matching*: solves the semantic heterogeneity problem by considering only labels at nodes and domain specific contextual information of the trees. In our approach we use semantic matching as extensively described in §4. Notice that the result of this stage is the set of correspondences holding between the nodes of the trees.

- *Structural tree matching*: exploits the results of the node matching and the structure of the tree to find the correspondences holding between the trees themselves (e.g., $tree_1$ is 0.7 similar to $tree_2$).

Let us consider the latter.

The exact structure matching algorithm exploits the results of the node matching algorithm. It is designed to succeed for equivalent terms and to fail otherwise. It expects the trees to have the same depth and the same number of children. More precisely we say that two trees T_1 and T_2 match iff for any node n_{11} in T_1 there is a node n_{21} in T_2 such that

- n_{11} semantically matches n_{21}, which in this case holds iff $c@n_{21}$ is equivalent to $c@n_{22}$ given the available background knowledge, where $c@n_1$ and $c@n_2$ are the concepts at nodes of n_1 and n_2;
- n_{11} and n_{21} reside on the same depth in T_1 and T_2, respectively;
- all ancestors of n_{11} are semantically matched to the ancestors of n_{21};

The pseudo code in Figure 12 illustrates an algorithm for exact structure matching.

exactStructureMatch takes two trees of nodes *source* and *target* as an input. **exactStructureMatch** returns an array of *MappingElements* holding between the nodes of the trees if there is an exact match between them and null otherwise. The array of *MappingElements result* is created (line 12) and filled by **exactTreeMatch** (line 13). **allNodesMapped** checks whether all the nodes of *source* tree are mapped to the nodes of the *target* tree (line 14). If this is the case there is an exact structure match between the trees and the set of computed mappings is returned (line 15). **exactTreeMatch** takes two trees of nodes *source* and *target* and array of MappingElements *result* as an input. It recursively fills *result* with the mappings computed by **nodeMatch** (line 23). **exactTreeMatch** starts from obtaining the roots of *source* and *target* trees (lines 19-20). The semantic relation holding between them is computed by **nodeMatch** (line 21) implementing the node matching algorithm. If the relation is equivalence, the corresponding mapping is saved to *result* array (lines 22-23) and the children of the root nodes are obtained (line 26-27). Finally the loops

Algorithm 12 Pseudo code for exact structure matching algorithm

```
1.Node struct of
2.    int nodeId;
3.    String label;
4.    String cLabel;
5.    String cNode;

6.MappingElement struct of
7.    int MappingElementId;
8.    Node source;
9.    Node target;
10.   String relation;

11.MappingElement[] exactStructureMatch (Tree of Nodes source,target)
12.    MappingElement[] result;
13.    exactTreeMatch(source,target,result);
14.    if (allNodesMapped(source,target,result))
15.       return result;
16.    else
17.       return null;

18.void exactTreeMatch(Tree of Nodes source,target,MappingElement[] result)
19.    Node sourceRoot=getRoot(source);
20.    Node targetRoot=getRoot(target);
21.    String relation= nodeMatch(sourceRoot,targetRoot);
22.    if (relation=="=")
23.       addMapping(result,sourceRoot,targetRoot,"=");
24.    else
25.       return;
26.    Node[] sourceChildren=getChildren(sourceRoot);
27.    Node[] targetChildren=getChildren(targetRoot);
28.    For each sourceChild in sourceChildren
29.       Tree of Nodes sourceChildSubTree=getSubTree(sourceChild);
30.       For each targetNode in target
31.          Tree of Nodes targetChildSubTree=getSubTree(targetChild);
32.           exactTreeMatch(sourceChildSubTree, targetChildSubTree, nodesToMatch);
```

on *sourceChildren* and *targetChildren* (lines 28-32) allow to call **exactTreeMatch** recursively for all pairs of sub trees rooted at *sourceChildren* and *targetChildren* elements.

6.3 Approximate matching via abstraction/refinement operations

In [61], Giunchiglia and Walsh categorize the various kinds of abstraction operations in a wide-ranging survey. They also introduce a new class of abstractions, called TI-abstractions (where TI means "Theorem Increasing"), which have the fundamental property of maintaining completeness, while losing correctness. In other words any fact which is true of the original term is also true of the abstract term, but not viceversa. And similarly, if a ground formula is true so is the abstract formula, but not vice versa. Dually, by taking the inverse of each abstraction operation, we can define a corresponding refinement operation which preserves correctness while loosing completeness. The second fundamental property of the abstraction operations is that they provide all and only the possible ways in which two first order terms can be made to differ by manipulations of their signature, still preserving completeness. In other words, this set of abstraction/refinement operations defines all and only the possible ways in which correctness and completeness are maintained when operating on first order terms and atomic formulas. This is the fundamental property which allows us to study and consequently quantify the semantic similarity (distance) between two first order terms. To this extent it is sufficient to determine which abstraction/refinement operations are necessary to convert one term into the other and to assign to each of them a cost that models the "semantic distance" associated to the operation.

Giunchiglia and Walsh's categories are as follows:

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

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

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

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

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

Acura + Nissan $\mapsto$ *Nissan.*

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

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

Bottle(A) $\mapsto$ *Bottle.*

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

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

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

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

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

$$(Bottle\ A) \mapsto^{\sqsubseteq_P} (Bottle) \mapsto^{\sqsubseteq_{Pd}} (Container) \tag{6.1}$$

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

Therefore, the problem of AME computation becomes a problem of minimal cost composition of the abstraction/refinement operations allowed for the given relation R that are necessary to convert one term into the other. In order to solve this problem we propose to represent abstraction/refinement operations as tree edit distance operations applied to the term trees. This allows to redefine the problem of AME computation into a tree edit distance problem.

In its traditional formulation, the tree edit distance problem considers three operations: (i) vertex deletion, (ii) vertex insertion, and (iii) vertex

replacement [132]. Often these operations are presented as rewriting rules:

$$(\text{i})\upsilon \rightarrow \lambda;\ (\text{ii})\lambda \rightarrow \upsilon;\ (\text{iii})\upsilon \rightarrow \omega; \tag{6.2}$$

where υ and ω correspond to the labels of nodes in the trees while λ stands for the special blank symbol. Figure 6.3 illustrates two applications of delete and replace tree edit operations.

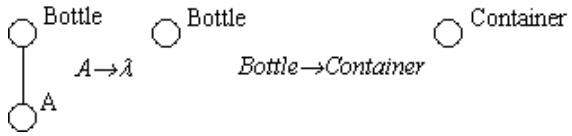

Figure 6.3: Delete and replace tree edit operations

Our proposal is to restrict the formulation of the tree edit distance problem in order to reflect the semantics of the first order terms. In particular we propose to redefine the tree edit distance operations in such a way that will allow them to have one-to-one correspondence to the abstraction/refinement operations presented previously in this section. Table 6.1 illustrates the correspondence between abstraction/refinement and tree edit operations. The first column presents the abstraction/refinement operations. The second column lists corresponding tree edit operations. The third column describes the preconditions of the tree edit operation use.

Table 6.1: The correspondence between abstraction/refinement operations and tree edit operations.

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

Consider, for example, the first line of Table 6.1. The predicate abstraction operation applied to first order term t_1 results with term t_2 ($t_1 \sqsupseteq_{Pd} t_2$). This abstraction operation corresponds to tree edit replacement operation applied to term tree of t_1 that replaces the node a with the node b ($a \rightarrow b$). Moreover the operation can be applied only in the case if (i) label a is a generalization of label b and (ii) both the nodes a and b in the term trees correspond to predicates in the first order terms.

6.4 Computing the global similarity between two trees

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

$$Cost = \sum_{i \in S} n_i * Cost_i \tag{6.3}$$

The solution of the problem then becomes to minimize *Cost* in Eq. 6.3 and, therefore, to determine the minimal set of operations (i.e., the one with the minimum cost) which transforms one tree into another. In Eq. 6.3 S stands for the set of the allowed tree edit operations; n_i stands for the number of i-th operations necessary to convert one tree into the other and $Cost_i$ defines the cost of the i-th operation. Our goal is to define the $Cost_i$ in a way to model the semantic distance.

A possible uniform proposal is to assign the same unit cost to all tree edit operations that, as from Table 6.1, have their abstraction theoretic counterparts. Table 6.2 illustrates the costs of the abstraction/refinement (tree edit) operations, depending on the relation (equivalence, abstraction

or refinement) being computed. Notice that the costs for estimating abstraction ($\sqsubseteq$) and refinement ($\sqsupseteq$) relations in AME have to be adjusted according to their definitions. In particular the tree edit operations corresponding to abstraction/refinement operations that are not allowed by the definition of the given relation have to be prohibited by assigning to them an infinite cost. Notice also that, we do not give any preference to a particular type of abstraction/refinement operations. Of course this strategy can be changed to satisfy certain domain specific requirements.

Table 6.2: Costs of the abstraction/refinement (tree edit) operations, exploited for computation of equivalence ($Cost_{\equiv}$), abstraction ($Cost_{\sqsubseteq}$) and refinement ($Cost_{\sqsupseteq}$) relations holding among the terms.

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

Consider, for example, the first line in Table 6.2. The cost of the tree edit distance operation that correspond to the propositional abstraction ($t_1 \sqsupseteq_{Pd} t_2$) is equal to 1 when used for the computation of equivalence ($Cost_{\equiv}$) and abstraction ($Cost_{\sqsupseteq}$) relations in AME. It is equal to ∞ when used for the computation of refinement ($Cost_{\sqsubseteq}$) relation.

Eq. 6.3 can now be used for computation of the tree edit distance score. However, when comparing two web service descriptions we are interested rather in similarity than in distance. We exploit the following equation to convert the distance produced by an edit distance algorithm into the similarity score:

$$sim = 1 - \frac{Cost}{max(number\_of\_nodes_1, number\_of\_nodes_2)} \tag{6.4}$$

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

Many existing tree edit distance algorithms allow to keep track of the nodes to which a replace operation is applied. Therefore, as a result they allow to obtain not only the minimal tree edit cost but also a minimal cost mapping among the nodes of the trees. According to [132] this minimal cost mapping is (i) one-to-one; (ii) horizontal order preserving between sibling nodes; and (iii) vertical order preserving. For example, the mapping depicted in Figure 6.1 complies to all these requirements while the mapping depicted in Figure 6.2 violates (ii). In particular the third sibling *Price* on the left tree is mapped to the third sibling *Cost* on the right tree while the fourth sibling *Color* on the right tree is mapped to the second sibling *Colour* on the left tree.

For the tree edit distance operations depicted in Table 6.1 we propose to keep track of nodes to which the tree edit operations derived from the replace operation are applied. In particular we consider the operations that correspond to predicate and domain abstraction/refinement ($t_1 \sqsupseteq_{Pd}$, $t_1 \sqsubseteq_{Pd}$, $t_1 \sqsupseteq_D$, $t_1 \sqsubseteq_D$). This allows us to obtain a mapping among the nodes of the term trees with the desired properties (i.e., there is only one-to-one correspondences in the mapping). Moreover it complies to the structure preserving matching requirements namely functions are mapped to functions and variables are mapped to variables. This is the case because (i) predicate and domain abstraction/refinement operations do not convert, for example, a function into a variable and (ii) the tree edit distance operations, as from Table 6.1, have a one-to-one correspondence with abstraction/refinement operations.

At the same time a mapping returned by a tree edit distance algorithm preserves the horizontal order among the sibling nodes, but this is

not desirable property for the data translation purposes. This is the case because the correspondences that do not comply to the horizontal order preservation requirements, like the one holding between *Colour* and *Color* on Figure 6.2, are not included in the mapping. However, as from Table 6.1, the tree edit operations corresponding to predicate and domain abstraction/refinement ($t_1 \sqsupseteq_{Pd}$, $t_1 \sqsubseteq_{Pd}$, $t_1 \sqsupseteq_D$, $t_1 \sqsubseteq_D$) can be applied only to those nodes of the trees whose labels are either generalizations or specializations of each other, as computed by the node matching algorithm. Therefore, given the mapping produced by the node matching algorithm we can always recognize the cases when the horizontal order between sibling nodes is not preserved and change the ordering of the sibling nodes to make the mapping horizontal order preserving. For example, swapping the nodes *Cost* and *Colour* in the right tree depicted on Figure 6.2 does not change the meaning of the corresponding term while allows the correspondence holding between *Colour* and *Color* on Figure 6.2 to be included in the mapping produced by a tree edit distance algorthm.

6.5 The approximate structure matching algorithm

As from above our goal is to find 'good enough' services [65] if perfect are not available. We start by providing a definition of the approximate structure matching as the basis for the algorithm.

We say that two nodes n_1 and n_2 in the trees T_1 and T_2 approximately match iff $c@n_1$ R $c@n_2$ holds given the available background knowledge, where $c@n_1$ and $c@n_2$ are the concepts at nodes of n_1 and n_2, and where $R \in \{\equiv, \sqsubseteq, \sqsupseteq, \wedge, \bot, not\ related\}$.

We say that two trees T_1 and T_2 match iff there is at least one node n_{11} in T_1 and a node n_{21} in T_2 such that

- n_{11} approximately matches n_{21};

- all ancestors of n_{11} are approximately matched to the ancestors of n_{21};

The approximate structure matching algorithm exploits the node matching algorithm presented in Section 6.2. First the approximate structure matching algorithm estimates the similarity of two terms by application of a tree edit distance algorithm with the tree edit operations and costs modified as described in Sections 6.3 and 6.4. The similarity scores are computed for equivalence, abstraction and refinement relations. For each of these cases the tree edit distance operation costs are modified as depicted on Table 6.2. The relation with the highest similarity score are assumed to hold among the terms. If the similarity score exceeds a given threshold the mappings connecting the nodes of the term trees, as computed by the tree edit distance algorithm, are returned by the matching routine what allows for further data translation.

Pseudo code below illustrates approximate structure matching algorithm.

Algorithm 13 Pseudo code for approximate structure matching algorithm

```
AME struct of
   Tree of Nodes source;
   Tree of Nodes target;
   String relation;
   double approximationScore;
1.MappingElement[] approximateStructureMatch(Tree of Nodes source, target,
double threshold)
2.    MappingElement[] result;
3.    approximateTreeMatch(source,target,result);
4.    AME ame=analyzeMismatches(source,target,result);
5.          if (getRelation(ame)==”=”) or (getRelation(ame)==”¡”)or
(getRelation(ame)==”¿”)
6.       if (getApproximationScore(ame)¿threshold)
7.          return result;
8.    return null;
```

approximateStructureMatch takes as input the *source* and *target* term trees and a *threshold* value. **approximateTreeMatch** fills the *result* array

(line 3) which stores the mappings holding between the nodes of the trees. In contrast to **exactTreeMatch** in Figure 12 **approximateTreeMatch** considers the semantic relations other than equivalence. An AME *ame* is computed (line 4) by **analyzeMismatches**. If *ame* stands for equivalence, abstraction or refinement relations (line 5) and if an *approximationScore* exceeds *threshold* (line 6) the mappings calculated by **approximateTreeMatch** are returned (line 7). **analyzeMismatches** calculates the aggregate score of tree match quality by exploiting a tree edit distance algorithm as described in Section 6.4.

6.6 Summary

In this section we have presented exact and approximate structure preserving semantic matching algorithms based on the formal theory of abstraction and utilizing state of the art tree edit distance algorithms for efficient semantic distance computation.

Part IV

Testbeds and evaluation

Chapter 7

Testbed generation

This chapter presents a method for large scale datasets construction good for evaluation of both Recall and Precision.

Material presented in this chapter has been developed in collaboration with Paolo Avesani and published in [6].

In this chapter we first present the matching evaluation problem and introduce commonly used measures for matching evaluation (§7.1). Then we discuss the ways of constructing large scale datasets good for Recall (§7.2) and Precision (§7.3) evaluation. Finally we present the results of datasets evaluation (§7.4).

7.1 Matching evaluation problem

The commonly accepted measures for a quantitative mapping evaluation are based on the well known in information retrieval measures of relevance, namely *Precision* and *Recall*.

Consider Figure 7.1; the calculation of these measures is based on the

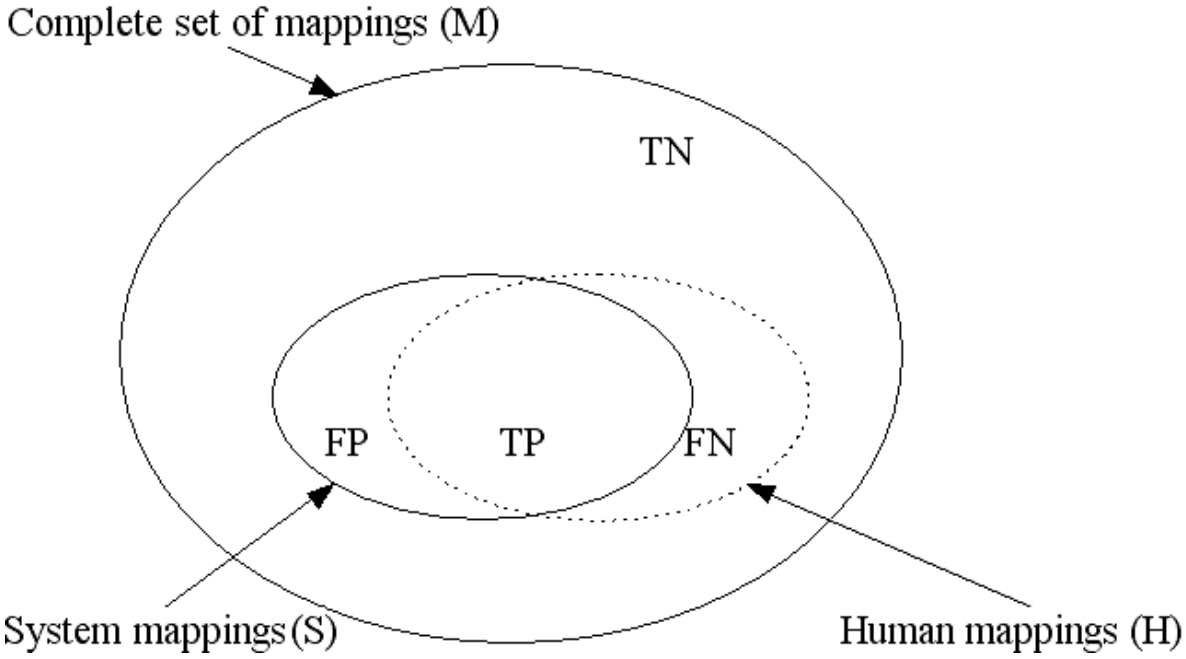

Figure 7.1: Basic sets of mappings

comparison between the mappings produced by a matching system (S in Figure 7.1) and a complete set of reference mappings H considered to be correct (the area inside the dotted circle in Figure 7.1). H is usually produced by humans. Here and further we refer to the set of all possible mappings (i.e., cross product of two input graphs) as M. Finally, the correct mappings found by the system are the *true positives*:

$$TP = S \cap H \tag{7.1}$$

The incorrect mappings found by the system are the *false positives*:

$$FP = S - S \cap H \tag{7.2}$$

The correct mappings missed by the system are *false negatives*:

$$FN = H - S \cap H \tag{7.3}$$

The incorrect mappings not returned by the system are the *true negatives*:

$$TN = M - S \cap H \tag{7.4}$$

Further we call the mappings in H *positive mappings*, and the mappings in

$$N = M - H = TN + FP \tag{7.5}$$

negative mappings.

Precision is a correctness measure which varies from [0,1]. It is calculated as

$$Precision = \frac{|TP|}{|TP + FP|} = \frac{H \cap S}{S} \tag{7.6}$$

Recall is a completeness measure which varies from [0,1]. It is calculated as

$$Recall = \frac{|TP|}{|TP + FN|} = \frac{H \cap S}{H} \tag{7.7}$$

However, neither Precision nor Recall alone can accurately evaluate the match quality. In particular, Recall can easily be maximized at the expense of a poor Recall by returning all possible correspondences, i.e. the cross product of two input graphs. At the same time, a high Precision can be achieved at the expense of a poor Recall by returning only few (correct) correspondences. Therefore, it it necessary to consider both measures or a combined measures.

F-measure is a global measure of the matching quality. It varies from [0,1] and calculated as a harmonic mean of Precision and Recall:

$$F - Mesaure = \frac{2 * Recall * Precision}{Recall + Precision} \tag{7.8}$$

Notice that the complete reference mapping H must be known in advance in order to calculate both Precision and Recall. This opens a problem of its acquisition. The problem is that the construction of H is a manual process which, in the case of matching is quadratic in respect to the size of the graphs to be matched. This process turns to be unfeasible for large datasets. For instance, in the dataset we have exploited in this work, namely Google, Yahoo and Looksmart web directories, each structure has the order of 10^5nodes. This means that construction of H would would require the manual evaluation o 10^{10} mappings.

7.2 A dataset for evaluating Recall

We compute an approximation of H proposed in [6]. As from [6] we apply the proposed methodology to the Google, Yahoo and Looksmart web directories. The key idea is to rely on a reference interpretation or nodes, constructed by analyzing which documents have been classified in which nodes. The assumption is that the semantics of nodes can be derived from their pragmatics, namely by analyzing the documents that are classified under the given nodes, In particular, the underlying intuition is that two nodes have equivalent meaning if the sets of documents classified under those nodes have a meaningful overlap. The basic idea is therefore to compute the relationship hypotheses based on the co-occurrence of documents.

Consider the example presented in Figure 7.2. Let N_1 be a node in the first taxonomy and N_2 be a node in the second taxonomy. D_1 and D_2 stand for the sets of documents classified under the nodes N_1 and N_2 respectively. A_2 denotes the documents classified in the ancestor node of

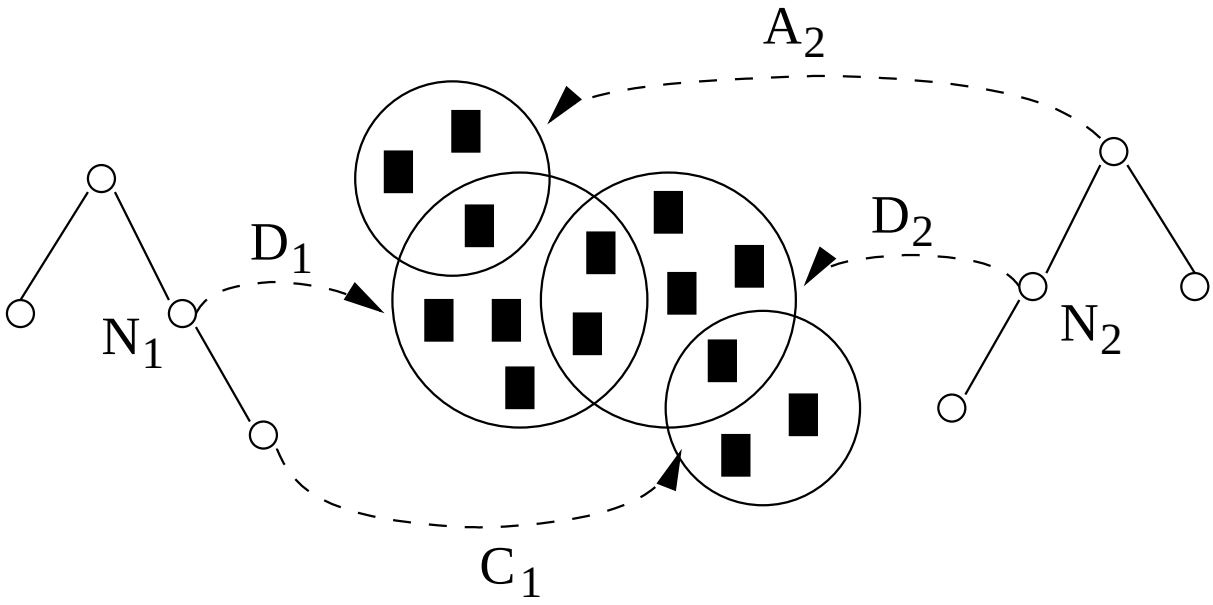

Figure 7.2: *TaxME*. Illustration of a document driven similarity assessment.

N_2; C_1 denotes the documents classified in the children nodes of N_1.

A simple *equivalence* measure is defined as

$$Eq(N_1, N_2) = \frac{|D_1 \cap D_2|}{|D_1 \cup D_2| - |D_1 \cap D_2 x|} \tag{7.9}$$

Notice that the range of $Eq(N_1, N_2)$ is [0,∞]. The intuition is that the more D_1 and D_2 overlap the bigger is $Eq(N_1, N_2)$ with $Eq(N_1, N_2)$ becoming infinite with $D_1 \equiv D_2$ Following what described in [6] $Eq(N_1, N_2)$ is normalized to [0,1]. The special case of $D_1 \equiv D_2$ is approximated to 1.

Given the two nodes N_1 and N_2 and the related sets D_1 and D_2 we introduce two additional sets: (i) the set of documents classified in the ancestor node of N_2, namely A_2, and (ii) the set of documents classified in the children nodes of N_1, namely C_1.

The *generalization* relationship holds when the first node has to be considered more general of the second node. Intuitively, it happens when the documents classified under the first node occur in the ancestor of the second node, or the documents classified under the second node occur in

the subtree of the first node. Following this intuition we can formalize the generalization hypothesis as

$$Mg(N_1, N_2) = \frac{|A_2 \cap D_1) \cup (C_1 \cap D_2)|}{|D_1 \cap D_2|} \tag{7.10}$$

The *specialization* relationship hypothesis $Lg(N_1, N_2)$ can be easily formulated exploiting the symmetry of the problem.

The TaxME dataset is computed starting from Google, Yahoo! and Looksmart. These web directories hold many interesting properties: they are widely known, they cover overlapping topics, they are heterogeneous, they are large, they address the same space of contents. All of this makes the working hypothesis of documents co-occurrence sustainable. The nodes are considered as categories denoted by lexical labels, the tree structures are considered as hierarchical relations, and the URLs classified under a given node are taken to denote documents. The following table summarize the total amount of processed data.

Table 7.1: Number of nodes and documents processed in the TaxME construction process.

Web directories	**Google**	**Looksmart**	**Yahoo!**
number of nodes	335902	884406	321585
number of urls	2425215	8498157	872410

Let us briefly summarize the five steps process used in the *TaxME* reference mapping construction.

Step 1 All three web directories are crawled, their hierarchical structure and their web content;

Step 2 The URLs that do not exist in at least one web directory are discarded;

Step 3 The nodes with a number of URLs under a given threshold (10 in the experiment) are pruned;

Step 4 A manual selection is performed with the goal of restricting the assessment of the similarity metric to the subtrees concerning the same topic; 50 pairs of sub trees are selected;

Step 5 For each of the subtree pair selected; an exhaustive assessment of correspondences holding between nodes is preformed. This is done by exploiting the equivalence metric defined in Eq. 7.9 and the corresponding generalization and specialization metrics. The TaxME similarity metric is computed as the biggest of the three metrics, namely

$$Sim_{TaxME} = max(Eq(N_1N_2), Lg(N_1, N_2), Mg(N_1, N_2)) \tag{7.11}$$

The distribution of mappings constructed using Sim_{TaxME} is depicted in Figure 7.3. for varying values of the metric.

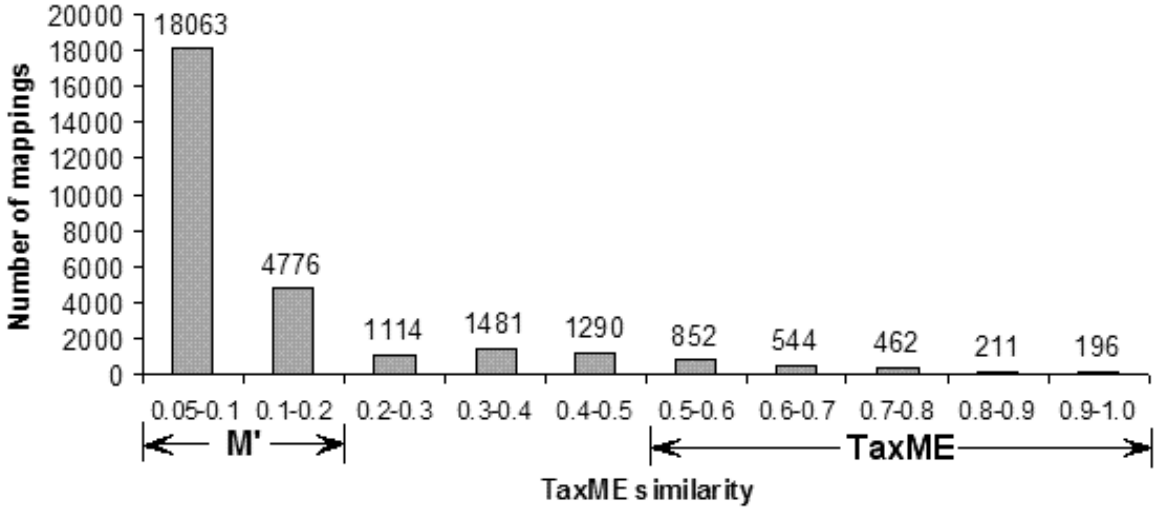

Figure 7.3: Distribution of mappings according to TaxME similarity metric

Notice that Sim_{TaxME} is very robust. The number of mappings is in fact very stable and grows substantially, of two orders of magnitude, only with a value of the metric less than 0.1. As a pragmatic decision, the mappings

with Sim_{TaxME} above 0.5 are taken to constitute the reference mappings TaxME. As a result, TaxME is composed from 2265 mappings. Half of them are equivalence relationships and half are generalization relationships. As depicted in Figure 7.4, TaxME is an incomplete reference mapping since it contains only part of the mappings in H. The key difference between Figures 7.4 and 7.1 is the fact that the complete reference mapping (the area inside the dotted circle in Figure 7.4) is simulated by exploiting an incomplete one (the area inside the dashed circle in Figure 7.4).

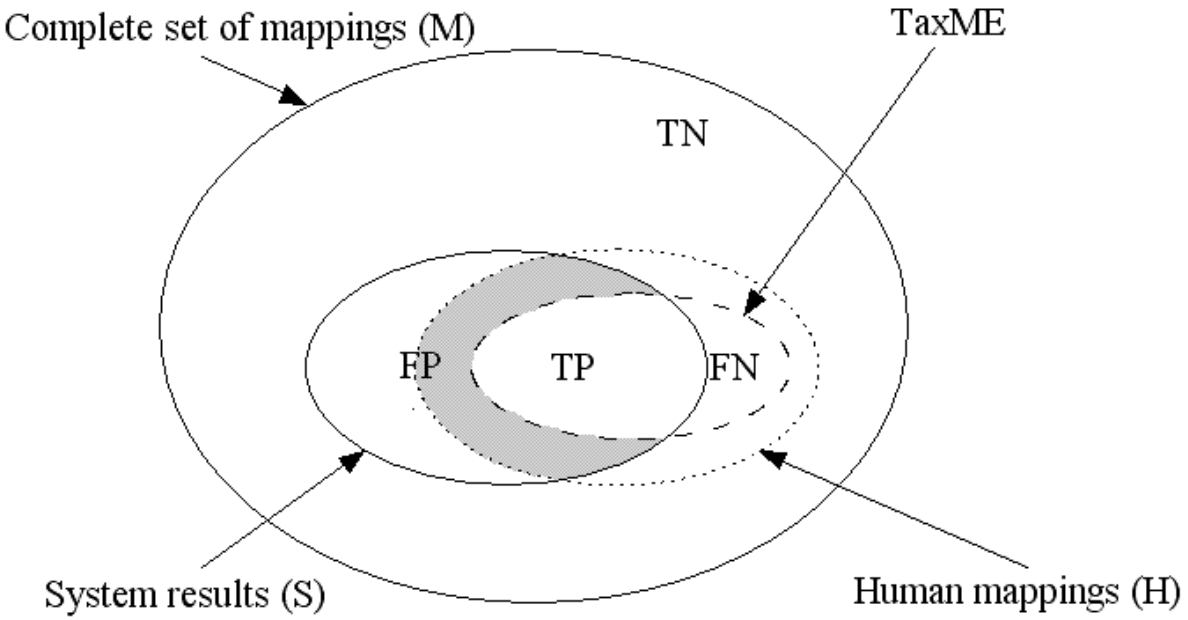

Figure 7.4: Mapping comparison using TaxME. TP, FN and FP stand for true positives, false negatives and false positives

However, if we assume that TaxME is a good representative of H we use Eq. 7.7 for an estimation of Recall. In order to ensure that this assumption holds a set of requirements have to be satisfied:

1. *Correctness*, namely the fact that $TaxME \subset H$ (modulo annotation errors).

2. *Complexity*, namely the fact that state of the art matching system experience difficulties when run on TaxME.

3. *Discrimination Capability*, namely the fact that different sets of mappings taken from TaxME are hard for the different systems.

4. *Incrementally*, namely the fact that TaxME allows for the incremental discovery of the weaknesses of the tested systems [1].

5. *Monotonicity*, namely the fact that the matching quality measures calculated on the subsets of the dataset does not differ substantially from the measures calculated on the whole dataset.

As discussed in [6] TaxME satisfies requirements 1,2,3,4. The fifth requirement is new, but TaxME satisfies it on wide range of the dataset samples.

In order to build TaxME 2, however, we need to verify another property of Sim_{TaxME}, namely its robustness. By robustness we mean the fact that the number of incorrect mappings is high only for very law values of Sim_{TaxME} and decreases very sharply as soon as these values increase. We need robustness as a it highlights the correspondence between the values of the similarity measure and the human observed similarity. To test the robustness of Sim_{TaxME}, we have randomly selected 100 mappings in 9 intervals of range 0.1 and one interval of range 0.05 as depicted in Figure 7.5 and manually evaluated their correctness. This resulted in a relatively small amount of manual work as we have analyzed around one thousand of mappings. The results are presented in Figure 7.5.

The results of this manual evaluation show that Sim_{TaxME} is very robust as:

- it is very stable with a small percentage of incorrect mappings for a very large range [0.3,1];

[1] We do not consider this property here as insignificant to our goals

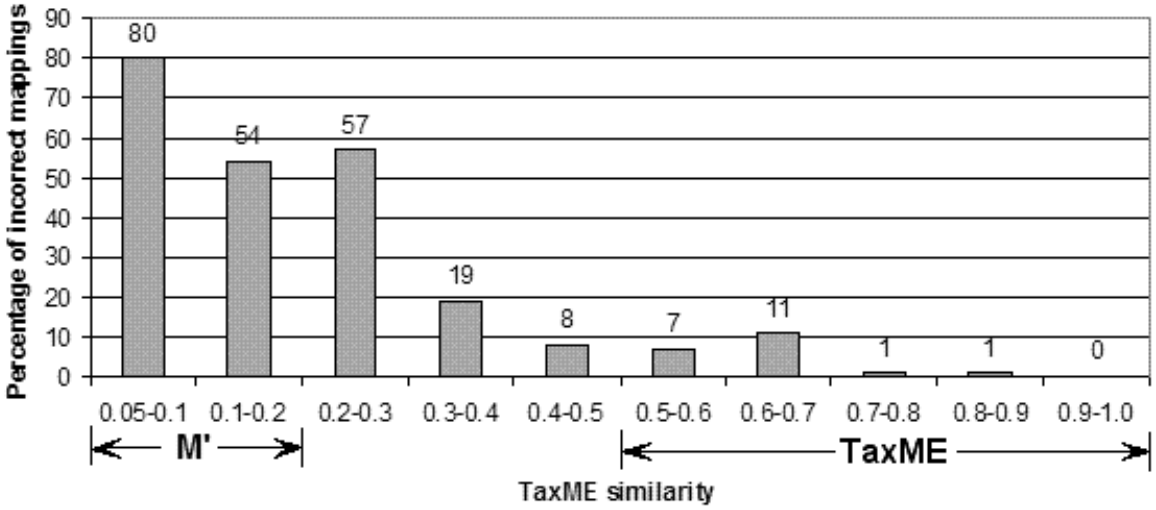

Figure 7.5: Distribution of incorrect mappings. Each column is calculated evaluating 100 randomly selected mappings

- the number of incorrect mappings becomes substantial for very small values of Sim_{TaxME}, namely with threshold less than 0.1.

7.3 A dataset for evaluating Precision

As from Eq 7.6 in order to evaluate Precision we need to know FP, which in turn, as from Figure 7.1, requires that we know *H*. However, as from Section 7.1, computing H in the case of a large scale matching task requires an implausible human effort. Notice also that we can not use the incomplete reference mapping composed from positive mappings i.e., *TaxME*, either. In this case, as shown in Figure 7.4, FP can not be computed. This is a case because $FP_{unknown} = S \bigcap (H - TaxME)$, marked as a grey area in Figure 7.4, is not known.

Our proposal in this thesis is to construct a reference mapping for the evaluation of both Recall and Precision, let us call il TaxME 2, defined as

$$TaxME2 = TaxME \cup N_{T2} \tag{7.12}$$

where N_{T2} is an incomplete mapping containing *only* negative mappings (i.e., $N_{T2} \subset M - H$ in Figure 7.4). Of course *TaxME 2* must be a good representative of M and therefore satisfy the three requirements described in the previous section and satisfied by *TaxME*. Notice that the request of correctness significantly limits the size of N_{T2} since each mapping has to be evaluated by a human annotator (i.e., $|N_{T2}| \ll |M - H|$). At the same time, N_{T2} must be big enough in order to be the source of meaningful results. Therefore, we require N_{T2} to be at least of the same size as *TaxME*, namely $|N_{T2}| \geq |TaxME|$.

N_{T2} is computed from complete mapping set M (as from Figure 7.1) in the following two macro steps:

Step 1 *Candidate mappings selection.* The goal of this step is to select a set M' where $M' \subseteq M$ which contains a big number of "hard" negative mappings.

Step 2 *Negative mappings selection.* The goal of this step is to filter all positive mappings from M'. In order to achieve this goal M' is first pruned to the size that allows manual evaluation of the mappings. Finally, the negative mappings are manually selected from the remaining mapping set.

Let us describe Step 1 and Step 2 in more detail.

7.3.1 Candidate mappings selection

The candidate mapping set M' is selected from M, as depicted in Figure 7.6.

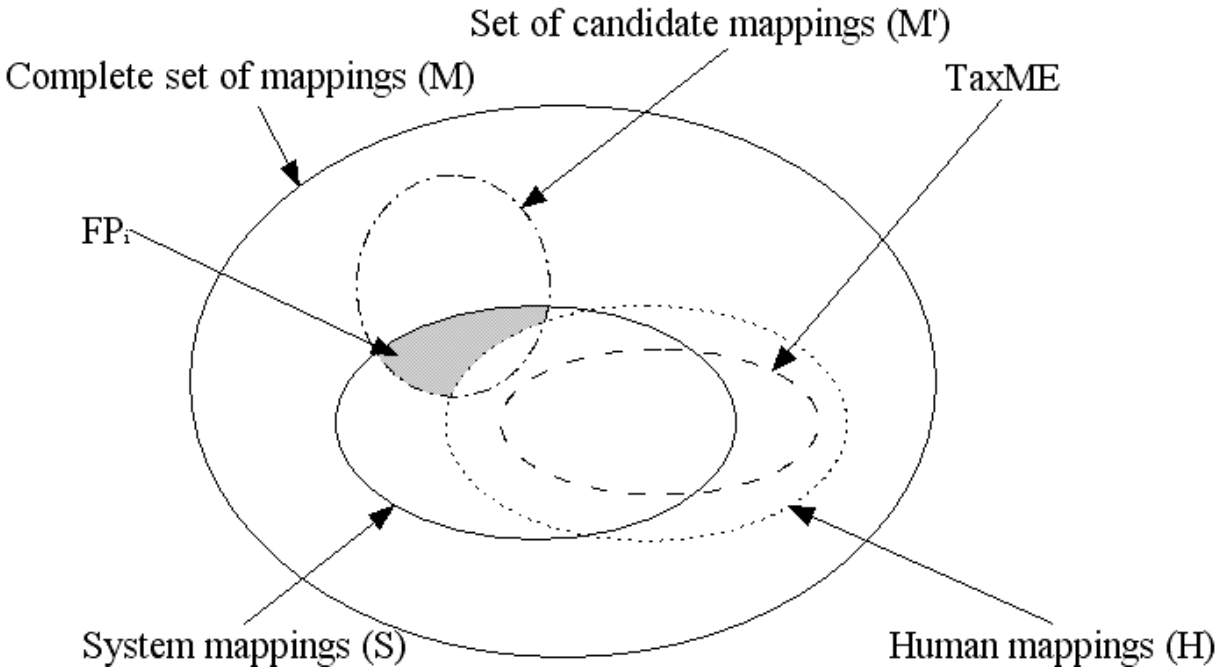

Figure 7.6: Mapping sets in TaxME 2. Gray area stands for FP_i a set of FP produced by a matching system on M'

The goal of this step is to ensure that M' contains a big number of "hard" negative mappings. Intuitively a "hard" negative mapping is mapping with high value of similarity measure which is incorrect according to manual annotation. Given the robustness of Sim_{TaxME} we have decided to exploit Sim_{TaxME} as a similarity measure for M' construction. Consider Figures 7.3 and 7.5. A big enough number of negative mappings can be obtained only for values of Sim_{TaxME} in the 0-0.2 range. As a pragmatic decision we have selected M' as a mappings having Sim_{TaxME} values in the 0.05-0.2 range. As from Figure 7.3, this allowed us to obtain 18063+4776=22836 candidate mappings.

7.3.2 Negative mappings selection

The negative mappings selection step is devoted to the computation of N_{T2}. The process is structured as follows:

Step 1 *Matching systems selection.* The goal of this step is to select a set of matching systems whose results are exploited for constructing N_{T2}. The set of the selected systems should be heterogeneous. By this we mean that the selected systems should make mistakes on different sets of mappings. Thus, the selected systems have to be the representatives of the different classes of the exerting matching techniques. This also prevents N_{T2} from being based towards a particular class of matching solutions.

Step 2 *Computation of negative mappings.* The goal of this step is to compute N_{T2} by exploiting the results obtained by running the selected matching systems on M'. In particular N_{T2} is computed from FP as $N_{T2} = \bigcup_i FP_i$, where FP_i stands for the FP produced by running the i-th matching system on M' (i.e., incorrect mappings in the set $S \bigcap M'$). The results of his exercise is depicted in Figure 7.6, where the grey area stands for FP_i. This construction schema ensures that N_{T2} will be hard for all exerting systems and discriminative given that the set of matching systems evaluated on M' is representative and heterogeneous. An implicit constraint is that the number of FPs produced by each of the systems should be comparable. This prevents the existence of a bias towards a particular class of matching solutions. Notice that the computation of FP (as from Eq. 7.2) requires the human annotation of the systems results.

Based on the classification of the matching systems originally presented [56] and then largely extended and augmented in [129][2] as a part of step 1 we have selected three matching systems namely COMA [31], Similarity Flooding (SF) [94] and S-Match (SM) [6, 64][3].

[2]See also http://www.ontologymatching.org

[3]In the evaluation discussed in this thesis we have used the basic version of S-Match and not the enhanced version described in [6]

During Step 2 we have executed COMA, SF and S-Match on M'. We also have manually evaluated the mappings found by the systems and selected the FP from them. Notice that we have not distinguished among different semantic relations while evaluating the matching quality. Therefore, for example, the mapping $A \sqsubseteq B$ produced by S-Match and $A_1 \equiv B_1$ produced by COMA have been considered as TP if $A \equiv B$ and $A_1 \sqsubseteq B_1$ are TP according to the human judgement. Finally we have computed N_{T2} as the union of FPs produced by the matching systems.

Table 7.2 provides a quantitative description of the content of N_{T2} and of the effort needed to build it.

Table 7.2: Total number of mapping and number of FP computed by COMA SF and S-Match on M'

	COMA	SF	SM
Found (S)	2553	2163	2151
Incorrect (FP)	870	776	781

As from the first row of the Table 7.2 the total number of annotated mappings was 2553+2163+2151=6867. Notice that is the 6 orders of magnitude lower than the number of mappings to be considered in the case of complete reference mapping. Notice also that the number of mappings per system is very balanced, as required . Figure 7.7 shows how the FPs produced by the systems are partitioned.

As from Figure 7.7, there are no FPs found by SM, COMA and SF, or even by SM and COMA together. There are the small intersections between the FPs produced by SM and by SF (0.1%) or by COMA and by SF (2.3%). These results justify our assumption that all 3 systems belong to different classes.

The final result is that N_{T2} consists of 2374 mappings. Notice that the size of N_{T2} is not equal to the sum of the FPs reported in the second row of Table 7.2 since, as from Figure 7.7, there is some intersection among these

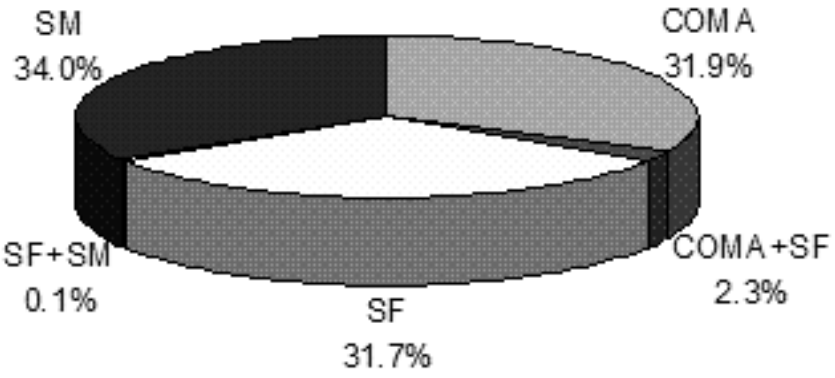

Figure 7.7: Partitioning of FPs computed by COMA, SF and S-Match on M'

sets. The union of N_{T2} with TaxME has allowed us to compute a reference mapping TaxME 2, good for the evaluation of both Recall and Precision, of 2265+2374=4639 mappings.

7.4 Evaluating dataset

In this section we present the evaluation of the Complexity, Discrimination Capability and Monotonicity of the TaxME 2. In particular we exploit the results of twelve matching systems (Apfel [38], CMS, ctxMatch2 [18], OLA [46], OMAP [131] and seven systems participated in OAEI-2006 [43] evaluation). For the systems we use the default settings or, if applicable, the settings provided by the authors for the OAEI-2005, 2006 [45, 43] evaluations. We also compare the results of the matching systems with the results of the systems exploited in the dataset construction process (COMA, SF and SM). The evaluation results, in terms of TP and FP, are presented in Table 7.3.

7.4.1 Complexity

Figure 7.8 presents the Precision of the systems when evaluated on *TaxME2*.

Table 7.3: Number of FP and TP on TaxME 2 dataset

	Apfel	CMS	ctxMatch	OLA	OMAP	COMA	SF	SM	Hmatch
FP	670	367	299	1356	1113	870	776	781	632
TP	269	319	298	724	694	876	218	669	303

	Falcon	Automs	RiMOM	OCM	COMA++	Prior
FP	1513	730	1416	712	1343	1085
TP	1030	330	915	356	608	552

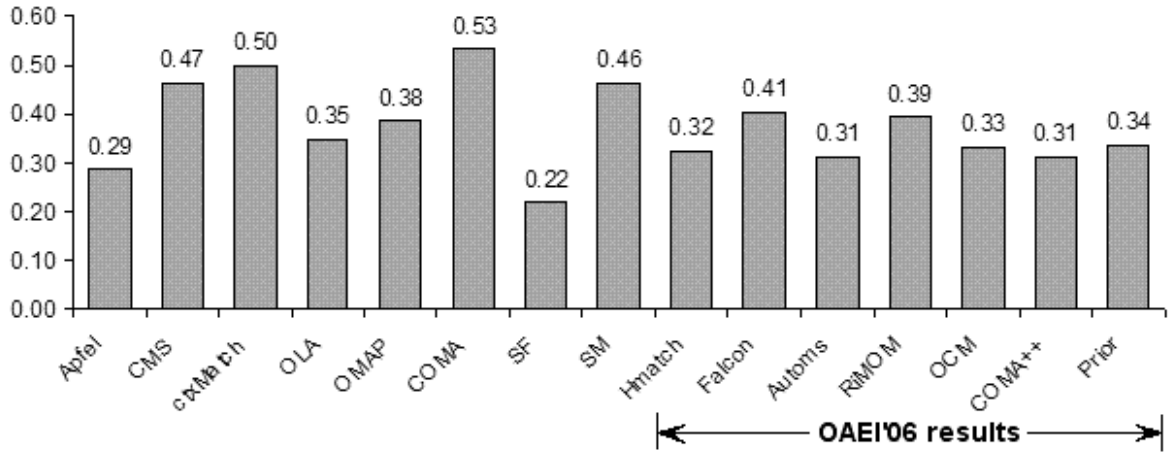

Figure 7.8: Evaluation results. Precision on TaxME 2 dataset

As from Figure 7.8 the maximum Precision is about 0.5, a value which significantly lower than the results obtained with the other datasets. For example, the average Precision demonstrated by Falcon, FOAM, CMS and OMAP on the real world part of the semantic tests (problems 301, 302, 303, 304) in the OAEI-2005 evaluation [45] was in the 0.91-0.93 range.

Figure 8.8 illustrates the Recall of the matching systems while Figure 7.10 presents the F-Measure as an aggregated matching quality measure. The best F-Measure is 0.45 what is significantly lower than the results demonstrated by the systems on the other datasets.

The results of our evaluation highlight the complexity of TaxME2. The other interesting observation is that the systems exploited in the dataset construction process demonstrate a performance which is comparable with

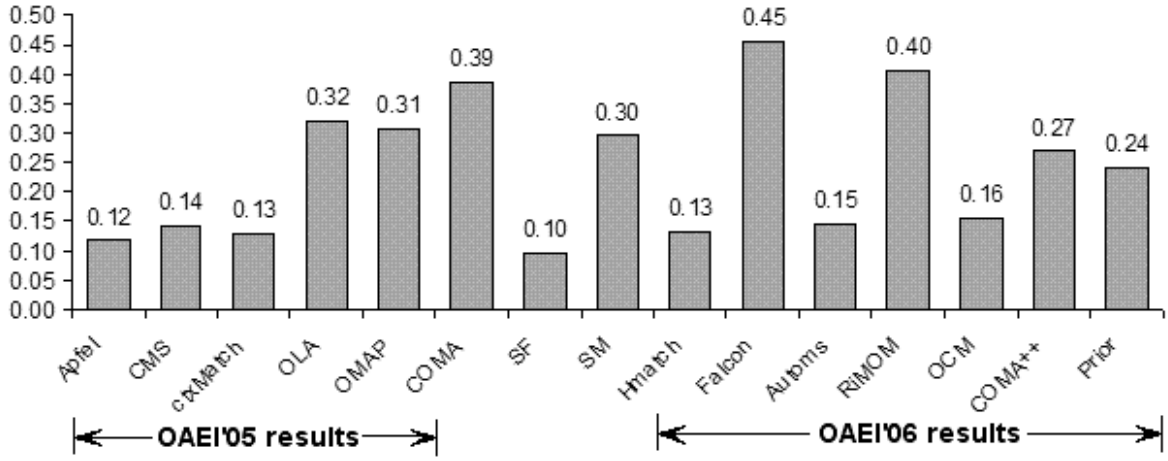

Figure 7.9: Evaluation results. Recall on TaxME 2 dataset

the other systems. In fact all evaluated systems have experienced the same problems as COMA, SF and SM. This fact justifies that TaxME 2 reflects the inherent properties of real world problems. At the same time, it is still very hard for the state of the art matching systems.

7.4.2 Discrimination capability

Consider Figures 7.11 and 7.12.

They present the partitioning of the FPs and the TPs in TaxME 2 according to the results of the matching systems. As from Figure 7.11 all matching systems provided the correct results only for 20% of the FPs while 25% of the FPs are incorrectly found by ten or more matching systems. At the same time 29% of the TPs are not found by any of the matching systems and 65% of the TPs are found by 2 or less of the matching systems. Figures 7.11 and 7.12 illustrate the fact that the different systems experience difficulties on different parts of the dataset (i.e., *TaxME 2* is discriminative or it is hard for the different systems in the different ways).

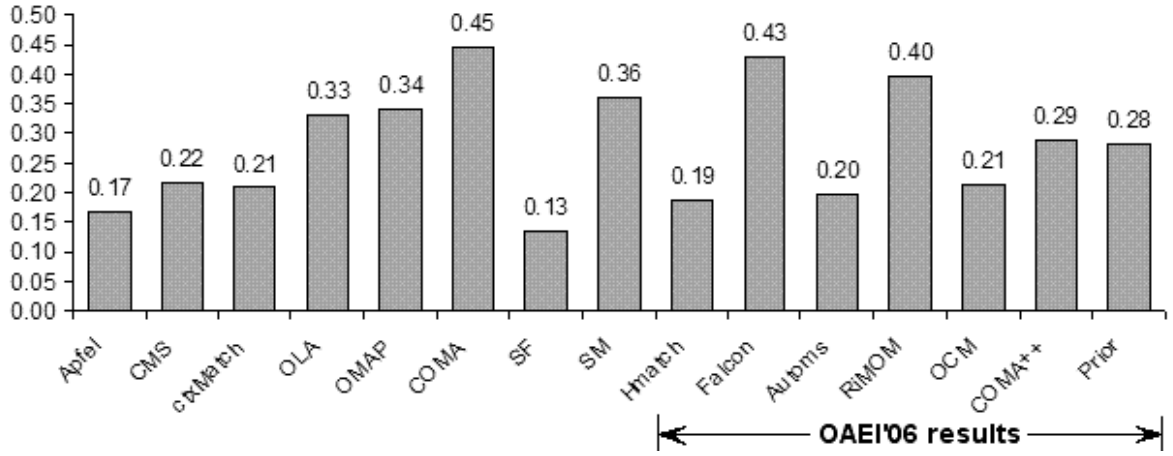

Figure 7.10: Evaluation results. F-Measure on TaxME 2 dataset

7.4.3 Monotonicity

A mapping dataset is said to demonstrate a monotonous behavior if the matching quality measures calculated on its subsets of gradually increasing size converge to the values obtained on some whole dataset, for all matching systems evaluated on a some datasets. This property illustrates the fact that the dataset as a whole and its parts are not biased to the particular matching solution(s). It also shows how the gradual increase in the dataset size influence on the results of the matching systems in terms of the matching quality and gives a clue of whether the further increase in the dataset size may significantly influence on the values of the matching quality measures.

In order to evaluate the monotonicity property we randomly sampled 50, 100, 200, 500, 1000 mappings form TaxME 2 dataset. For example, in order to obtain a 100 mappings sample 50 mappings were randomly selected and added to the previously selected 50 mappings sample. Than the matching quality measures for the matching systems were calculated on the samples of various size. An error was computed as

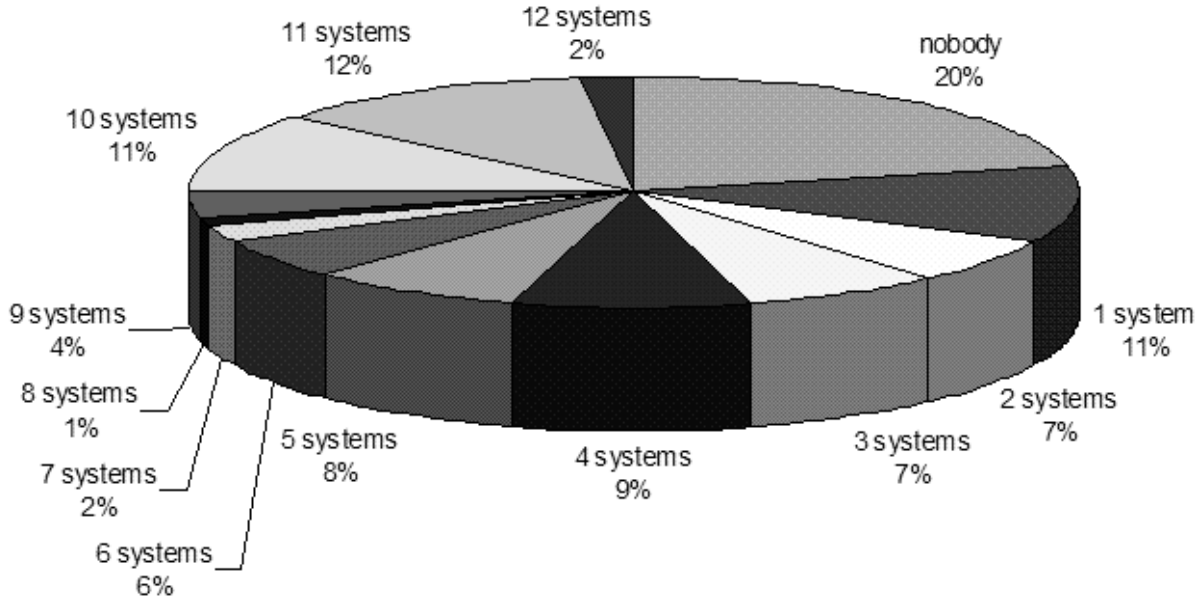

Figure 7.11: Partitioning of FPs found by matching systems in TaxME 2 dataset according to the number of systems that found them

$$Error_{Measure} = \frac{|Measure_{sample} - Measure_{dataset}}{Measure_{dataset}} \tag{7.13}$$

where $Measure_{sample}$ stands for a matching quality measure calculated on the sample; $Measure_{dataset}$ denotes a matching quality measure calculated on the whole dataset. Thus, for example, if a matching system had on TaxME 2 dataset Precision of 0.2 and on the 100 mappings sample randomly selected from the dataset its Precision is 0.21 the error of the system is $Error_{Precision} = \frac{|0.21-0.2|}{0.2} = 0.05$

$Error_{Precision}$, $Error_{Recall}$ and $Error_{F-Measure}$ for various sample sizes averaged for 10 sample selections are depicted on Figures 7.13, 7.14 and 7.15 respectively.

Notice that the error drops very quickly with the sample size increase. Therefore, the matching quality measures obtained on the randomly selected samples of various size converge quickly to their values of the "full" dataset. In particular given 500 mappings sample $Error_{Precision}$ and $Error_{Recall}$

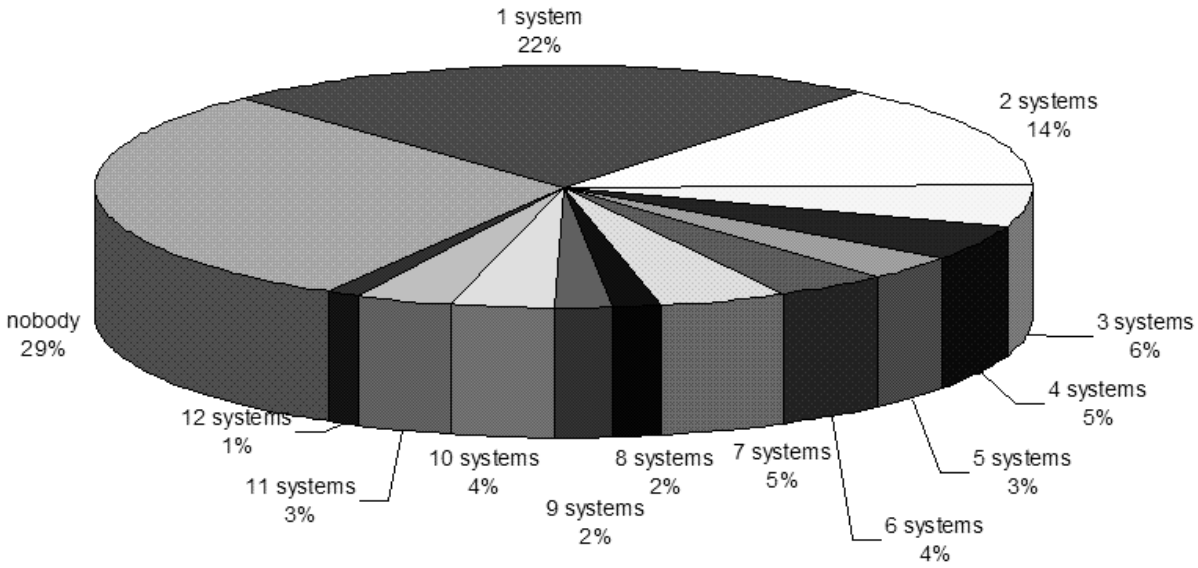

Figure 7.12: Partitioning of TPs found by matching systems in TaxME 2 dataset according to the number of systems that found them

is less than 10% what, given the results depicted on Figures 7.8 and 8.8 correspond to 0.02-0.05 difference in absolute values. Considering this difference as marginal we conclude that TaxME 2 is monotonous.

7.5 A dataset for structure preserving semantic matching evaluation

We have evaluated the matching quality of the structure preserving semantic matching algorithms on 132 pairs of first order logic terms. Half of the pairs were composed of the equivalent terms (e.g., *journal(periodical-publication)* and *magazine (periodical-publication)*) while the other half were composed from similar but not equivalent terms (e.g., *web-reference(publication-reference)* and *thesis-reference (publication-reference)*). The terms were extracted from different versions of the Standard Upper Merged Ontology (SUMO)[4] and the Advance Knowledge Transfer (AKT)[5] ontologies.

[4] `http://ontology.teknowledge.com/`
[5] `http://www.aktors.org`

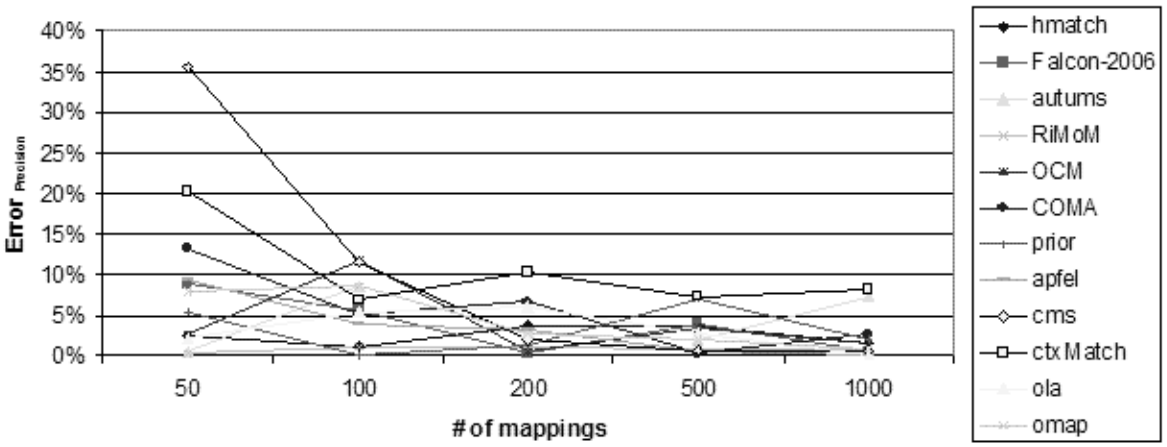

Figure 7.13: $Error_{Precision}$ depending from the sample size

We extracted all the differences between versions 1.50 and 1.51, and 1.51 and 1.52 of the SUMO ontology and between versions 1, 2.1 and 2.2 of the AKT-portal and AKT-support ontologies[6]. These are both first order ontologies, so many of these differences mapped well to the potential differences between terms that we are investigating. However, some of them were more complex, such as differences in inference rules, or consisted of ontological objects being added or removed rather than altered, and had no parallel in our work. These pairs of terms were discarded and our tests were run on all remaining differences between these ontologies. Therefore, we have simulated the situation when the service descriptions are defined exploiting the two versions of the same ontology.

7.6 Summary

In this chapter we have presented a method for large scale dataset construction good for evaluation of both Recall and Precision. We also introduced a set of properties essential for the dataset to be used for matching evalu-

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

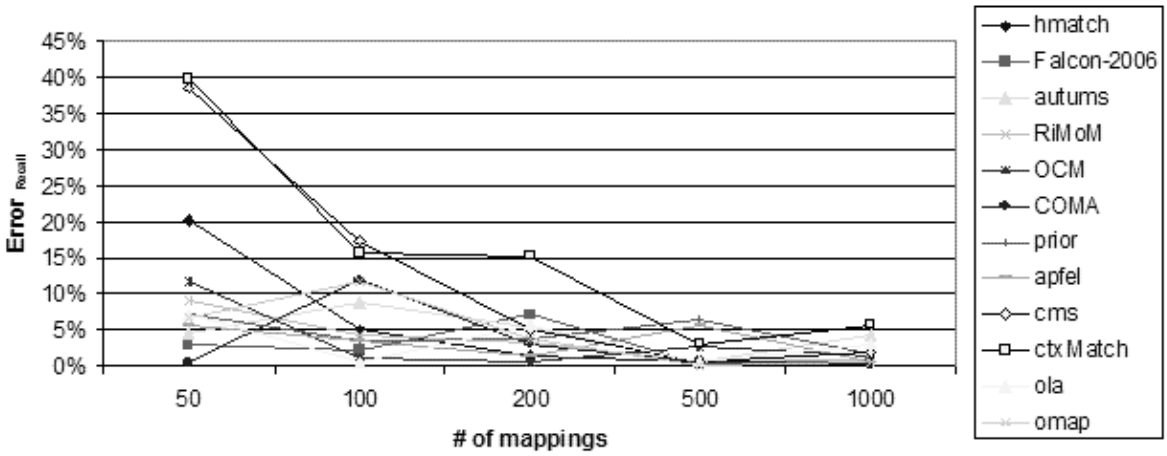

Figure 7.14: $Error_{Recall}$ depending from the sample size

ation. Finally we evaluated the dataset exploiting a wide range of state of the art matching systems. According to the evaluation results the dataset comply to the relevant properties.

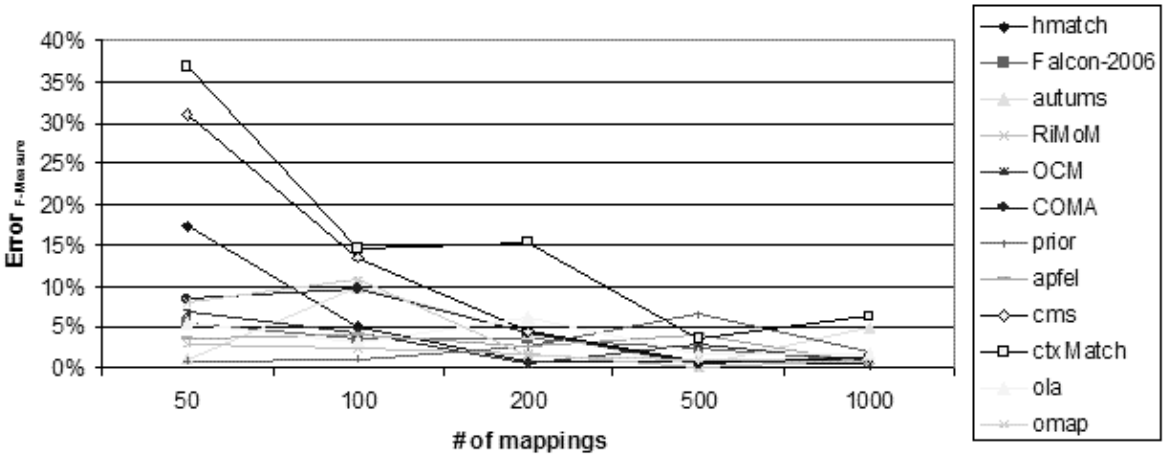

Figure 7.15: $Error_{F-Measure}$ depending from the sample size

Chapter 8

Evaluation

This chapter provides evaluation results for S-Match system and an implementation of the structure preserving semantic matching algorithms. The results have been obtained and compared based on the measures outlined in §7.1.

Material presented in this chapter has been developed in collaboration with Pavel Schvaiko and published in [58, 59, 60, 64].

In this chapter we first discuss evaluation setup (§8.1). Then, we provide evaluation results (§8.2).

8.1 Evaluation setup

The evaluation was performed on seven matching tasks from different application domains, see Table 8.1. There are three matching tasks from a business domain (#1,3,5). The first business example (#1) describes two company profiles: a Standard one (mini) and Yahoo Finance (mini), while, #5, represents their full versions. The third business example (#3) deals with BizTalk[1] purchase order schemas. There is one matching task from an academy domain (#2). It describes courses taught at Cornell University (mini) and at the University of Washington (mini). Finally, there

[1] http://www.microsoft.com/biztalk

Table 8.1: Some indicators of the complexity of the test cases

#	matching task	max depth	#nodes	#labels per tree	concepts of nodes
1	Yahoo(mini) *vs.* Standard(mini)	2/2	10/16	22/45	Conjunctive Disjunctive
2	Cornell *vs.* Washington	3/3	34/39	62/64	Conjunctive Disjunctive
3	CIDX *vs.* Excel	3/3	34/39	56/58	Conjunctive Disjunctive
4	Looksmart *vs.* Yahoo	10/8	140/74	222/101	Conjunctive Disjunctive
5	Yahoo *vs.* Standard	3/3	333/115	965/242	Conjunctive Disjunctive
6	Google *vs.* Yahoo	11/11	561/665	722/945	Conjunctive Disjunctive
7	Google *vs.* Looksmart	11/16	706/1081	1048/1715	Conjunctive Disjunctive

are three matching tasks on general topics (#4,6,7) as represented by the well-known web directories, such as Google[2], Yahoo![3], and Looksmart[4]. Table 8.1 provides some indicators of the complexity of these test cases[5].

The reference mappings (also called expert mappings) for some of these problems (namely for the tasks #1,2,3) were established manually. Then, the results computed by the systems have been compared with expert mappings. It is worth noticing that the task of creation of expert mappings is an error-prone and a time consuming one. Even if for the moment of writing this thesis we have created expert mappings for the biggest matching tasks (#6,7) of Table 8.1, we do not report these findings in this thesis. Addressing in full detail the emerged issues along that process as well as the matching results achieved is out of scope of this thesis, see for some details [6, 60]. Thus, in this evaluation study we focus mostly on the performance characteristics of S-Match, involving large matching tasks, namely schemas with hundreds and thousands of nodes. Notice, scalability properties of matching systems is among the most important problems of schema

[2]http://www.google.com/Top/

[3]http://dir.yahoo.com/

[4]http://www.looksmart.com/

[5]Source files and description of the schemas tested can be found at our project web-site, experiments section: http://www.dit.unitn.it/ accord/

matching (in general) these days, see e.g., [14, 34]. Quality characteristics of the S-Match results which are presented here address only medium size schemas. The results of S-Match in a large-scale evaluation are presented in Section 7.4.

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 mappings which express similarity between schema elements. 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 expert mappings. The second observation is that, since S-Match returns a matrix of relations, while all other systems return a list of the best mappings, we used some filtering rules. More precisely we have the following two rules: (i) discard all the mappings 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 mappings standing for those relations.

As match quality measures we have used the following indicators: precision, recall, overall, and F-measure. As a performance measure we have used time. It estimates how fast systems are when producing mappings fully automatically. Time is very important for us, 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 mappings have been computed: from the first schema 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 14).

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. This threshold has been defined after experimentation on several schema matching tasks (see for details the URL in footnote 14). Finally, all the element level matchers of the third approximation level (e.g., gloss-based matchers) were not involved in the evaluation since all the matching tasks under consideration were successfully resolved by the matchers of Table 4.1 which belong to the first and the second approximation levels; see [60] for the preliminary evaluation results of matchers belonging to the third approximation level as well as for the tasks where they are useful.

While computing precision and recall for structure preserving semantic matching algorithms we have considered the correspondences holding among first order terms rather than the nodes of the term trees. Thus, for instance, *journal(periodical-publication*$_1$*)=magazine(periodical-publication*$_2$*)* was considered as single correspondence rather than two correspondences, namely *journal=magazine* and *periodical-publication*$_1$*=periodical-publication*$_2$.

8.2 Evaluation results

We present the time performance results for all the tasks of Table 8.1, while quality results, as from the previous discussion are possible to estimate only for some of the matching tasks (#1,2,3). The evaluation results for the matching problems #1,2,3 are shown in Figures 8.1,8.2,8.3.

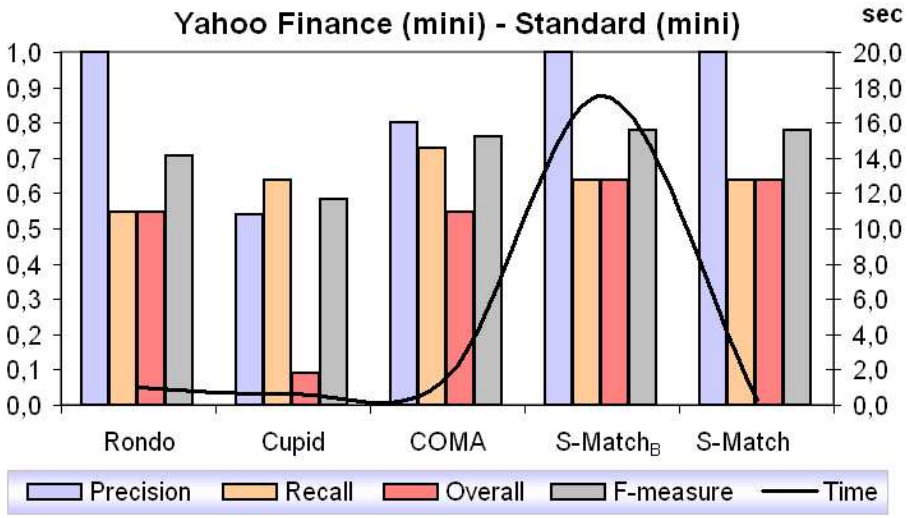

Figure 8.1: Evaluation results: Yahoo Finance (mini) vs. Standard (mini), test case #1

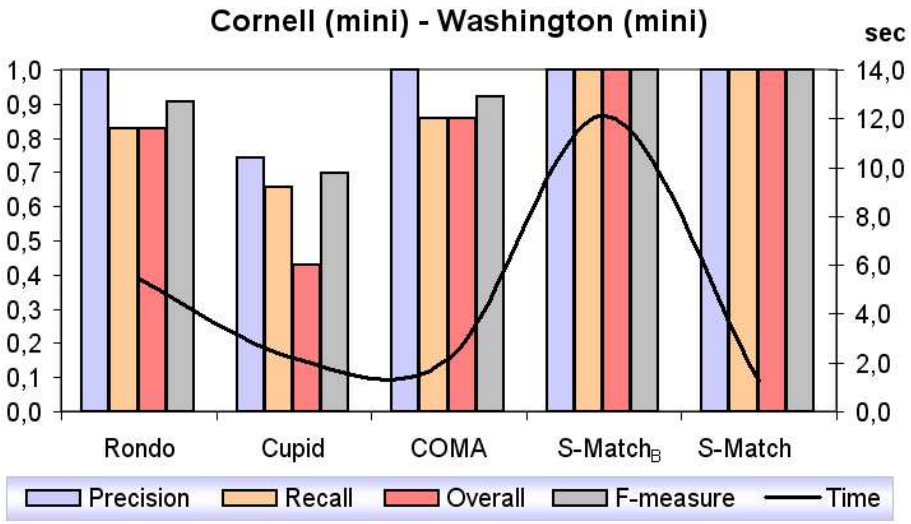

Figure 8.2: Evaluation results: Cornell (mini) vs. Washington (mini), test case #2

For example, in Figure 8.2, 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 mappings, S-Match will return all of them and only the correct ones.

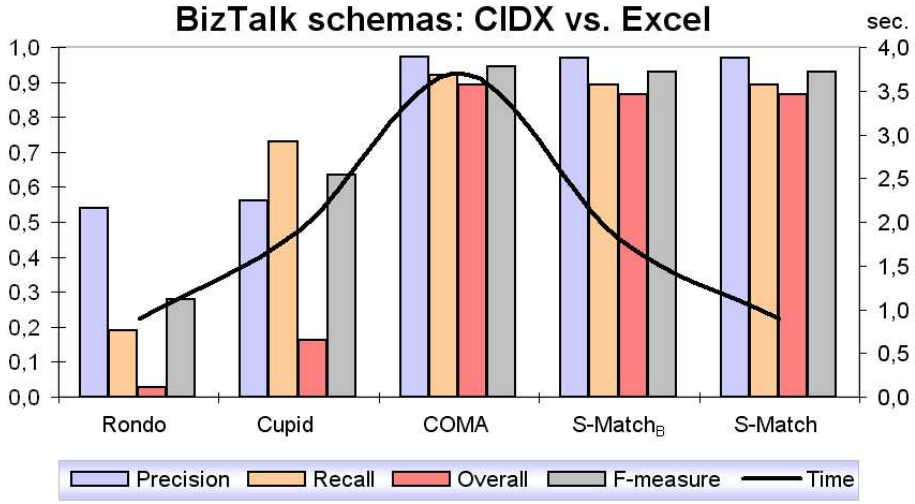

Figure 8.3: Evaluation results: CIDX vs. Excel, test case #3

For a pair of BizTalk schemas: CIDX vs. Excel, 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.

The time performance results obtained for the matching tasks #4,5,6,7 of Table 8.1 are presented in Figure 8.4. Cupid went out of memory on all the tasks. Therefore, we present the results for other systems.

In the case of Looksmart-Yahoo matching problem the trees contain about hundred nodes each. S-Match works about 18% faster than $S-Match_B$ and about 2% slower than COMA. SF, in turn, works about 3 times faster than S-Match. The relatively poor improvement (18%) occurs because our optimizations are implemented in a straightforward way. More

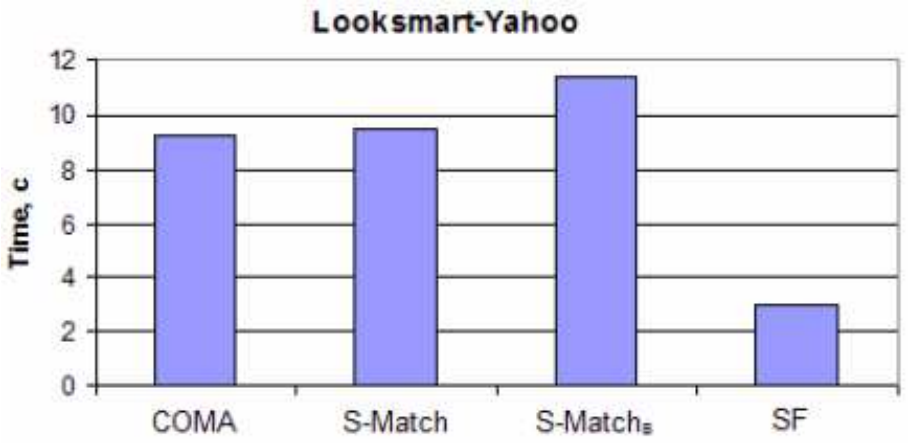

Figure 8.4: Execution times: Looksmart vs. Yahoo, test case #4

precisely, on small trees (e.g., test case #4) a big constant factor[6] dominates the growth of all other components in S-Match computational complexity formula.

On Yahoo-Standard matching problem S-Match works about 40% faster than $S - Match_B$. It performs 1% faster than COMA and about 5 times slower than SF. 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. The optimizations cannot significantly influence the system performance.

The next two matching problems are much bigger than the previous ones. They contain hundreds and thousands of nodes. On these trees SF went out of memory. Therefore, we provide the results only for the other systems. In the case of Google-Yahoo matching task S-Match is more than 6 times faster than $S - Match_B$. COMA performs about 5 times slower than the optimized version. These results suggest that the optimizations described in this thesis are better suited for big trees. In the case of the biggest matching problem, involving Google-Looksmart, S-Match performs about 9 times faster than COMA, and about 7 times

[6]This is also known in the literature as an implementational constant.

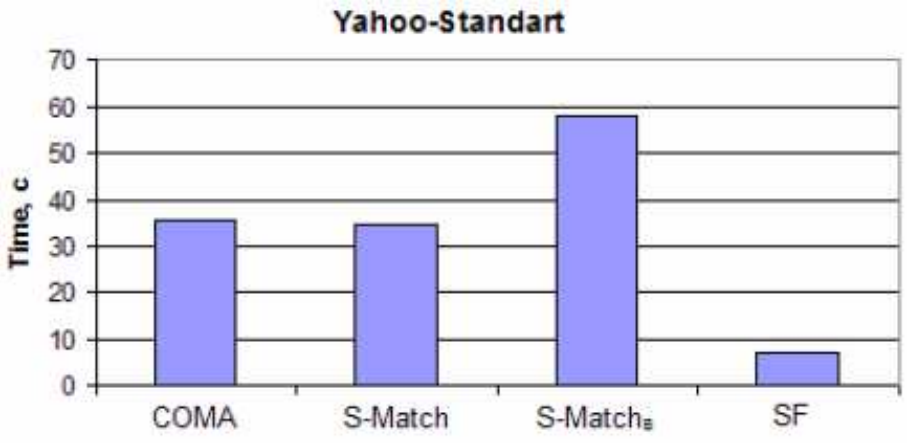

Figure 8.5: Execution times: Yahoo vs. Standard, test case #5

faster than $S - Match_B$.

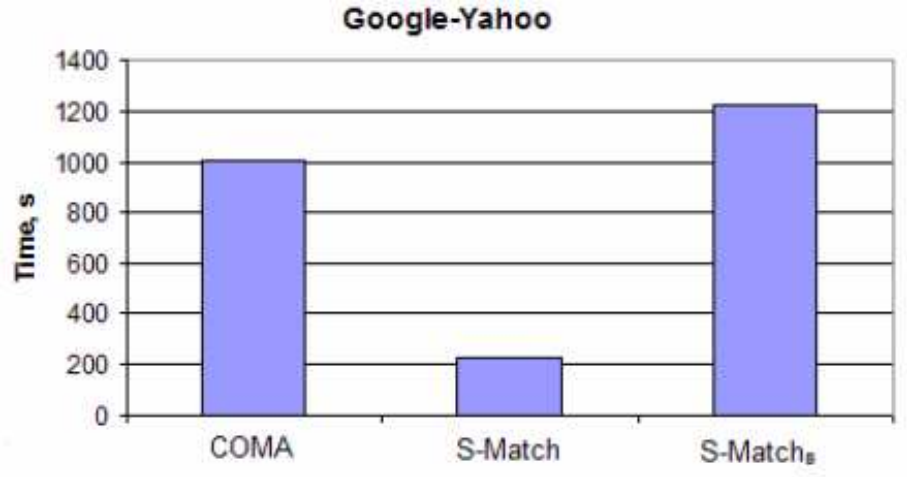

Figure 8.6: Execution times: Google vs. Yahoo, test case #6

Having considered matching tasks of Table 8.1, we conclude that S-Match performs (in terms of execution time) slightly slower than COMA and SF on the schemas with one up to three hundred of nodes (see, Figures 8.4,8.5). At the same time, S-Match is considerably faster on the schemas with more than five hundreds nodes (see, Figures 8.6,8.7), thereby indicating system scalability.

Interestingly enough our exact structure matching algorithm was able to find 36 correct correspondences what stands for 54% of Recall with

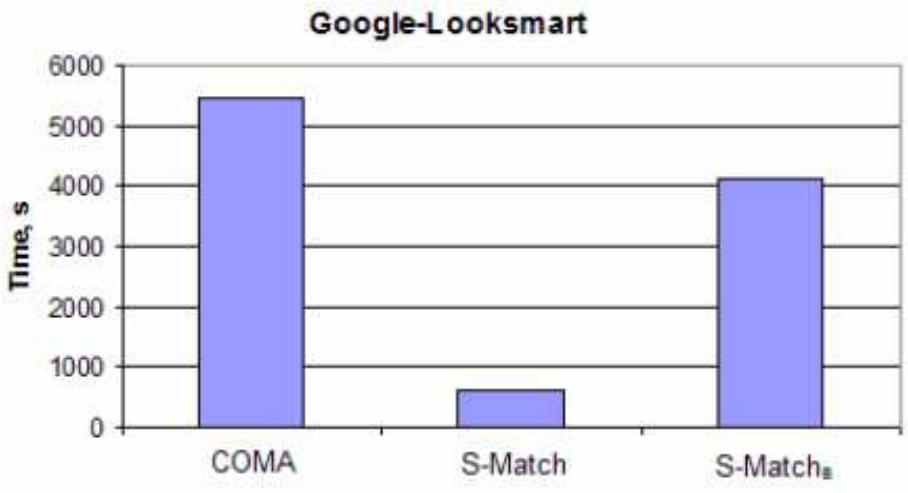

Figure 8.7: Execution times: Google vs. Looksmart, test case #7

100% Precision. All mismatches (or correct correspondences not found by the algorithm) corresponded to structural differences among first order terms which exact structure matching algorithm is unable to capture. The examples of correctly found correspondences are given below:

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

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

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

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

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

Figure 8.8 presents the matching quality measures depending on the cut-off threshold value for approximate structure preserving matching algorithm. As from Figure 8.8, the algorithm demonstrates high matching

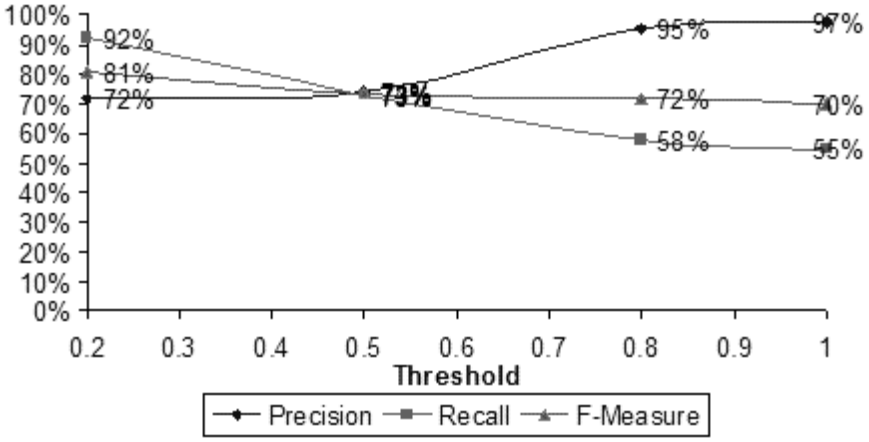

Figure 8.8: The matching quality measures depending on threshold value for approximate structure matching algorithm

quality on the wide range of threshold values. In particular, F-Measure values exceed 70% for the given range. Table 8.2 summarizes the time performance of the matching algorithm. It presents the average time taken by

Table 8.2: Time performance of approximate structure matching algorithm (average on 132 term matching tasks)

	Node matching Step 1 and 2	Node matching Step 3 and 4	Structure matching
Time, ms	134.1	3.3	0.9

the various steps of the algorithm on 132 term matching tasks. As from the table, Step 1 and 2 of the node matching algorithm significantly slow down the whole process. However these steps correspond to the linguistic preprocessing that can be performed once offline (see §4.2). Given that the terms can be automatically annotated with the linguistic preprocessing results (see §4.2) once when changed, the overall runtime is reduced to 4.2 ms, which corresponds roughly to 240 term matching tasks per second.

8.3 Summary

In this chapter we have presented comparative evaluation of semantic matching against the other state of the art systems. The results are encouraging and empirically prove the strength of our approach.

However, as our evaluation results show, it is very difficult to know a priori the quality to expect from a matching system. Matching tasks are so different that a system can perform very well on some data and not that well on some other. This means that in order to justify the claim of a matching system to be generic, a lot of work has to be done yet, especially to address all the issues that arise when dealing with large-scale matching tasks.

Part V

Conclusions

Chapter 9

Summary

In this thesis we have proposed, developed and implemented a novel approach to ontology matching, called semantic matching, discussed its technical details and thorough evaluation. Specifically, the main findings of each chapter of the thesis are summarized one by one in sequel.

We showed that there are many applications that may need ontology matching. We showed that there are various existing ways of expressing knowledge that are found in diverse applications. These ways of expressing knowledge can be viewed as different forms of ontologies that may need to be matched (Chapter 1). Unlike many other works, we aimed to treat the matching problem in a unified way and provide a common roof under the heading of ontology matching for many existing instantiations of this problem, such as schema matching, catalog matching, etc. Finally, we technically defined the ontology matching problem.

We showed that ontology matching can take advantage of innumerable basic techniques composed and supervised in diverse ways (Chapter 2). We provided a systematic view over the available techniques by classifying them and providing some guidelines which help in identifying families of matching methods.

We reviewed existing schema-based matching systems which emerged

during the last decade (Chapter 3). These were presented in light of the classifications developed in Chapter 2. We also pointed to concrete basic matcher and matching strategies used in the considered systems. We summarized some global observations concerning the presented systems and outlined a number of constant features that are shared by the majority of them.

Having analyzed in detail the state of the art we proposed an approach to ontology matching called semantic matching (Chapter 4). We developed a new semantic matching algorithm called S-Match. We discussed with the help of examples and pseudo-code the main macro steps of the algorithm.

We showed how efficiency of the basic semantic matching algorithm can be improved within the semantic matching settings (Chapter 5).

We demonstrated how a semantic matching framework can be extended in order to deal with more complex structures in (Chapter 6). This illustrates extensibility of the semantic matching algorithms and helps in their application to the various problems.

We discussed the thorough evaluation of the matching systems in Chapter 7. In particular we have focused on the test bed generation. As a part of this effort we have discussed the TaxME a large scale data set constructed from Google, Yahoo! and Looksmart web directories.

We performed an evaluation of the semantic matching approach (Chapter 8). As our comparative evaluation shows it is very difficult to know a priori the quality to expect from a matching system. Matching tasks are so different that a system can perform very well on some, usually small test cases, while not that well on some other, usually large-scale test cases. Analysis of the mistakes done by a system opens a number of ways for further improvements.

We would like to make a final remark. The remark concerns some assumptions and limitations of the proposed solution. In particular, the pro-

posed solution naturally assumes that the ontologies to be matched have a meaningful overlap, thus these are worth been matched. The proposed approach reduces the conceptual heterogeneity only to a certain extent, though, for example, cases such as geometry axiomatized with points as primitive objects and geometry axiomatized with spheres as primitive objects are not handled. At last, although we have aimed at producing a generic matching solution, a lot of work still needs to be done.We have only investigated matching of tree-like structures produced out of classifications and catalogs, while it has still to be analyzed whether the presented solution will properly handle the trees generated, e.g., out of relational schemas and the other forms of ontologies.

Bibliography

[1] Zharko Aleksovski, Michel Klein, Warner ten Kate, and Frank van Harmelen. Matching unstructured vocabularies using a background ontology. In *Proceedings of the International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, 2006.

[2] Yigal Arens, Chun-Nan Hsu, and Craig Knoblock. Query processing in the SIMS information mediator. In *Readings in Agents*, pages 82–90. AAAI press, 1996.

[3] Paolo Atzeni, Paolo Cappellari, and Philip Bernstein. ModelGen: Model independent schema translation. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 1111–1112, 2005.

[4] Paolo Atzeni, Paolo Cappellari, and Philip Bernstein. Model-independent schema and data translation. In *Proccedings of the International Conference on Extending Database Technology (EDBT)*, pages 368–385, 2006.

[5] David Aumüller, Hong-Hai Do, Sabine Maßmann, and Erhard Rahm. Schema and ontology matching with COMA++. In *Proceedings of the International Conference on Management of Data (SIGMOD), Software Demonstration*, 2005.

[6] Paolo Avesani, Fausto Giunchiglia, and Mikalai Yatskevich. A large scale taxonomy mapping evaluation. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 67–81, 2005.

[7] S. Banerjee and T. Pedersen. Extended gloss overlaps as a measure of semantic relatedness. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 267–270, 2003.

[8] Carlo Batini, Maurizio Lenzerini, and Shamkant Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[9] Domenico Beneventano, Sonia Bergamaschi, Stefano Lodi, and Claudio Sartori. Consistency checking in complex object database schemata with integrity constraints. *IEEE Transactions on Knowledge and Data Engineering*, 10(4):576–598, 1998.

[10] Sonia Bergamaschi, Domenico Beneventano, Silvana Castano, and Maurizio Vincini. MOMIS: An intelligent system for the integration of semistructured and structured data. Technical Report T3-R07, Università di Modena e Reggio Emilia, Modena (IT), 1998.

[11] Sonia Bergamaschi, Silvana Castano, and Maurizio Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1):54–59, 1999.

[12] Jacob Berlin and Amihai Motro. Database schema matching using machine learning with feature selection. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 452–466, 2002.

[13] Philip Bernstein, Alon Halevy, and Rachel Pottinger. A vision of management of complex models. *SIGMOD Record*, 29(4):55–63, 2000.

[14] Philip Bernstein, Sergei Melnik, M. Petropoulos, and Christoph Quix. Industrial-strength schema matching. *SIGMOD Record*, 33(4):38–43, 2004.

[15] Paolo Bouquet, Marc Ehrig, Jérôme Euzenat, Enrico Franconi, Pascal Hitzler, Markus Krötzsch, Luciano Serafini, Giorgos Stamou, York Sure, and Sergio Tessaris. Specification of a common framework for characterizing alignment. Deliverable D2.2.1, Knowledge web NoE, 2004.

[16] Paolo Bouquet, Bernardo Magnini, Luciano Serafini, and Stefano Zanobini. A SAT-based algorithm for context matching. In *Proceedings of the International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT)*, pages 66–79, 2003.

[17] Paolo Bouquet, Luciano Serafini, and Stefano Zanobini. Semantic coordination: A new approach and an application. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 130–145, 2003.

[18] Paolo Bouquet, Luciano Serafini, Stefano Zanobini, and Simone Sceffer. Bootstrapping semantics on the web: meaning elicitation from schemas. In *Proceedings of the World Wide Web Conference (WWW)*, pages 505–512, 2006.

[19] Michael Boyd, Sasivimol Kittivoravitkul, Charalambos Lazanitis, Peter McBrien, and Nikos Rizopoulos. AutoMed: A BAV data integration system for heterogeneous data sources. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 82–97, 2004.

[20] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Description logics for information integration. In *Computational Logic:*

Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, pages 41–60. 2002.

[21] Silvana Castano, Valeria De Antonellis, and Sabrina De Capitani di Vimercati. Global viewing of heterogeneous data sources. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):277–297, 2000.

[22] Silvana Castano, Alfio Ferrara, and Stefano Montanelli. Dynamic knowledge discovery in open, distributed and multi-ontology systems: Techniques and applications. In David Taniar and Wenny Rahayu, editors, *Web Semantics and Ontology*. Idea Group Publishing, 2005.

[23] Silvana Castano, Alfio Ferrara, and Stefano Montanelli. Matching ontologies in open networked systems: Techniques and applications. *Journal on Data Semantics (JoDS)*, V:25–63, 2006.

[24] Kevin Chang, Bin He, and Zhen Zhang. Toward large scale integration: Building a metaquerier over databases on the web. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, pages 44–55, 2005.

[25] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *Meeting of the Information Processing Society of Japan*, pages 7–18, 1994.

[26] William Cohen, Pradeep Ravikumar, and Stephen Fienberg. A comparison of string metrics for matching names and records. In *Proceedings of the Workshop on Data Cleaning and Object Consolidation at the International Conference on Knowledge Discovery and Data Mining (KDD)*, 2003.

[27] Nuno Alexandre Pinto da Silva. *Multi-dimensional service-oriented ontology mapping.* PhD thesis, Universidade de Trás-os-Montes e Alto Douro, 2004.

[28] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[29] Hong-Hai Do. *Schema matching and mapping-based data integration.* PhD thesis, University of Leipzig, 2005.

[30] Hong-Hai Do, Sergei Melnik, and Erhard Rahm. Comparison of schema matching evaluations. In *Proceedings of the GI-Workshop Web and Databases, Erfurt (DE)*, 2002.

[31] Hong-Hai Do and Erhard Rahm. COMA – a system for flexible combination of schema matching approaches. In *Proceedings of the International Conference on Very Large Databases*, pages 610–621, 2002.

[32] An-Hai Doan, Pedro Domingos, and Alon Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proceeding of International Conference on Management of Data (SIGMOD)*, pages 509–520, 2001.

[33] An-Hai Doan, Pedro Domingos, and Alon Halevy. Learning to match the schemas of data sources: A multistrategy approach. *Machine Learning*, 50(3):279–301, 2003.

[34] An-Hai Doan and Alon Halevy. Semantic integration research in the database community: A brief survey. *AI Magazine, Special Issue on Semantic Integration*, 26(1):83–94, 2005.

[35] An-Hai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Ontollogy matching: a machine learning approach. In Steffen Staab

and Rudi Studer, editors, *Handbook of ontologies*, International handbooks on information systems, chapter 18, pages 385–404. Springer Verlag, Berlin (DE), 2004.

[36] Pedro Domingos and Michael Pazzani. Beyond independence: Conditions for the optimality of the simple bayesian classifier. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 105–112, 1996.

[37] Marc Ehrig. *Ontology alignment: bridging the semantic gap*. PhD thesis, University of Karlsruhe, 2006.

[38] Marc Ehrig, Staab Staab, and York Sure. Bootstrapping ontology alignment methods with APFEL. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 186–200, 2005.

[39] Marc Ehrig and Steffen Staab. QOM – quick ontology mapping. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 683–697, 2004.

[40] Marc Ehrig and York Sure. Ontology mapping – an integrated approach. In *Proceedings of the European Semantic Web Symposium (ESWS)*, pages 76–91, 2004.

[41] Jérôme Euzenat. Towards composing and benchmarking ontology alignments. In *Proceedings of the Workshop on Semantic Integration at the International Semantic Web Conference (ISWC)*, pages 165–166, 2003.

[42] Jérôme Euzenat. An API for ontology alignment. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 698–712, 2004.

[43] Jérôme Euzenat, Malgorzata Mochol Pavel Shvaiko, Heiner Stuckenschmidt, Ondřej Šváb, Vojtěch Svátek, Willem Robert van Hage, and Mikalai Yatskevich. Results of the ontology alignment evaluation initiative 2006. In *Proceedings of the International Workshop on Ontology Matching (OM) at the International Semantic Web Conference (ISWC)*, 2006.

[44] Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2007. to appear.

[45] Jérôme Euzenat, Heiner Stuckenschmidt, and Mikalai Yatskevich. Introduction to the ontology alignment evaluation 2005. In *Proceedings of the Workshop on Integrating Ontologies at the International Conference on Knowledge Capture (K-CAP)*, 2005.

[46] Jérôme Euzenat and Petko Valtchev. Similarity-based ontology alignment in OWL-lite. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 333–337, 2004.

[47] Ronald Fagin, Phokion Kolaitis, Lucian Popa, and Wang Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 83–94, 2004.

[48] Avigdor Gal. Managing uncertainty in schema matching with top-k schema mappings. *Journal on Data Semantics (JoDS)*, VI:90–114, 2006.

[49] Avigdor Gal, Giovanni Modica, Hassan Jamil, and Ami Eyal. Automatic ontology matching using application semantics. *AI Magazine*, 26(1):21–32, 2005.

[50] Aldo Gangemi, Nicola Guarino, Claudio Masolo, and Alessandro Oltramari. Sweetening WordNet with DOLCE. *AI Magazine*, 24(3):13–24, 2003.

[51] Aldo Gangemi, Nicola Guarino, and Alessandro Oltramari. Conceptual analysis of lexical taxonomies: the case of wordnet top-level. In *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS)*, pages 285–296, 2001.

[52] Michael Garey and David Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W. H. Freeman & Co., 1979.

[53] E. Giunchiglia and R. Sebastiani. Applying the davis-putnam procedure to non-clausal formulas. In *Proceedings of AI*IA*, pages 84–94, 1999.

[54] Fausto Giunchiglia. Contextual reasoning. *Epistemologia, special issue on I Linguaggi ele Macchine*, XVI:345–364, 1993.

[55] Fausto Giunchiglia, Maurizio Marchese, and Ilya Zaihrayeu. Encoding classifications into lightweight ontologies. In *Proceedings of the European Semantic Web Conference (ESWC)*, pages 80–94, 2006.

[56] Fausto Giunchiglia and Pavel Shvaiko. Semantic matching. *The Knowledge Engineering Review*, 18(3):265–280, 2003.

[57] Fausto Giunchiglia and Pavel Shvaiko. Semantic matching. In *Proceedings of the Workshop on Ontologies and Distributed Systems at the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 139–146, 2003.

[58] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. S-Match: an algorithm and an implementation of semantic matching. In *Pro-*

ceedings of the European Semantic Web Symposium (ESWS), pages 61–75, 2004.

[59] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. Semantic schema matching. In *Proceedings of the International Conference on Cooperative Information Systems (CoopIS)*, pages 347–365, 2005.

[60] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. Discovering missing background knowledge in ontology matching. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 382–386, 2006.

[61] Fausto Giunchiglia and Toby Walsh. A theory of abstraction. *Artificial Intelligence*, 57(2-3):323–390, 1992. Also IRST-Technical Report 9001-14, IRST, Trento, Italy.

[62] Fausto Giunchiglia and Mikalai Yatskevich. Element level semantic matching. In *Proceedings of the Meaning Coordination and Negotiation Workshop at the International Semantic Web Conference (ISWC)*, 2004.

[63] Fausto Giunchiglia, Mikalai Yatskevich, and Enrico Giunchiglia. Efficient semantic matching. In *Proceedings of the European Semantic Web Conference (ESWC)*, pages 272–289, 2005.

[64] Fausto Giunchiglia, Mikalai Yatskevich, and Pavel Shvaiko. Semantic matching: algorithms and implementation. *Journal on Data Semantics (JoDS)*, IX, 2006. to appear.

[65] Fausto Giunchiglia and Ilya Zaihrayeu. Making peer databases interact - a vision for an architecture supporting data coordination. In *Proceedings of the International Workshop on Cooperative Information Agents (CIA)*, pages 18–35, 2002.

[66] François Goasdoué, Véronique Lattes, and Marie-Christine Rousset. The use of CARIN language and algorithms for information integration: The PICSEL system. *International Journal of Cooperative Information Systems*, 9(4):383–401, 2000.

[67] Irving John Good. *The estimation of probabilities: an essay on modern Bayesian methods.* Classics Series. The MIT press, 1965.

[68] Nicola 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.

[69] Laura Haas, Mauricio Hernández, Howard Ho, Lucian Popa, and Mary Roth. Clio grows up: from research prototype to industrial tool. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 805–810, 2005.

[70] Bin He and Kevin Chang. Automatic complex schema matching across web query interfaces: A correlation mining approach. *ACM Transactions on Database Systems*, 31(1):1–45, 2006.

[71] G. Hirst and D. St-Onge. Lexical chains as representation of context for the detection and correction malapropisms. In *C. Fellbaum, editor, WordNet: An electronic lexical database and some of its applications. Cambrige, MA: The MIT Press.*, 1997.

[72] Eduard Hovy. Combining and standardizing large-scale, practical ontologies for machine translation and other uses. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, pages 535–542, 1998.

[73] J.J. Jiang and D.W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of the International Conference on Research in Computational Linguistics*, 1998.

[74] Yannis Kalfoglou and Marco Schorlemmer. Ontology mapping: the state of the art. *The Knowledge Engineering Review*, 18(1):1–31, 2003.

[75] Vipul Kashyap and Amit Sheth. Semantic and schematic similarities between database objects: a context-based approach. *The VLDB Journal*, 5(4):276–304, 1996.

[76] David Kensche, Christoph Quix, Mohamed Amine Chatti, and Matthias Jarke. Gerome: A generic role based metamodel for model management. In *Proceedings of the International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE)*, pages 1206–1224, 2005.

[77] Henry Kucera and W Nelson Francis. *Computational analysis of present-day American English*. Providence, Brown University Press, 1967. http://CEUR-WS.org/Vol-128/.

[78] James Larson, Shamkant Navathe, and Ramez Elmasri. A theory of attributed equivalence in databases with application to schema integration. *IEEE Transactions on Software Engineering*, 15(4):449–463, 1989.

[79] Daniel Le Berre. Sat4j: A satisfiability library for Java. http://www.sat4j.org/, 2004.

[80] Claudia Leacock, Martin Chodorow, and George Miller. Using corpus statistics and WordNet relations for sense identification. *Computational Linguistics*, 24(1):1–40, 1998.

[81] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 233–246, 2002.

[82] Nicola Leone, Gianluigi Greco, Giovambattista Ianni, Vincenzino Lio, Giorgio Terracina, Thomas Eiter, Wolfgang Faber, Michael Fink, Georg Gottlob, Riccardo Rosati, Domenico Lembo, Maurizio Lenzerini, Marco Ruzzi, Edyta Kalka, Bartosz Nowicki, and Witold Staniszkis. The INFOMIX system for advanced integration of incomplete and inconsistent data. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 915–917, 2005.

[83] Wen-Syan Li and Chris Clifton. Semantic integration in heterogeneous databases using neural networks. In *Proceedings of the Very Large Data Bases Conference (VLDB)*, pages 1–12, 1994.

[84] Wen-Syan Li and Chris Clifton. SEMINT: a tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data and Knowledge Engineering*, 33(1):49–84, 2000.

[85] Dekang Lin. An information-theoretic definition of similarity. In *Proceedings of the International Conference of machine learning (ICML)*, pages 296–304, 1998.

[86] Alexander Mädche, Boris Motik, Nuno Silva, and Raphael Volz. MAFRA – a mapping framework for distributed ontologies. In *Proceedings of the International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, pages 235–250, 2002.

[87] Jayant Madhavan, Philip Bernstein, An-Hai Doan, and Alon Halevy. Corpus-based schema matching. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 57–68, 2005.

[88] Jayant Madhavan, Philip Bernstein, and Erhard Rahm. Generic schema matching using Cupid. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 48–58, 2001.

[89] B. Magnini, Luciano 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 the International Semantic Web Conference (ISWC)*, 2003.

[90] Andrew McCallum and Kamal Nigam. A comparison of event models for naive bayes text classification. In *Proceedings of the Workshop on Learning for Text Categorization at the National Conference on Artificial Intelligence (AAAI)*, 1998.

[91] Deborah McGuinness, Richard Fikes, James Rice, and Steve Wilder. An environment for merging and testing large ontologies. In *Proceedings of the International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, pages 483–493, 2000.

[92] Open information model, version 1.0. http://mdcinfo/oim/oim10.html, 1999.

[93] Sergey Melnik, Philip Bernstein, Alon Halevy, and Erhard Rahm. Supporting executable mappings in model management. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 167–178, 2005.

[94] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: a versatile graph matching algorithm. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 117–128, 2002.

[95] Sergey Melnik, Erhard Rahm, and Philip Bernstein. Developing metadata-intensive applications with Rondo. *Journal of Web Semantics*, 1(1):47–74, 2003.

[96] Sergey Melnik, Erhard Rahm, and Philip Bernstein. Rondo: A programming platform for model management. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 193–204, 2003.

[97] Eduardo Mena, Vipul Kashyap, Amit Sheth, and Arantza Illarramendi. Observer: An approach for query processing in global information systems based on interoperability between pre-existing ontologies. In *Proceedings of the International Conference on Cooperative Information Systems (CoopIS)*, pages 14–25, 1996.

[98] George Miller. WordNet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[99] Renée Miller, Laura Haas, and Mauricio Hernández. Schema mapping as query discovery. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, pages 77–88, 2000.

[100] Renée Miller, Mauricio Hernández, Laura Haas, Lingling Yan, Howard Ho, Ronald Fagin, and Lucian Popa. The Clio project: managing heterogeneity. *SIGMOD Record*, 30(1):78–83, 2001.

[101] Tova Milo and Sagit Zohar. Using schema matching to simplify heterogeneous data translation. In *Proceedings of the Very Large Databases Conference (VLDB)*, pages 122–133, 1998.

[102] Prasenjit Mitra and Gio Wiederhold. Resolving terminological heterogeneity in ontologies. In *Proceedings of the Workshop on Ontolo-*

gies and Semantic Interoperability at the European Conference on Artificial Intelligence (ECAI), 2002.

[103] Prasenjit Mitra, Gio Wiederhold, and Jan Jannink. Semi-automatic integration of knowledge sources. In *Proceedings of the International Conference On Information Fusion (FUSION)*, 1999.

[104] Prasenjit Mitra, Gio Wiederhold, and Martin Kersten. A graph-oriented model for articulation of ontology interdependencies. In *Proccedings of the International Conference on Extending Database Technology (EDBT)*, pages 86–100, 2000.

[105] Giovanni Modica, Avigdor Gal, and Hasan Jamil. The use of machine-generated ontologies in dynamic information seeking. In *Proceedings of the International Conference on Cooperative Information Systems (CoopIS)*, pages 433–448, 2001.

[106] Felix Naumann, Ching-Tien Ho, Xuqing Tian, Laura Haas, and Nimrod Megiddo. Attribute classification using feature analysis. In *Proceedings of the International Conference on Data Engineering (ICDE)*, page 271, 2002.

[107] Ian Niles and Adam Pease. Towards a standard upper ontology. In *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS)*, pages 2–9, 2001.

[108] Henrik Nottelmann and Umberto Straccia. sPLMap: A probabilistic approach to schema matching. In *Proceedings of the European Conference on Information Retrieval Research (ECIR)*, pages 81–95, 2005.

[109] Natalya Noy. Semantic integration: A survey of ontology-based approaches. *SIGMOD Record*, 33(4):65–70, 2004.

[110] Natalya Noy. Tools for mapping and merging ontologies. In Steffen Staab and Rudi Studer, editors, *Handbook of ontologies*, International handbooks on information systems, chapter 18, pages 365–384. Springer Verlag, Berlin (DE), 2004.

[111] Natalya Noy and Marc Musen. SMART: Automated support for ontology merging and alignment. In *Proceedings of the Workshop on Knowledge Acquisition, Modeling, and Management*, 1999.

[112] Natalya Noy and Marc Musen. The PROMPT suite: interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies (IJHCS)*, 59(6):983–1024, 2003.

[113] Natalya Noy and Mark Musen. Anchor-PROMPT: Using non-local context for semantic matching. In *Proceedings of the Workshop on Ontology and Information Sharing at the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 63–70, 2001.

[114] Luigi Palopoli, Luigi Pontieri, Giorgio Terracina, and Domenico Ursino. Intensional and extensional integration and abstraction of heterogeneous databases. *Data Knowledge Engineering*, 35(3):201–237, 2000.

[115] Luigi Palopoli, Domenico Saccá, Giorgio Terracina, and Domenico Ursino. Uniform techniques for deriving similarities of objects and subschemes in heterogeneous databases. *IEEE Transactions on Knowledge and Data Engineering*, 15(2):271–294, 2003.

[116] Luigi Palopoli, Domenico Saccà, and Domenico Ursino. An automatic techniques for detecting type conflicts in database schemes. In *Proceedings of the Conference on Information and Knowledge Management (CIKM)*, pages 306–313, 1998.

[117] Luigi Palopoli, Giorgio Terracina, and Domenico Ursino. DIKE: a system supporting the semi-automatic construction of cooperative information systems from heterogeneous databases. *Software–practice and experinece*, 33(9):847–884, 2003.

[118] Christine Parent and Stefano Spaccapietra. Issues and approaches of database integration. *Communications of the ACM*, 41(5):166–178, 1998.

[119] Christine Parent and Stefano Spaccapietra. Database integration: the key to data interoperability. In *Object-Oriented Data Modeling*. The MIT Press, Cambridge (MA US), 2000.

[120] S. Patwardhan, S. Banerjee, and T. Pedersen. Using measures of semantic relatedness for word sense disambiguation. 2003.

[121] Siddharth Patwardhan and Ted Pedersen. Using wordnet based context vectors to estimate the semantic relatedness of concepts. In *Proceedings of the EACL 2006 Workshop Making Sense of Sense - Bringing Computational Linguistics and Psycholinguistics Together*, pages 1–8, 2006.

[122] D. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[123] Alun Preece, Kit-Ying Hui, Alex Gray, Philippe Marti, Trevor Bench-Capon, Dean Jones, and Zhan Cui. The KRAFT architecture for knowledge fusion and transformation. *Knowledge-Based Systems*, 13(2-3):113–120, 2000.

[124] Erhard Rahm and Philip Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

[125] Erhard Rahm, Hong-Hai Do, and Sabine Maßmann. Matching large XML schemas. *SIGMOD Record*, 33(4):26–31, 2004.

[126] Phillip Resnik. Semantic similarity in a taxonomy: an information-based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence Research (JAIR)*, 11:95–130, 1999.

[127] Mayssam Sayyadian, Yoonkyong Lee, An-Hai Doan, and Arnon Rosenthal. Tuning schema matching software using synthetic scenarios. In *Proceedings of the Very Large Data Bases Conference (VLDB)*, pages 994–1005, 2005.

[128] Hinrich Schütze. Dimensions of meaning. In *SC*, pages 787–796, 1992.

[129] Pavel Shvaiko and Jérôme Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics (JoDS)*, IV:146–171, 2005.

[130] Stefano Spaccapietra and Christine Parent. Conflicts and correspondence assertions in interoperable databases. *SIGMOD Record*, 20(4):49–54, 1991.

[131] Umberto Straccia and Raphaël Troncy. oMAP: Combining classifiers for aligning automatically owl ontologies. In *Proceedings of the International Conference on Web Information Systems Engineering (WISE)*, pages 133–147, 2005.

[132] K. Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.

[133] Kai Ming Ting and Ian Witten. Issues in stacked generalization. *Journal of Artificial Intelligence Research (JAIR)*, 10:271–289, 1999.

[134] G. Tsetin. On the complexity proofs in propositional logics. Technical report, Seminars in Mathematics, 8, 1960.

[135] Holger Wache, Thomas Voegele, Ubbo Visser, Heiner Stuckenschmidt, Gerhard Schuster, Holger Neumann, and Sebastian Hübner. Ontology-based integration of information - a survey of existing approaches. In *Proceedings of the Workshop on Ontologies and Information Sharing at the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 108–117, 2001.

[136] Antoine Zimmermann, Markus Krötzsch, Jérôme Euzenat, and Pascal Hitzler. Formalizing ontology alignment and its operations with category theory. In *Proceedings of the International Conference on formal ontologies for information systems (FOIS)*, 2006.

[137] Sagit Zohar. Schema-based data translation. Master's thesis, Tel-Aviv University (IL), 1997.