SPIMBench: A Scalable, Schema-Aware Instance Matching Benchmark for the Semantic Publishing Domain

Tzanina Saveta

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes Campus, Heraklion, GR-70013, Greece

Thesis Advisor: Prof. *Dimitris Plexousakis*

University of Crete
Computer Science Department

SPIMBench: A Scalable, Schema-Aware Instance Matching Benchmark for the Semantic Publishing Domain

Thesis submitted by
Tzanina Saveta
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author:	Tzanina Saveta
Committee approvals:	Dimitris Plexousakis Professor, Thesis Supervisor
	Yannis Tzitzikas Assistant Professor, Committee Member
	Irini Fundulaki Principal Researcher, Committee Member
Departmental approval:	Antonis A. Argyros Professor, Director of Graduate Studies

Heraklion, October 2014

Abstract

Instance matching systems and methods need to be tested using well defined and widely accepted benchmarks to determine the weak and strong points thereof and also to motivate the development of more complete systems. A benchmark should test the overall quality of the instance matching system in terms of measures such as precision, recall, and F-measure as well as the ability to handle large and diverse datasets.

A number of benchmarks have already been proposed to test the performance of instance matching techniques mostly for XML and relational data but, more recently, also for RDF, the type of data prevalent in the Web of Data. Instance Matching benchmarks for RDF data are the first to consider the problem of instance matching when a real world object is represented in different ways that do not all conform to the same RDFS or OWL schema. Meaning that in addition to lexical differences among entities representing the same object, these benchmarks consider *structural* differences such as property splitting or aggregation. However, to the best of our knowledge, none of the proposed benchmarks to date considers the more complex *logical* constructs that can be expressed in terms of rich OWL constructs. The logical transformations proposed by existing benchmarks all remain at the level of simple RDFS constraints.

In this thesis we propose the *Semantic Publishing Instance Matching Benchmark*, in short, SPIMBench inspired from the Semantic Publishing domain. SPIMBench is based on the BBC (`http://www.bbc.com/`) ontologies that represent information about *creative works* (called *journalistic assets*) created by the publisher's editorial team. SPIMBench proposes and implements *i)* a *scalable data generator*, *ii)* a set of *transformations* on source data to obtain the target data that include, in addition to the standard value and structural transformations, *logical* ones that go beyond the standard RDFS constructs and include expressive OWL constructs, namely *instance (in)equality*, *equivalence* of classes and properties, *property constraints* and *complex class definitions*, a *iii) weighted gold standard* that can be used for debugging instance matching systems and finally, *iv)* a set of *metrics* used to assess the performance of an instance matching system.

Περίληψη

Τα τελευταία χρόνια, η αύξηση των διαθέσιμων *Συνδεδεμένων Δεδομένων* (*Linked Data*) στον Παγκόσμιο ιστό έχει αποτελέσει τον θεμέλιο λίθο στην ανάπτυξη *Συστημάτων Αντιστοίχισης Στιγμιοτύπων (Instance Matching Systems)*. Όπως για τα συστήματα Βάσεων Δεδομένων, έτσι και εδώ, *Πλαίσια Αξιολόγησης Συστημάτων Αντιστοίχισης Στιγμιοτύπων* (*Instance Matching Benchmarks)* έχουν αναπτυχθεί για τον έλεγχο απόδοσης των προαναφερθέντων συστημάτων με βασικό σκοπό τον προσδιορισμό των μειονεκτημάτων τους για την περαιτέρω βελτίωση των λειτουργιών τους.

Ένα πλαίσιο αξιολόγησης συστημάτων ταυτοποίησης στιγμιοτύπων θα πρέπει να ελέγχει τη συνολική ποιότητα του συστήματος αντιστοίχισης στιγμιοτύπων με μετρικές όπως η *ακρίβεια* (*precision*), η *ανάκληση* (*recall*), και το *F-measure* καθώς και την ικανότητα να χειρίζεται σύνολα δεδομένων *μεγάλου όγκου*.

Πλαίσια αξιολόγησης έχουν ήδη προταθεί για τον έλεγχο της απόδοσης συστημάτων αντιστοίχισης στιγμιοτύπων για δεδομένα XML και δεδομένα σχεσιακών βάσεων και πρόσφατα για τα δεδομένα RDF τα οποία έχουν αρχίσει να επικρατούν στον Παγκόσμιο Ιστό. Τα συστήματα αξιολόγησης που λαμβάνουν υπ΄ όψιν δεδομένα εκφρασμένα σε RDF είναι τα πρώτα τα οποία εξέτασαν το πρόβλημα της αντιστοίχισης στιγμιοτύπων όταν ένα αντικείμενο του πραγματικού κόσμου έχει διαφορετικές περιγραφές που χρησιμοποιούν τα ίδια ή διαφορετικά RDFS (ή τα εκφραστικότερα OWL) σχήματα.

Αυτό σημαίνει πως εκτός από τις *λεξικολογικές* διαφορές μεταξύ των στιγμιοτύπων που περιγράφουν την ίδια οντότητα του πραγματικού κόσμου, τα πλαίσια αξιολόγησης λαμβάνουν υπ΄ όψιν διαφορές σε επίπεδο *σχήματος* όπως τη διάσπαση ή τη συνάθροιση μίας ιδιότητας ενός στιγμιότυπου. Ωστόσο, σύμφωνα με τη βιβλιογραφία, κανένα από τα προτεινόμενα πλαίσια αξιολόγησης μέχρι σήμερα δεν λαμβάνει υπ΄ όψιν τις πιο πολύπλοκες δομές σε επίπεδο *σχήματος* τα οποία μπορούν να εκφραστούν, χρησιμοποιώντας τα πλούσια δομικά στοιχεία της γλώσσας του Σημασιολογικού Ιστού OWL. Οι μετασχηματισμοί που έχουν προταθεί παραμένουν όλοι στο επίπεδο των απλών δομών όπως εκείνες περιγράφονται στην γλώσσα RDFS .

Στην παρούσα εργασία προτείνουμε το *Semantic Publishing Instance Matching*

Benchmark, εν συντομία SPIMBench , ένα πλαίσιο αξιολόγησης εμπνευσμένο από το *Semantic Publishing Benchmark* SPB. To SPIMBench , όπως το SPB , είναι βασισμένο στις οντολογίες όπως έχουν δοθεί από το BBC (`http://www.bbc.com/`) οι οποίες χρησιμοποιήθηκαν απο τον συγκεκριμένο δημοσιογραφικό οργανισμό για την δημοσίευση Σημασιολογικά Εμπλουτισμένων Δεδομένων. Στο SPIMBench προτείνουμε και υλοποιούμε μία *α) επεκτάσιμη γεννήτρια δεδομένων*, *β)* ένα σύνολο *μετασχηματισμών* πού αποτελούνται από τους καθιερωμένους *λεξικολογικούς*, *δομικούς* και μετασχηματισμούς σε επίπεδο *λογικού σχήματος*. Οι τελευταίοι μετασχηματισμοί υπερβαίνουν τα καθιερωμένα δομικά στοιχεία και περιλαμβάνουν εκφραστικά δομικά στοιχεία όπως *ισότητα/ανισότητα στιγμιοτύπων*, *ισοδυναμία* των κλάσεων και των ιδιοτήτων σε επίπεδο σχήματος, *περιορισμό ιδιοτήτων*, *περίπλοκους ορισμούς κλάσεων*, και τέλος *γ)* έναν *σταθμισμένο χρυσό κανόνα* ο οποίος μπορεί να χρησιμοποιηθεί για τον εντοπισμό σφαλμάτων στα συστήματα αντιστοίχισης στιγμιοτύπων.

Acknowledgements

I wouldn't have made it this far without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here.

Above all, I would like to thank my parents Agim Saveta and Sofia Saveta for their personal support and great patience at all times. My sister Viola Saveta who is there for me at any time, as always, for which my mere expression of thanks likewise does not suffice.

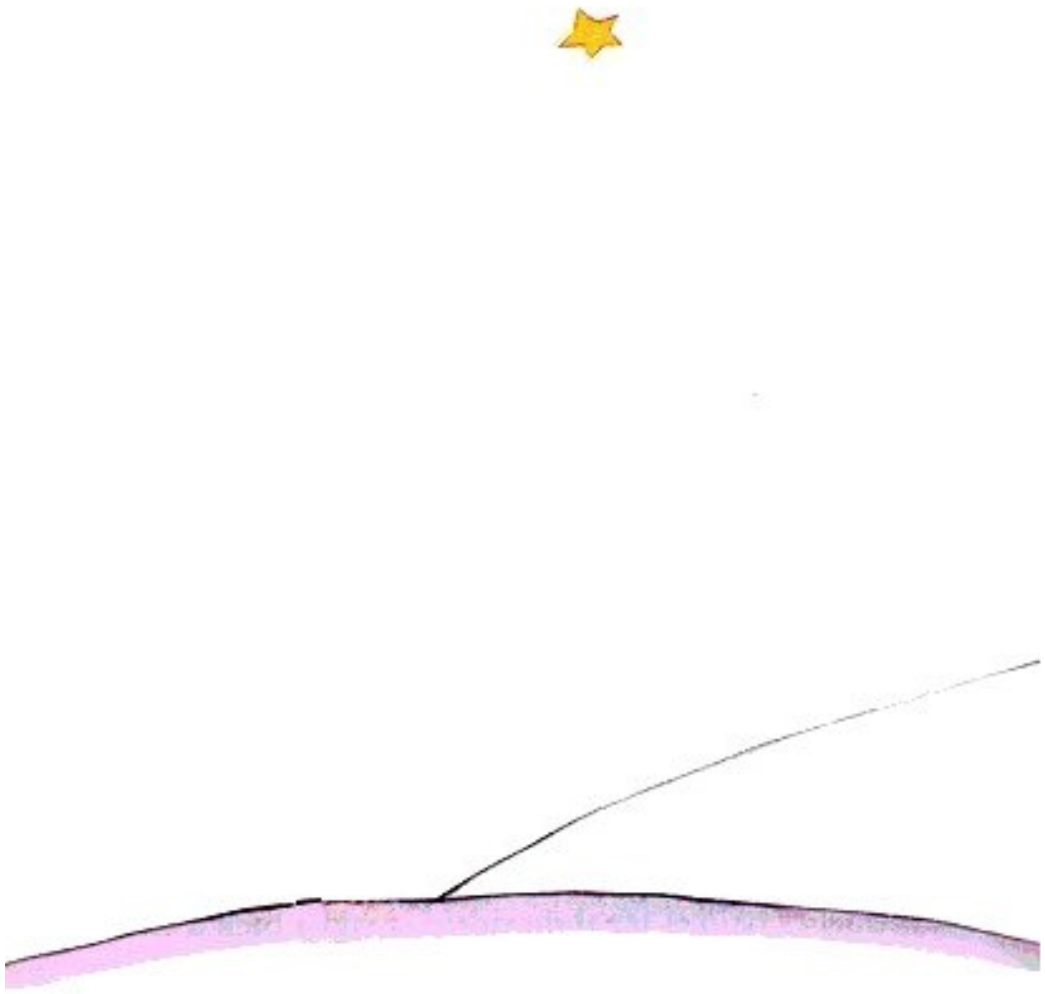
This thesis would not have been possible without the help, support and patience of my supervisors, Professor of the Computer Science Department Dimitris Plexousakis and Principal Researcher of the Institute of Computer Science Irini Fundulaki. They have been invaluable on both an academic and a personal level, for which I am extremely grateful.

I would also like to thank Professor Melanie Herschel from IPVS - University of Stuttgart, moreover, Researcher Giorgos Flouris and Evangelia Daskalaki from Institute of Computer Science (ICS) - FORTH whose their guidance helped me in all the time of research of this thesis. Special thanks are given to Axel-Cyrille Ngonga Ngomo from the University of Leipzig for the collaboration we had, his advices and the unsurpassed knowledge of Semantic Web and Machine Learning.

I particularly want to thank all my friends whose their care helped me overcome setbacks and stay focused on my study. I greatly value their friendship and I deeply appreciate their belief in me.

Next I would like to thank the Institute of Computer Science (ICS), Foundation of Research and Technology Hellas (Forth) and more specifically the members of the Information Systems laboratory (ISL) for the support, the goodwill to help and the great time we had during my graduate studies.

Finally, I appreciate the financial support from FP7 European Project LDBC (Linked Data Benchmark Council) that funded parts of the research discussed in this thesis.



Contents

List of Figures

List of Tables

Chapter 1

Introduction

Instance matching, also known under the names of entity resolution [1], duplicate detection [2], record linkage [3], object identification in the context of databases [4] and many others in the literature, refers to the problem of identifying instances that describe the *same real world object*. The problem has been studied for many decades in the relational data setting [2]. With the increasing adoption of Semantic Web Technologies and the publication of large interrelated RDF datasets and ontologies that form the Linked Data Cloud[1], data integration problems such as entity resolution become more crucial than in the relational data setting; in this new open environment, there is a high degree of heterogeneity both at the schema and instance level in addition to the rich semantics that accompany the former expressed in terms of expressive languages such as OWL [5] and RDFS [6]. In this context where *scale*, and *heterogeneity* are crucial parameters of the problem, new instance matching techniques have been proposed [7, 8].

Instance matching systems and methods need to be tested using *well defined and widely accepted benchmarks* to determine the weak and strong points of a method or system and also to motivate the development of more complete systems. An instance matching benchmark comprises:

- *benchmark dataset(s)* associated with a domain of interest so as to have meaningful and interpretable results.

[1]`http://linkeddata.org/`

- *a gold standard* used to judge the completeness and soundness of the instance matching approach.
- a set of *test cases*, each addressing a different kind of requirements such as *lexical* (or *value*), *structural*, and *logical* modifications that an instance matching system should support.
- a set of metrics to assess the overall performance of the instance matching system.

A benchmark should test the overall quality of the instance matching system in terms of *precision*, *recall*, and *F-measure* as well as the ability to handle large and diverse datasets (*efficiency*, *scalability* and *diversity* dimensions). A number of benchmarks have already been proposed to test the performance of instance matching techniques mostly for XML and relational data [9] but, more recently, also for RDF data, the type of data prevalent in the Web of Data [10, 11, 12, 13, 14, 15, 16, 17].

The benchmarks considering data expressed in terms of RDF are the first to consider the problem of instance matching when a real world object is represented in different ways that do not all conform to the same RDFS or OWL schema. Meaning that in addition to lexical differences among entities representing the same object, these benchmarks consider *structural* differences such as property splitting or aggregation. However, to the best of our knowledge, none of the proposed benchmarks to date considers the more complex *logical* constraints that can be expressed, in terms of rich OWL constructs. The logical transformations proposed by existing benchmarks all remain at the level of simple RDFS constraints such as *subclass-of* or the OWL *same-as* constraint. Such constraints, when present, can be used by instance matching techniques to potentially obtain better results, however, none of the state of the art benchmarks employ those.

In this thesis we propose the *Semantic Publishing Instance Matching Benchmark*, in short, SPIMBench, a benchmark inspired from the *Semantic Publishing Benchmark* SPB. SPIMBench, like SPB, is based on the BBC (`http://www.bbc.com/`) ontologies, which lie in the *Semantic Publishing* domain.

SPIMBench proposes and implements

- a *scalable data generator* that produces synthetic *source* and *target* data consistent with the extended SPIMBench schema to be used for testing the performance of instance matching systems.
- a set of *transformations* on source data to obtain the target data. The set of transformations supported by SPIMBench includes *value* and *structural* ones as those have been proposed in a large number of representative instance matching benchmarks; and finally the *logical* ones that go beyond the standard RDFS constructs and include expressive OWL constructs, namely *instance (in)equality*, *equivalence* of classes and properties, *property constraints* and *complex class definitions.*
- a *detailed weighted gold standard* that records for each pair of (source, target) instances an entry that stores (a) the type of transformation applied, (b) the property on which it is applied and (c) the weight for the individual transformation. The detailed gold standard can be used for debugging instance matching systems since we explicitly store the transformations applied to a source to obtain a target instance as well as their degree of similarity.
- a *weighted gold standard* that records for each pair of (source, target) instances the weight of the distance between them.

Structure. In Chapter 2, we discuss related work, and in Chapter 3 we cover basic concepts and definitions that we use throughout this thesis. Chapter 4 discusses the SPIMBench schema (Section 4.1), the employed benchmark metrics (Section 4.2), transformations (Section 4.3), the data generator (Section 4.4) and finally the gold standard (Section 4.5). Last, in Section 4.6 we provide scalability experiments that show that the SPIMBench transformations do not introduce any additional overhead. Conclusions are provided in Chapter 5.

Chapter 2

Related Work

A number of benchmarks have been developed to test the performance of instance matching systems mostly for XML and relational data [9, 18, 19]. These benchmarks do not take into account the features inherent in the, rich in semantics, ontologies of the Linked Data Cloud expressed in the Ontology Web Language (OWL) [5] and the less expressive RDF Schema Vocabulary(RDFS) [20].

An instance matching benchmark comprises of *benchmark dataset(s)* associated with a domain of interest in order to be able to have meaningful interpretable results. Benchmarks are distinguished to *real* or *synthetic*: the former consider existing datasets, whereas the latter produce datasets that are mostly used for stressing the capability of the systems to discover interesting matches. In the case of real benchmarks, this can be achieved by selecting the appropriate datasets, a difficult task. Hence, a lot of benchmarks include data from different domains, thereby producing datasets that are not intuitive. Benchmarks come usually with a *ground truth* or *gold standard* used to judge the completeness and soundness of the instance matching approach; gold standards are provided either in the form of *pairs of matched instances* or *matching links* that identify *similar* instances (i.e., instances that refer to the same real world entity). Furthermore, benchmarks also come with the standard metrics of *precision*, *recall* and *f-measure*. A benchmark comes also with a a set of *test cases*, each addressing a different kind of data heterogeneities that instance matching algorithms should test. Usually, *synthetic* *benchmarks* propose test cases in order to provide a more systematic way for testing the matching systems' performance. These are built on *transformations* such as *value*, *structural* and *logical* modifications or combinations thereof. Depending on the complexity of the modifications, these can be *simple* or *complex* ones, the latter mostly referring to *structural* and *logical* heterogeneities [21].

In particular, [21] considered the following types of variations:

- *value differences:* these include misspellings of the names at both the schema and instance level (e.g., typographical errors), as well as the use of different formats to represent the same kind of information.
- *structural heterogeneities:* these consider changes mostly at the schema level, such as different nesting levels for properties, class and property hierarchies, the use of different aggregation criteria for the representation of properties etc.
- *logical heterogeneities:* these support instantiation of instances to classes that belong to the same or different explicitly or implicitly defined hierarchies.

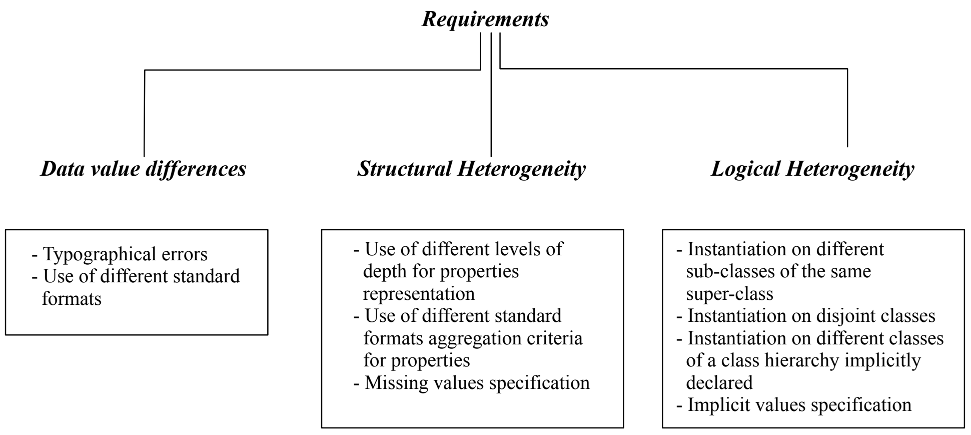

Requirements
Data value differences
Structural Heterogeneity
Logical Heterogeneity
- Typographical errors
- Use of different standard formats
- Use of different levels of depth for properties representation
- Use of different standard formats aggregation criteria for properties
- Missing values specification
- Instantiation on different sub-classes of the same super-class
- Instantiation on disjoint classes
- Instantiation on different classes of a class hierarchy implicitly declared
- Implicit values specification

Figure 2.1: Modifications for Instance Matching

2.1 Benchmarks

2.1.1 Ontology Alignment Evaluation Initiative(OAEI)

The most popular framework for testing ontology matching systems is the one published by the Ontology Alignment Evaluation Initiative (OAEI) [22]. Since 2005, OAEI organizes an annual campaign aiming at evaluating *ontology matching* solutions and technologies using a *fixed set of benchmarks*. In 2009, OAEI introduced the *Instance Matching (IM) Track*, which focuses on the evaluation of different instance matching techniques and tools for RDF data. The track proposed two different benchmarks to this end: *ARS* and *IIMB* benchmarks [10] (the TSD benchmark was also proposed in 2009 in the instance matching track, but was canceled). The metrics used to measure the effectiveness of the instance matching tools for each of the benchmarks were the standard metrics of *precision*, *recall* and *f-measure*. The ARS benchmark considers *real datasets* with a relatively *small* number of instances (in the order of thousands), obtained from three different sources from the domain of scientific publications. The instance matching systems tested discover matches between the instances of the aforementioned datasets.

In addition to these real datasets, the OAEI 2009 IM track proposed the *synthetic* ISLab Instance Matching Benchmark [11] (IIMB). The benchmark considers a *single source dataset* from the OKKAM project[1], along with an OWL reference ontology, to which a set of *transformations* are applied in order to obtain a *target dataset*. These transformations are organized in 37 different test scenarios, each of which is comprised of the reference ontology (schema and instances), the modified ontology that is obtained by applying a set of simple and complex *value*, *structural* and *simple logical* modifications mostly for *class hierarchies*, as well as combinations of the above. As in the case of ARS, the dataset contains a very small number of instances (around 2000), and hence cannot be used to test the ability of the instance matching systems to scale.

The OAEI 2010 Instance Matching track [12] included two new classes of tests, namely the *Data Interlinking (DI)*, and the *OWL Data* test; the first was devel-

[1]OKKAM Project: `http://www.okkam.org/`

oped to test the ability of the systems to *interlink* resources in the Linked Data Cloud. Following the paradigm of ARS, the former uses *real datasets* expressed in RDF that contain information on drugs and their adverse effects, diseases, cheminformatics; it also considers LinkedMDB[2] dataset that contains movie information. The purpose of this benchmark was to test the ability of the systems to find matches between instances originating from *different domains*, with no *a priori* knowledge of the application domain, the datasets or the respective schemas. The DI benchmark was designed to test whether the instance matching systems *scale* for large datasets, since the proposed datasets were as large as DBpedia (containing hundreds of thousands of instances). For each pair of tests, a gold standard was provided to test the effectiveness of the employed instance matching tool and technique. The gold standard was provided in the form of a reference alignment dataset (i.e., a set of links between the reference and the target ontologies) where links are manually created.

The OAEI 2010 OWL Data Benchmark, considered the dataset from the IIMB benchmark (discussed above), as well as a small dataset that contains data for persons and restaurants; the latter was considered in order to increase the diversity of the benchmark data. Considering that part of the data were synthetic, this benchmark sets the basis for evaluating the ability of the instance matching systems to detect different kinds of *transformations*. The task focused on two main goals, namely to provide an evaluation dataset for various kinds of *transformations*, and to cover a wide spectrum of possible techniques and tools. The difference between this benchmark and the aforementioned ones is that the tested instance matching system should consider a certain form of *(simple) reasoning* in order to link the resources.

The OAEI 2011 Instance Matching Track [13], a follow-up of OAEI 2010 IM Track, proposed two benchmarks: the first was based on the IIMB benchmark mentioned earlier, and the latter on a set of *real datasets* from the New York Times (NYT), DBpedia, FreeBase and GeoNames provided along with a number of OWL `owl:sameAs` links. The datasets for the first benchmark were produced using

[2]LinkedMDB: `http://datahub.io/dataset/linkedmdb`

the SWING [23] data generator applied on the FreeBase dataset; the latter was developed to test the interlinking of the instances in the aforementioned datasets. The purpose of the second task was to re-build the links within the NYT dataset as well as to discover additional links to the ones that were already provided along with the datasets.The gold standard (expressed in terms of links between resources) was extracted from the links provided along with the NYTimes dataset and curated by NY Times journalists and curators.

Following the OAEI 2011 IM, the OAEI 2012 IM Track included benchmarks proposed by OAEI 2011 Track, along with the Sandbox dataset that was added to provide examples of some specific matching problems like name spelling and other controlled variations for strings. The reason for adding this new dataset was to test the instance matching tools that are in an initial phase of their development process (providing a kind of a micro-benchmark). In 2013, OAEI proposed RDFT [14], an automatically generated RDF benchmark that includes controlled distortions into the source RDF data. Those transformations are *value*, *structural* and *translations* for a certain type of data (comments and labels). The relatively small source dataset (in the order of few hundreds of triples) is a subset of DBpedia about computer scientists. The alignments were provided for the training dataset but not for the whole evaluation set, hence the evaluation is blind. In the latest OAEI 2014 Track [3] two benchmarks were proposed: one for identity recognition and one for similarity recognition. The datasets for the Identity Recognition sub-task have been produced by automatically modifying a set of original data (expressed as OWL ABOXes) in order to obtain different versions of the same description where different languages and representation formats are employed. The datasets for the similarity recognition tasks were obtained through a crowdsourcing process.

OAEI IM bencnhmarks have been extensively used to test the performance of instance matching systems for a number of diverse domains using both real and synthetic datasets in order to test all specific aspects of instance matching for Linked Data. Nevertheless, not all proposed benchmarks tackle important aspects of the instance matching problem. More specifically, the majority of the bench-

[3]OAEI 2014 IM Track: `http://islab.di.unimi.it/im_oaei_2014/index.html`

	Classes		
Statistics	*Person*	*Organization*	*Locations*
Total #sameAs links	14884	8003	8786
Links to Freebase	4979	3044	1920
Links to DBpedia	4977	1949	1920
Links to NYTimes	4979	3044	1920
Links to GeoNames	0	0	1789

Table 2.1: Links from the NYTimes dataset to FreeBase, DBpedia and GeoNames

marks introduced in 2009 and 2010 consider only a small number of triples, except the Data Interlinking benchmark where the very large number of instances leads to an error-prone gold standard since it is more or less impossible to construct manually a correct alignment. OAEI 2011, 2012 benchmarks consider larger datasets and offer a precise and error prone gold standard. In addition, the transformations considered in the synthetic benchmarks do not consider a combinations of simple modifications or even modifications that take into consideration schema information. Finally, the OAEI benchmarks employ the *standard metrics of precision and recall* to measure the performance and quality of the instance matching techniques against a predefined reference alignment (the gold standard).

2.1.2 ONTOlogy Matching Benchmark With Many Instances (ONTOBI)

ONTOlogy Matching Benchmark With Many Instances (ONTOBI) [15, 16] is an instance matching benchmark that uses the DBpedia ontology (version 3.4). The benchmark proposes 16 different test cases that take into account *simple* and *complex transformations* that are applied on the reference ontology; simple modifications are actually *value transformations* that include misspellings, insertion/deletion of comments attached to classes, the use of different data formats for both data and schema, the removal of data types in class attributes whereas complex modifications refer to structural ones (e.g., schema expansion, use of different languages, random names, synonyms). Changes of class comments is an important change

Simple Modifications	*Complex Modifications*
Misspellings	Expanded/flattened Schema
Insertion/Deletion of Comments	Use of Different Language
Different Data formats	Use of Random names
No use of data types	Use of Synonyms
Overlapping Datasets	Disjoint Data Sets

Table 2.2: Simple and Complex Modifications in ONTOBI

since they describe the semantics of the class, so this information can be used to detect homonyms or synonyms and consequently help in the process of schema and instance matching. Another simple modification at the schema level is the removal of information related to data types; such information provides additional hints on the semantics of the instance values. In addition ONTOBI uses either *overlapping datasets*, in that case the task is simple in the sense that an instance matching system should find at least the instances that are common in both datasets or *disjoint datasets* in which case, the instance matcher should use intelligent techniques to discover the possible matches. Target datasets used in those test cases are created by applying the aforementioned modifications on a small set of DBpedia instances. Each of the test cases is also accompanied by a reference alignment.

2.1.3 STBenchmark

STBenchmark [17] is a benchmark that takes as input one reference ontology, and applies several transformations in order to get a modified reference ontology. STBenchmark supports a basic set of scenarios that represent the minimum set of transformations which should be supported by any matching system for both data and schema. In addition, it contains a generator for instances and matching scenarios that can be used to produce more complex ones namely *instance copying* with the same or *randomly generated* identifiers, *constant value generation*, *assignment* of instances of a class to different classes (one of the logical transformations that SPIMBench supports), *structural* transformations such as *unnesting* (flattening) or *nesting* of schema properties. STBenchmark has a *matching scenario* generator *SGen* that takes as input parameters related to the characteristics of the reference ontology (*schema level only*), and produces a matching scenario. Given that transformations are also applicable to schema, SGen can be used as the target schema generator; STBenchmark employs the instance generator *IGen* that uses the template-based XML data generator ToxGene [24]. The latter takes as input a schema and a set of configuration parameters and returns the set of instances that conform to the input schema. These instances are then used as input SGen along with a matching scenario to produce the target instances.

STBenchmark shows an interesting systematic way of creating different kinds of testbeds for instance matching. STBenchmark employs artificial, randomly created instances with meaningless content, rather than real-world data, which are not that useful for testing all aspects of matching systems, because artificial data follow a more or less strict pattern and make it difficult to completely simulate the data obtained through manual curation. Another disadvantage of the STBenchmark is that no reference alignment is generated, as with ONTOBI and OAEI benchmarks, so it cannot be disseminated easily and used by many instance matching systems.

2.1.4 Discussion on Instance Matching Benchmarks

The benchmarks presented above are constructed to test matching systems, especially with respect to instance-based matchers. We reviewed them and pointed out their strengths and weaknesses. We summarize here the characteristics of the benchmarks following the aspects introduced in [25]. These are:

- *systematic procedure* according to which the matching tasks are reproducible and the execution has to be comparable

- *continuity*, related to the continuous improvement of the matching tasks improved

- *quality* and *equity*. The first requires that the evaluation rules are exact and the quality of the ontologies is high. The second ensures that no system should be privileged during the evaluation process.

- *dissemination* meaning that the benchmark and the results should be publicly available and other systems have reported them in their evaluations.

	OAEI ARS/ TDS/ VLCR/	**OAEI IIMB**	**OAEI DI**	**STBenchmark**	**ONTOBI**	**OAEI NYT/ FreeBase**
systematic procedure	+	+	+	+	+	+
continuity	+	+	+	n/a	n/a	+
quality	+	+	+	n/a	+	+
equity	o	o	+	n/a	+	o
large ontologies	-	o	+	n/a	+	+
schema modifications	-	-	+	o	+	+
instance modifications	+	+	+	o	+	+
dissemination	+	+	+	-	o	+
intelligibility	+	+	+	-	o	+

Table 2.3: Comparison of Instance Matching Benchmarks: "+", fully satisfied, "o" partially satisfied and "-" not satisfied.

- *intelligibility* ensures that the reference alignments and the alignments produced by the systems should be available.

Zaiss et. al. [16] adds four more aspects to the previously mentioned ones to evaluate instance matching systems. The majority of these aspects refer to the *types* of modifications applied to produce the target ontology from the reference ontology. Namely, these are *schema* and *instance modifications*. Table 2.3 summarizes the comparison of the discussed benchmarks according to the aforementioned dimensions.

From Table 2.3 we can see that the OAEI benchmarks satisfy the *systematic procedure*, *continuity*, *quality*, *dissemination* and *intelligibility* requirements. Hence these benchmarks can be eventually adopted as a standard for testing instance matching tools and techniques. STBenchmark and ONTOBI do not satisfy the "continuity" requirement which is ensured only when the benchmarks are used by a number of instance matching systems over the years. ONTOBI is publicly available and consequently partially satisfies the dissemination and intelligibility requirements.

The benchmarks that use DBpedia ontologies (OAEI and ONTOBI) are in general of *good quality*: Wikipedia data is reliable since the instances are assigned to correct concepts and for matching purposes it does not matter if the information is always up-to-date. On the other hand, the STBenchmark is mostly an ontology modifier and consequently it cannot be judged for its quality. *Equity* is ensured only by using different modifications and combinations of the reference ontology to produce the target data such that no type of matching system is favored or disadvantaged. OAEI benchmarks partially satisfy this requirement: instance-based methods are disadvantaged due to the lack of a reasonable amount of instances. Furthermore, the ontologies are quite small and there are no modifications executed at the instance level (except in the case of the IIMB Benchmark).

2.2 Benchmark generators

Semantic Web INstance Generation (SWING) [23] is a benchmark data generator. More specifically, SWING provides a general framework for creating benchmarks to be used by instance matching tools; SWING supports a number of transformations on *values* such as (blank character addition and deletion), changing the dates and number formats, abbreviations, addition of random characters, use of synonyms, shuffling, addition and deletion of tokens; transformations on *structure* are also supported, those being changes in property depth, deletions and additions of properties as well as splitting of property values. Finally, SWING also provides *schema transformations* that include the deletion of class, inversion of properties, changes in the property hierarchy and finally use of disjoint classes. The SWING benchmarking framework supports a superset of the transformations supported by the aforementioned benchmarks but only includes a few semantic variations namely the ones along the class and property hierarchy. It also produces along with the transformed dataset, a gold standard that records the matched instances and is used by the matching tools to measure their performance.

Entity Matching Benchmark (EMBench) [26] is a benchmark generator for relational data designed on the same principles as SWING. EMBench considers only value and structural transformations and similar to SWING is built on the creation of matching scenarios, but on contrary to it, it does not produce a gold standard.

To sum up, there is no single benchmark that tackles both the scalability and the data diversity (mainly the logical transformations) aspects sufficiently. SPIM-

Bench is a *synthetic benchmark* for the *semantic publishing domain*. As discussed in Section 1, it is based on an real ontology provided by BBC and applies the *value* and *structural* transformations proposed by SWING. In addition to those transformations, SPIMBench also supports *logical transformations* that refer to schema constructs; those kinds of transformations are not considered by the aforementioned benchmarks (real or synthetic ones). In addition, SPIMBench generator can produce large datasets (up to billions of triples) thereby addressing the scalability aspects of instance matching systems. In addition, the SPIMBench generator extends the SPB generator that produces datasets using distributions that mimic real world data. Hence, SPIMBench advances the state of the art regarding instance matching benchmarks. Last but not least, SPIMBench produces a *weighted* gold standard that can be used to test the performance of the systems regarding their ability to discover the matches, where weights represent the *similarity distance* between the instances in the source and target datasets. A detailed presentation and comparison of the benchmarks is given in [27].

Chapter 3

Preliminaries

The objective of the Semantic Web is to build an infrastructure of machine-readable semantics for data on the Web. The Resource Description Framework (RDF) [6] enables the encoding, exchange, and reuse of structured data, while providing the means for publishing both human-readable and machine-processable vocabularies.

The popularity of the RDF data model [6] and RDF Schema language (RDFS) [20] is due to the flexible and extensible representation of information, independently of the existence or absence of a schema, under the form of *triples.* A triple is of the form *(subject, predicate, object)* where the *predicate* (also called property) denotes the *relationship* between *subject* and *object.* An RDF triple, (s,p,o), asserts the fact that *subject* is associated with *object* through *property.* An *RDF graph* is a *set of triples* and can be viewed as a *node* and *edge labeled directed graph* with subjects and objects of triples being the nodes of the graph and predicates the edges.

RDF Schema (RDFS) language [20] provides a built-in vocabulary for asserting user-defined schemas in the RDF data model and is designed to introduce useful semantics to RDF triples. RDFS names such as `rdf:Resource`, `rdfs:Class` and `rdf:Property` could be used as objects of triples describing *class* and *property* types. RDFS also provides some useful relationships (properties) between resources, like *subsumption* or *instantiation.*

The OWL Web Ontology Language [5] is designed for use by applications that need to process the content of information instead of just presenting information to humans, and is used to (a) *create an ontology*, (b) *state facts* about a domain and (c) *reason about ontologies* to determine consequences of what was named and stated. OWL provides a much richer set of constructs and semantics than RDFS that allows more complicated reasoning. It has three increasingly-expressive sub-languages, namely OWL-Lite, OWL-DL, and OWL-Full. OWL incorporates the RDFS semantics and in addition to those, OWL distinguishes between *object* and *data* type properties, supports the definition of *class descriptions* and *axioms*; in addition, it defines *property schema constructs*, properties that define relations to others, supports *global cardinality restrictions* on properties and the specification of *logical characteristics* for properties. OWL2 [28], a sucessor of OWL is a powerful language of high complexity that has led to new opportunities in reasoning. OWL2 provides five different sublanguages *Full*, *DL*, *RL*, *EL* and *QL* that trade off expressivity for tractability and speed of reasoning.

Complex class descriptions are specified using (a) *enumeration*, (b) *property restriction* through value and cardinality constraints and (c) *sets operations* on classes, namely intersection, union and complementation. Class axioms refer to the specification of *subsumption*, *equivalence* and *disjointness* of classes. OWL (through RDFS) provides support for *subsumption*, and the definition of *domain* and *range* of properties. Similar to classes, OWL allows the specification of relations to other properties such as *equivalent* and *inverse*; global cardinality restrictions are specified by defining *functional* and *inverse functional* properties; properties can be defined to be *transitive* and/or *symmetric.*

Last, OWL allows the specification of axioms for *individuals* or *instances*, such as *class membership*, *property values* as well as facts about the instance identity. More specifically, OWL allows one to specify that two instances refer to the *same* or to a *different* real world individual.

The OWL constructs along with a partial axiomatization in the form of first order implications that we use in this work are shown in Table 3.1 that describes the semantics of *Class Axioms*, Table 3.3 that gives the semantics of *Classes*; the semantics of *schema vocabulary* are shown in Table 3.4 and finally the semantics of *property axioms* and *equality* are given in Tables 3.2 and 3.5 and respectively. The semantics are given as quantified first-order implications over a ternary predicate T that represents an RDF triple; hence, $T(s, p, o)$ represents a triple with subject s, predicate p and object o. If the **If** part of the rule is empty, then it means that the statement is always true, and if the conclusion of the rule is *false* then there is a contradiction.

	If	**Then**
CAX-SCO	$(?c_1, \texttt{rdfs:subClassOf}, ?c_2)$ $(?x, \texttt{rdf:type}, ?c1)$	$(?x, \texttt{rdf:type}, ?c2)$
CAX-EQC1	$(?c_1, \texttt{owl:equivalentClass}, ?c_2)$ $(?x, \texttt{rdf:type}, ?c_1)$	$(?x, \texttt{rdf:type}, ?c_2)$
CAX-EQC2	$(?c_1, \texttt{owl:equivalentClass}, ?c_2)$ $(?x, \texttt{rdf:type}, ?c_2)$	$(?x, \texttt{rdf:type}, ?c_1)$
CAX-DW	$(?c_1, \texttt{owl:disjointWith}, ?c_2)$ $(?x, \texttt{rdf:type}, ?c_1)$ $(?x, \texttt{rdf:type}, ?c_2)$	FALSE
CAX-ADC	$(?x, \texttt{rdf:type}, \texttt{owl:AllDisjointClasses})$ $(?x, \textit{owl:members}, ?y)$ LIST[$?y$, $?c_1$, …, $?c_n$] $(?z, \texttt{rdf:type}, ?c_i)$ $(?z, \texttt{rdf:type}, ?c_j)$	FALSE

Table 3.1: Semantics of Class Axioms

According to [20, 29] if a class c_1 is a subclass of c_2 (triple $(c_1, \texttt{rdfs:subClassOf}, c_2)$), then the instances of the former $(x, \texttt{rdf:type}, c_1)$ are also instances of the latter $(x, \texttt{rdf:type}, c_2)$. The same holds for subsumption between properties. Rules CAX-SCO in Table 3.1 and PRP-SPO1 in Table 3.2 describe these semantics.

Just like individuals, *equivalent classes* and *properties* appear in the Web of Data, so OWL2 provides the constructs `owl:equivalentClass` and `owl:equivalentProperty` respectively, to denote classes (or properties) that denote the same real-world *concept* (or *property*). The rules for equivalence are shown in Tables 3.1 and 3.2. Being equivalent, classes share the same *instances* (rules CAX-EQC1, CAX-EQC2), *subclasses*, *superclasses*, *properties*; the same holds for equivalent properties as well (rules PRP-EQP1, PRP-EQP2).

Defining two classes c_1, c_2 as *disjoint* implies that they cannot share common instances. Disjointness is denoted using the `owl:disjointWith` construct. Dis-

	If	**Then**
PRP-FP	$(?p, \texttt{rdf:type}, \texttt{owl:FunctionalProperty})$ $(?x, ?p, ?y_1)$ $(?x, ?p, ?y_2)$	$(?y_1, \texttt{owl:sameAs}, ?y_2)$
PRP-IFP	$(?p, \texttt{rdf:type}, \texttt{owl:InverseFunctionalProperty})$, $(?x_1, ?p, ?y)$ $(?x_2, ?p, ?y)$	$(?x_1, \texttt{owl:sameAs}, ?x_2)$
PRP-EQP1	$(?p_1, \texttt{owl:equivalentProperty}, ?p_2)$ $(?x, ?p_1, ?y)$	$(?x, p_2, ?y)$
PRP-EQP2	$(?p_1, \texttt{owl:equivalentProperty}, ?p_2)$ $(?x, ?p_2, ?y)$	$(?x, p_1, ?y)$
PRP-PDW	$(?P_1, \texttt{owl:propertyDisjointWith}, ?P_2)$ $(?x, ?P_1, ?y)$ $(?x, ?P_2, ?y)$	FALSE
PRP-ADP	$(?x, \texttt{rdf:type}, \texttt{owl:AllDisjointProperties})$ $(?x, \mathit{owl:members}, ?y)$ LIST$[?y,\ ?P_1,\ ?P_2,\ \ldots\ ?P_n]$ $(?u, ?P_1, ?z)$ $(?u, ?P_2, ?z)$	FALSE
PRP-SPO1	$(?p_1, \texttt{rdfs:subPropertyOf}, ?p_2)$ $(?x, ?p_1, ?y)$	$(?x, ?p_2, y)$

Table 3.2: Semantics of Axioms about Properties

jointness between classes is generalized to multiple ones using the `owl:AllDisjointClasses` construct. The semantics of said constructs implemented by rules CAX-DW, CAX-ADC are shown in Table 3.1. Similar constructs `owl:propertyDisjointWith`, `owl:AllDisjointProperties` exist for specifying *disjoint properties*, i.e., properties that cannot share common instances. Rules PRP-ADP, PRP-PDW of Table 3.2 show some consequences of the semantics of the above constructs.

Inverse functional and functional properties are useful to denote values that uniquely identify an entity. Note that, due to the fact that the semantics of OWL2 do not include the Unique Name Assumption (UNA), functional and inverse functional properties should not be viewed as integrity constraints, because they cannot directly (by themselves) lead to contradictions. Instead, they force us to assume (infer) that certain individuals are the same, as indicated by rules PRP-FP and PRP-IFP in Table 3.2.

	If	Then
CLS-INT1	$(?c, \texttt{owl:intersectionOf}, ?x)$ LIST[$?x$, $?c_1$, …, c_n] $(?y, \texttt{rdf:type}, ?c_1)$ … $(?y, \texttt{rdf:type}, ?c_n)$	$(?y, \texttt{rdf:type}, ?c)$
CLS-INT2	$(?c, \texttt{owl:intersectionOf}, ?x)$ LIST[$?x$, $?c_1$, …, c_n] $(?y, \texttt{rdf:type}, ?c)$	$(?y, \texttt{rdf:type}, ?c_1)$ $(?y, \texttt{rdf:type}, ?c_2)$ $(?y, \texttt{rdf:type}, ?c_3)$ … $(?y, \texttt{rdf:type}, ?c_n)$

Table 3.3: Semantics of Classes

The `owl:unionOf` construct is used to construct a new class, that is the union of two (or more) other classes. Dually, the `owl:intersectionOf` construct is used to construct a new class that is the intersection of two (or more) other classes. As with all OWL constructs, the semantics of `owl:unionOf` are intentional, i.e., all instances that are *known* to be instances of either of c_1, c_2 will be also instances of their union, and vice-versa, i.e., known instances of the union will be instances of either c_1 or c_2 (or both). According to rules SCM-UNI and SCM-INT shown in Table 4.10, a class c defined as *union* (respectively *intersection*) of a set of existing classes c_1, c_2 … c_n, then c is inferred as their *superclass* (respectively *subclass*).

According to the *Semantics of Schema Vocabulary* [30], class and property subsumption are *transitive* (see Rules SCM-SCO, SCM-SPO respectively in Table 3.4). More specifically, the existence of $(c_1, \texttt{rdfs:subClassOf}, c_2)$ and (c_2,`rdfs:subClassOf`, c_3) in a dataset should cause the inference of $(c_1, \texttt{rdfs:subClassOf}, c_3)$.

Obviously, a pair of individuals cannot be the same and different at the same time, thus the rule EQ-DIFF1. By its definition, `owl:sameAs` has the properties of equivalence relations, i.e., it is reflexive, symmetric and transitive. *Reflexivity* implies that $(x, \texttt{owl:sameAs}, x)$ for all resources x (rule EQ-REF). The relation being *symmetric* means that $(x, \texttt{owl:sameAs}, y)$ implies $(y, \texttt{owl:sameAs}, x)$ (rule EQ-SYM). Finally, *transitivity* implies that from $(x, \texttt{owl:sameAs}, y)$ and $(y, \texttt{owl:sameAs}, z)$ we should infer $(x, \texttt{owl:sameAs}, z)$ (rule EQ-TRANS).

	If	**Then**
SCM-INT	$(?c, \texttt{owl:intersectionOf}, ?x)$ LIST[$?x$, $?c_1$, …, c_n]	$(?c, \texttt{rdfs:subClassOf}, ?c_1)$ $(?c, \texttt{rdfs:subClassOf}, ?c_2)$ … $(?c, \texttt{rdfs:subClassOf}, ?c_n)$
SCM-UNI	$(?c, \texttt{owl:unionOf}, ?x)$ LIST[$?x$, $?c_1$, …, c_n]	$(?c_1, \texttt{rdfs:subClassOf}, ?c)$ $(?c_2, \texttt{rdfs:subClassOf}, ?c)$ … $(?c_n, \texttt{rdfs:subClassOf}, ?c)$
SCM-SPO	$(?p_1, \texttt{rdfs:subPropertyOf}, ?p_2)$ $(?p_2, \texttt{rdfs:subPropertyOf}, ?p_3)$	$(?p_1, \texttt{rdfs:subPropertyOf}, ?p_3)$
SCM-SCO	$(?c_1, \texttt{rdfs:subClassOf}, ?c_2)$ $(?c_2, \texttt{rdfs:subClassOf}, ?c_3)$	$(?c_1, \texttt{rdfs:subClassOf}, ?c_3)$

Table 3.4: Semantics of Schema Vocabulary

EQ-TRANS	$(?x, \texttt{owl:sameAs}, ?y)$ $(?y, \texttt{owl:sameAs}, ?z)$	$(?x, \texttt{owl:sameAs}, ?z)$
EQ-DIFF1	$(?x, \texttt{owl:sameAs}, ?y)$ $(?x, \texttt{owl:differentFrom}, ?y)$	FALSE

Table 3.5: Semantics of Equality

Chapter 4

SPIMBench: Semantic Publishing Instance Matching Benchmark

In this Chapter we discuss the *Semantic Publishing Instance Matching Benchmark*. More specifically, we give in Section 4.1 a description of the employed schemas. Section 4.2 discusses the proposed metrics and the transformations supported by SPIMBench are presented in Section 4.3. The data generator is discussed in Section 4.4 and the gold standard in Section 4.5. Finally, Section 4.6 discusses scalability experiments for SPIMBench.

4.1 SPIMBench Schema

SPIMBench uses seven *core* and three *domain* RDF ontologies provided by BBC. The former define the main entities and their properties, required to describe essential concepts of the benchmark namely, *creative works*, *persons*, *documents*, BBC *products* (news, music, sport, education, blogs), *annotations* (*tags*), *provenance* of resources and *content management system* information. The latter are used to express concepts from a *domain of interest* such as football, politics, entertainment among others. The employed ontologies have 74 classes, 88 and 28 *data* and *object* type properties respectively. They contain 60 `rdfs:subClassOf`, 17 `rdfs:subPropertyOf`, 105 `rdfs:domain` and 115 `rdfs:range` RDFS [20] properties. On the other hand the ontologies contain a limited number of OWL [31] constructs: they contain 8 `owl:oneOf` class axioms that allow one to define a class by enumeration of its instances and one `owl:TransitiveProperty` property. Although the ontologies consider a non negligible number of classes and properties, class and property hierarchies are very shallow. More specifically, the class hierarchy has a maximum depth of *3* whereas the property hierarchy has a depth of *1*. A detailed presentation of the ontologies employed by SPIMBench can be found in [32].

In this section we discuss briefly a fragment of the *Creative Works* core ontology shown in Figure 4.1. Ontologies are represented as *node* and *edge labeled directed* graphs where *classes* and *their instances* are depicted by an *oval*, and *properties* as *edges* between nodes, where the name of the property is the *label* of the edge.

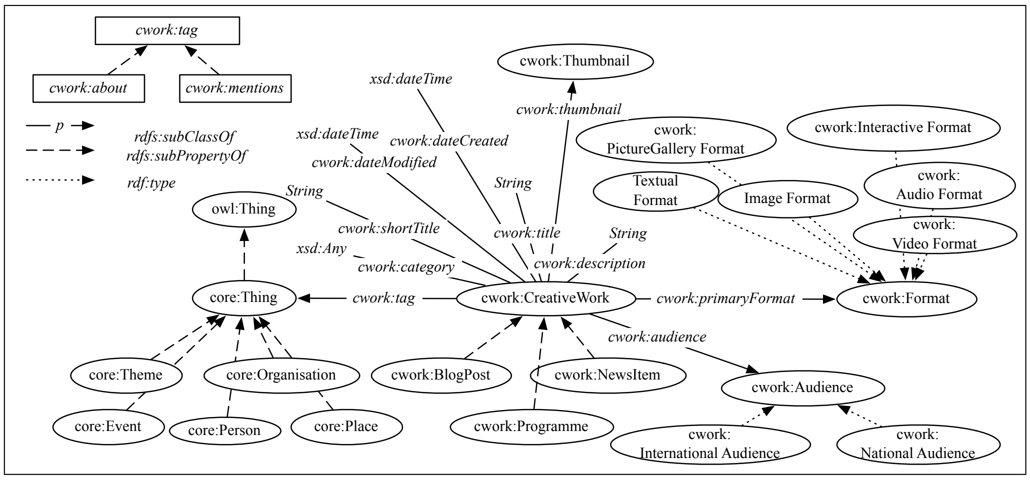


Figure 4.1: BBC Creative Works Ontology

The main class is `cwork:CreativeWork` (shown in Figure 4.1) that collects all RDF descriptions of creative works (also called *journalistic assets*) created by the publisher's editorial team. This class is defined as a subclass of `core:Thing` (subclass of *owl:Thing*), allowing in this way the creation of complex information graphs. A creative work has a number of properties such as `cwork:title`, `cwork:shortTitle`, `cwork:description`, `cwork:dateModified`, `cwork:dateCreated`, cwork:audience, cwork:format among others; it has a category (property cwork: category) and can be tagged (property cwork:tag) with *anything* (i.e. instances of class *owl:Thing*). The latter property is further specialized (through the rdfs:subPropertyOf relation) to properties cwork:about and cwork:mentions that are heavily used during the creation of creative works and subsequently in SPIMBench transformation processes. Creative works can be instances of classes cwork:NewsItem, cwork:Programme and cwork:BlogPost, all defined as subclasses of class cwork: CreativeWork.

The BBC ontologies also use classes such as core:Place, core:Event, core: Organisation, core:Person, and core:Theme, all defined as subclasses of class core:Thing. Figure 4.2 provides an example of a creative work in RDF turtle format. GeoNames[1] reference dataset has been included for further enriching the annotations with geo-locations data to enable the formulation of geo-spatial queries.

SPIMBench also uses *reference datasets* that are employed by the data generator to produce the data of interest. These datasets are snapshots of the real datasets provided by BBC.

In order to incorporate more OWL constructs, we extended the BBC ontologies with concepts from DBpedia[2] and FOAF[3] ontologies. More specifically, we used the FOAF class foaf:Person, and the DBpedia classes dbpedia:Place, dbpedia:Event, dbpedia:Organisation, dbpedia:Sport all defined as *equivalent* to the classes with the same name in the BBC ontologies using the owl:equivalentClass property; all classes were defined as *subclasses* of core:Thing. We do not include all their properties as those are defined in the ontologies, but focus only on the ones that are useful in the context of SPIMBench.

Moreover, we used a set of properties from those classes and declared them as *equivalent* to properties with the same label defined in their equivalent DBpedia and FOAF classes; equivalence was defined through the owl:equivalentProperty property. We have also included classes from the Travel Ontology that defines

[1]GeoNames: http://www.geonames.org/
[2]DBpedia: dbpedia.org
[3]The Friend of a Friend (FOAF) project: http://www.foaf-project.org/

travel-related entities[4], all defined as *subclasses* of BBC class `core:Thing`.

```
<http://www.bbc.co.uk/things/1#id> a cwork:NewsItem ;
  cwork:title "Grant Shapps cup can territorial practiced partisan countries attract
             ambition where wrestling." ;
  cwork:shortTitle "adoption does secular personal competition court cup." ;
  cwork:category <http://www.bbc.co.uk/category/PoliticsPersonsReference> ;
  cwork:description "their commerce countries decided amassed one method
              merely hard participants combined so administer private enactments." ;
  cwork:about dbpedia:Llangyfelach , dbpedia:Emily_Thornberry ,dbpedia:Northern_Fury_FC ,
              dbpedia:Lisa_Howard_(reporter) , dbpedia:Alun_Cairns ;
  cwork:mentions geonames:2657358 ;
  cwork:audience cwork:NationalAudience ;
  cwork:liveCoverage "false"^^<http://www.w3.org/2001/XMLSchema#boolean> ;
  cwork:primaryFormat cwork:TextualFormat , cwork:InteractiveFormat ;
  cwork:dateCreated "2011-02-21T01:17:16.916+02:00"
              ^^<http://www.w3.org/2001/XMLSchema#dateTime>;
  cwork:dateModified "2011-06-26T18:26:45.900+03:00"
              ^^<http://www.w3.org/2001/XMLSchema#dateTime>;
  cwork:thumbnail <http://www.bbc.co.uk/thumbnail/1907108784> ;
  cwork:altText "thumbnail atlText for CW http://www.bbc.co.uk/context/1#id" ;
  bbc:primaryContentOf <http://www.bbc.co.uk/things/154167351#id> .

<http://www.bbc.co.uk/things/154167351#id>
  bbc:webDocumentType bbc:Mobile ;
  bbc:productType bbc:Education ;
  core:primaryTopic dbpedia:Les_Fradkin .

<http://dbpedia.org/resource/Les_Fradkin> a foaf:Person ;
  foaf:name "Les Fradkin" ;
  foaf:surname "Fradkin" ;
  foaf:givenName "Les" ;
  dc:description "guitarist" .
```

Figure 4.2: Example: Creative Work Instance

[4]Travel Ontology:`http://swatproject.org/travelOntology.asp`

```
@prefix dbpedia-owl: <http://dbpedia.org/ontology/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix core:    <http://www.bbc.co.uk/ontologies/coreconcepts/> .
@prefix travel: <http://www.co-ode.org/roberts/> .
@prefix owl: <http://www.w3.org/2002/07/owl> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema> .

dbpedia-owl:Organisation rdfs:subClassOf core:Thing .
dbpedia-owl:Place rdfs:subClassOf core:Thing .
dbpedia-owl:Theme rdfs:subClassOf core:Thing .
dbpedia-owl:Event rdfs:subClassOf core:Thing .

foaf:Person rdfs:subClassOf core:Thing .

travel:AdministrativeDivision rdfs:subClassOf core:Thing .
travel:TierOneAdministrativeDivision rdfs:subClassOf travel:AdministrativeDivision .
travel:GeographicalFeature rdfs:subClassOf core:Thing .
travel:bodyOfLand rdfs:subClassOf travel:GeographicalFeature .
travel:Continent rdfs:subClassOf travel:bodyOfLand .
travel:Island rdfs:subClassOf core:Thing .
travel:City rdfs:subClassOf core:Thing .
travel:Coastline rdfs:subClassOf core:Thing .
travel:Country rdfs:subClassOf core:Thing .
travel:EuropeanIsland rdfs:subClassOf core:Thing .
travel:City rdfs:subClassOf core:Thing .
travel:River rdfs:subClassOf core:Thing .
travel:Recognised rdfs:subClassOf core:Thing .
travel:River rdfs:subClassOf core:Thing .

dbpedia-owl:Organisation owl:equivalentClass core:Organisation .
foaf:Person rdfs:subClassOf core:Person .
dbpedia-owl:Place rdfs:subClassOf core:Place .
dbpedia-owl:Theme rdfs:subClassOf core:Theme .
dbpedia-owl:Event rdfs:subClassOf core:Event .

cwork:NewsItem owl:disjointWith cwork:Programme .
cwork:NewsItem owl:disjointWith cwork:BlogPost .

ldbc:Thing owl:unionOf
   ( foaf:Person  dbpedia-owl:Event
     dbpedia-owl:Organisation dbpedia-owl:Place core:Theme ) .

ldbc:Person_Organisation owl:intersectionOf
   ( foaf:Person dbpedia-owl:Organisation ) .
ldbc:Individual_Corporation owl:intersectionOf
   ( foaf:Person dbpedia-owl:Organisation ) .

ldbc:Event_Place_Theme owl:intersectionOf
   ( dbpedia-owl:Event dbpedia-owl:Place core:Theme  ) .
ldbc:Happening_Spot owl:intersectionOf
   ( dbpedia-owl:Event dbpedia-owl:Place ) .
```

Figure 4.3: Example: SPIMBench FOAF, Travel and DBpedia `rdfs:subClassOf`, `owl:equivalentClass`, `owl:disjointWith`, `owl:intersectionOf`, `owl:unionOf` Schema triples (a)

```
@prefix dbpedia-owl: <http://dbpedia.org/ontology/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix core:    <http://www.bbc.co.uk/ontologies/coreconcepts/> .
@prefix travel: <http://www.co-ode.org/roberts/> .
@prefix owl: <http://www.w3.org/2002/07/owl> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos> .

travel:hasArea rdfs:subPropertyOf travel:hasPhysicalProperty .
travel:hasPopulation rdfs:subPropertyOf travel:hasStatistic .

foaf:name owl:equivalentProperty core:name .
foaf:surname owl:equivalentProperty core:surname .
foaf:givenName owl:equivalentProperty core:givenName .

dbpedia-owl:country owl:equivalentProperty core:country .
dbpprop:country owl:equivalentProperty core:country .

dbpprop:population owl:equivalentProperty core:population .
dbpprop:manager owl:equivalentProperty core:manager .
dbpprop:name owl:equivalentProperty core:name .
dbpprop:nickname owl:equivalentProperty core:nickname .
dbpprop:website owl:equivalentProperty core:website .
dbpprop:caption owl:equivalentProperty core:caption .
dbpprop:equipment owl:equivalentProperty core:equipment .
dbpprop:olympic owl:equivalentProperty core:olympic .
dbpprop:team owl:equivalentProperty core:team .

dcterms:subject owl:equivalentProperty core:subject .

geo:geometry owl:equivalentProperty core:geometry .

dc:description owl:equivalentProperty core:description .

dbpedia-owl:birthPlace owl:equivalentProperty core:birthPlace .
dbpedia-owl:birthDate owl:equivalentProperty core:birthDate .
dbpedia-owl:deathPlace owl:equivalentProperty core:deathPlace .
dbpedia-owl:deathDate owl:equivalentProperty core:deathDate .


core:primaryTopic  rdf:type owl:ObjectProperty, owl:FunctionalProperty .

bbc:primaryContentOf rdf:type owl:ObjectProperty , owl:InverseFunctionalProperty .

[ rdf:type owl:AllDisjointProperties ;
  owl:members ( core:facebook core:officialHomepage core:twitter )
] .
```

Figure 4.4: Example: SPIMBench FOAF, Travel and DBpedia `rdfs:subPropertyOf`, `owl:equivalentProperty`, `owl:FunctionalProperty`, `owl:inverseOf` and `owl:AllDisjointProperties` Schema triples (b)

As mentioned above, we did not include all classes of the aforementioned ontologies but a subset thereof; in addition, we included only a subset of their properties. More specifically, for `dbpedia:Event`, we focused on properties `rdfs:label`, rdfs:comment, dbpedia–owl:country and dcterms:subject; for class dbpedia: Organisation we included data properties rdfs:label and rdfs:comment as well as the object properties dbpprop:manager, dbpprop:name, dbpprop:nickname and dbpprop:website. For class dbpedia:Sport we keep data properties rdfs:comment and dbpprop:caption, and object properties dbpprop:olympic, dbpprop:team and dbpprop:equipment. Last in the case of class dbpedia:Place we used data properties foaf:name, rdfs:comment and object properties dbpedia–owl:country and geo:geometry.

Regarding the FOAF ontology we focused our attention on the foaf:Person class; we considered its data type properties foaf:name,foaf:surname, foaf: given-Name, dc:description, dbpedia–owl:birthDate,dbpedia–owl:deathDate and object properties dbpedia–owl:birthPlace and dbpedia–owl:deathPlace.

From the Travel ontology we included classes travel:AdministrativeDivision, travel:bodyOfLand, travel:City, travel:TierOneAdministrativeDivision, travel:Coastline, travel:Continent, travel:Country, travel:Island, travel: EuropeanIsland, travel:River to create a class hierarchy of length 3 with its root being class owl:Thing. Finally, we also considered classes travel:Recognised defined as a subclass of owl:Thing.

The enhanced SPIMBench schema contains :

rdfs:Class	31	owl:equivalentProperty	18
owl:equivalentClass	8	owl:disjointWith	8
owl:DatatypeProperty	38	owl:ObjectProperty	98
owl:FunctionalProperty	3	owl:InverseFunctionalProperty	1
rdfs:subClassOf	83	rdfs:subPropertyOf	19
rdfs:range	145	rdfs:domain	134
owl:unionOf	1	owl:intersectionOf	4
owl:oneOf	12		

Table 4.1: SPIMBench Schema

We have also incorporated reference datasets from the DBpedia, FOAF and Travel ontologies.

From DBpedia we obtained:

`dbpedia:Event`	2416
`dbpedia:Organisation`	2368
`dbpedia:Place`	2345
`dbpedia:Sport`	139
`foaf:Person`	1276

Table 4.2: SPIMBench DBpedia Instances

From Travel we obtained:

`travel:AdministrativeDivision`	372	`travel:City`	57
`travel:Coastline`	23	`travel:Continent`	6
`travel:Country`	55	`travel:GeographicalFeature`	905
`travel:Island`	591	`travel:Ocean`	5
`travel:River`	49	`travel:bodyOfLand`	598
`travel:Person_agent`	4	`travel:TierOneAdministrativeDivision`	21

Table 4.3: SPIMBench Travel Instances

Figures 4.3 and 4.4 show a part of the triples considered from the DBpedia, FOAF and Travel ontologies and their relationship with the core BBC ontologies.

4.2 Metrics

The metrics that SPIMBench supports to test how systems perform are the following:

- Precision/ Recall / F-measure: these metrics are used to determine the *effectiveness* of the instance matching systems. We use the standard definition of *precision*, *recall* and *f-measure*: *precision* is the fraction of the *intersection* of the *relevant* and *retrieved* instances over the *retrieved instances*, whereas *recall* is the fraction of the *intersection* of *relevant* and *retrieved instances* over the *relevant instances*. In the case of instance matching, retrieved instances are the instances matched by the instance matching systems, and the relevant instances are the matched instances that are also reported in the provided *gold standard*. Precision can be seen as a measure of *exactness* or *quality*, whereas recall is a measure of *completeness*. *F-measure* is a metric that combines precision and recall. It is calculated as their *harmonic mean*. When comparing the results of the instance matching process with the gold standard, one can calculate the *true positive* (tp) (correct), the *false positive* (fp) (unexpected) and the *false negative* (fn) (missing) results. Precision, recall and f-measure can then be computed as follows:

$$\begin{aligned} precision &= \frac{tp}{tp + fp} \\ recall &= \frac{tp}{tp + fn} \\ fmeasure &= 2 \times \frac{precision \times recall}{precision + recall} \end{aligned}$$

- RUN TIMES The matched systems should also record the time needed to discover the matches when comparing the source and target datasets. The running times should be reported in seconds. The time that it takes for an instance matching system to compute the matches is an important criteria, but not as important as precision, recall and F-measure since such systems are judged primarily on the basis on their results: systems with higher quality results are more preferable than ones with lower quality, even if the latter compute matches faster.

4.3 Transformations

As mentioned above, SPIMBench supports all kinds of *transformations*, namely *lexical*, *structural* and *logical* [21] as well as *simple* and *complex* ones. Transformations are applied on *source instances* to obtain a set of *target instances*; this pair of instances are then used as input by an instance matching system (along with the gold standard) to test their performance.

4.3.1 Lexical/Value Transformations

Lexical or *Value* transformations refer to mainly typographical errors and the use of different data formats. In SPIMBench we use the transformations shown in Table 4.4 that have been proposed and implemented in SWING [23] and in OAEI [22].

Each transformation takes as input a *data type property* as specified in the benchmark's schema, and a *severity* that determines the importance of the modification.

- Value transformations VT1 and VT2 refer to the addition/deletion of blank or random character in a string whereas VT2 also refers to the modification of a random character).
- VT3 refers to the deletion,addition and shuffling of a token (i.e., sequence of characters) in a string. Transformations

VT1 - VT3 are different cases of *mispellings*, a category of transformations that is rather common in practice especially in the case in which data has been created by humans which is exactly the case with the semantic publishing scenario.

- VT4 concerns the use of different standards employed for representing values, especially in the case of dates. We consider four types of date transformations, namely short, medium, long and full. For example, date "1990-10-17", will be transformed to "10/17/90" for short, "Oct 17, 1990" for medium, "October 17, 1990" for long and "Wednesday, October 17, 1990" for a full transformation.
- VT5 refers to the change of a number to another that we generate randomly.
- VT6 refers to possible abbreviations that can be found in texts such as "United States of America" vs "USA"; to support this last transformation we used a list of publicly available country names and their abbreviations.
- VT7 refers to transformations related to the use of synonyms and antonyms that have been taken from Wordnet[5]; for instance, "poor" and its synonym "inadequate".
- In a real world scenario such as the one that we are discussing in this paper, authors employ different words to convey one meaning and in general the use of synonyms is present in any text. Stemming is applied using transformation

[5]http://wordnet.princeton.edu/

VT8.

- SPIMBench also supports *multilinguality* (transformation VT9) from English to 65 languages[6].

The last two transformations are characterized as *semantic variations* [26].

VT1	Blank Character Addition/Deletion
VT2	Random Character Addition/Deletion/Modification
VT3	Token Addition/Deletion/Shuffle
VT4	Date Format
VT5	Change Number
VT6	Abbreviation
VT7	Synonym/Antonym
VT8	Stem of a Word
VT9	Multilinguality

Table 4.4: SPIMBench Lexical/Value Transformations

4.3.2 Structural transformations

This type of transformations refers to the changes that occur to the properties of instances such as *splitting*, *aggregation*, *deletion* and *addition*. Splitting refers to expanding properties (e.g., property *address* could be split to *street* and *number*) whereas aggregation refers to merging a number of properties to a single one, such as *firstName* and *lastName* to *fullName*. In SPIMBench we support all the structural transformations that are proposed and implemented in SWING (the latter does not support aggregation of properties). These transformations are a superset of those considered in the majority of the benchmarks discussed in Chapter 2.

4.3.3 Logical Transformations

Logical modifications are primarily used to test if the matching systems take into consideration RDFS and OWL constructs to discover matches between instances that can be found only when considering schema information; these modifications

[6]The language to which the English texts are translated is a parameter defined in a configuration file.

go beyond those discussed above. To the best of our knowledge, SPIMBench is the first instance matching benchmark that considers schema constructs when producing the target from source instances. The RDFS/OWL constructs that we consider in SPIMBench are:

- *instance (in)equality* constructs `owl:sameAs`, `owl:differentFrom`
- *equivalence* schema constructs `owl:equivalentProperty`, `owl:equivalent-Class`
- *disjointness* schema constructs `owl:disjointWith`, `owl:propertyDisjoint-With`
- RDFS *schema* constructs namely `rdfs:subClassOf`, `rdfs:subPropertyOf`
- *property constraints* namely `owl:FunctionalProperty` and `owl:Inverse-FunctionalProperty` and finally
- constructs supporting *complex class definitions* using the `owl:intersection-Of` and `owl:unionOf` features.

Tables 4.5 - 4.10 present the set of tests based on the logical transformations that we consider in SPIMBench. Columns SD and TD refer to the *source* and *target* data respectively i.e., the latter are the instances we obtain by applying transformations to the former ones. Column SCHEMA TRIPLES refers to *schema triples* that the instance matcher under test should take into consideration when performing the matching tasks. Finally, column GS stores the entries of the gold standard. In all the tables we write u to refer interchangeably to an RDF instance and its URI. We write $u \sim u'$ to state that u and u' are matched instances.

Tests **LTSUBC**, **LTEQC** shown in Table 4.5 consider the `rdfs:subClassOf` and `owl:equivalentClass` constructs resp.; Tests **LTEQP**, **LTSUBP** (given in Table 4.6 resp.) take into account the `rdfs:subPropertyOf` and `owl:equivalent-Property` constructs respectively.

We will discuss first the case of classes. Given an instance u_i of class C in SD, we create an instance u'_i in TD, instance of class C' by copying the properties of u_i (except `rdf:type` triples). In **LTSUBC**, C is a subclass of C' (schema triple $(C,$`rdfs:subClassOf`$,C')$) and in **LTEQC** C and C' are equivalent classes - schema triple $(C,$`owl:equivalentClass`$,C')$.

	SD	TD	SCHEMA TRIPLES	GS
LTSUBC	$(u_1, \texttt{rdf:type}, C)$	$(u'_1, \texttt{rdf:type}, C')$	$(C, \texttt{rdfs:subClassOf}, C')$	$u_1 \sim u'_1$
LTEQC	$(u_1, \texttt{rdf:type}, C)$	$(u'_1, \texttt{rdf:type}, C')$	$(C, \texttt{owl:equivalentClass}, C')$	$u_1 \sim u'_1$

Table 4.5: Tests for `rdfs:subClassOf`, `owl:equivalentClass`

	SD	TD	SCHEMA TRIPLES	GS
LTEQP	(u_1, p_1, o_1)	(u'_1, p_2, o_1)	$(p_1, \texttt{owl:equivalentProperty}, p_2)$	$u_1 \sim u'_1$
LTSUBP	(u_1, p_1, o_1)	(u'_1, p_2, o_1)	$(p_1, \texttt{rdfs:subPropertyOf}, p_2)$	$u_1 \sim u'_1$

Table 4.6: Tests for `rdfs:subPropertyOf`, `owl:equivalentProperty`

The rationale for stating in both cases that the two instances are *matches* is straightforward: in the first case we assume that the instances are of similar type due to the `rdfs:subClassOf` semantics (rule CAX-SCO, Table 3.1 and rule SCM-SCO, Table 3.4); in the second, they are of exactly the same type due to the semantics of class equivalence (rules CAX-EQC1, CAX-EQC2, Table 3.1). The rationale for properties is exactly the same: for **LTSUBP** (`rdfs:subPropertyOf` construct) the two instances are considered as matches when rules PRP-SPO1 in Table 3.2 and SCM-SPO in Table 3.4 are taken into account for `rdfs:subPropertyOf`; for **LTEQP**, we consider the semantics of `owl:equivalentProperty` as specified in PRP-EQP1 and PRP-EQP2 rules (Table 3.2).

Tests **LTSAMEAS1** and **LTSAMEAS2** shown in Table 4.7 consider the `owl-:sameAs` OWL construct. For **LTSAMEAS2** and for an instance of choice u_i, we create, as we discussed, above instance u'_i; We also produce an instance u''_i by applying a large set of modifications on u'_i. In the target dataset TD, we introduce triple $(u'_i, \texttt{owl:sameAs}, u''_i)$; this is necessary in order to challenge instance matching tools regarding their ability to find matches using the `owl:sameAs` links and not simply by matching the property values of instances. If the matcher under test considers the `owl:sameAs` construct, it should produce in addition to the match $u_i \sim u'_i$, the match $u_i \sim u''_i$, something that would not be possible for a matcher otherwise (rule EQ-TRANS, Table 3.5).

	SD	TD	SCHEMA TRIPLES	GS
LTSAMEAS1	$(u_1, \texttt{rdf:type}, C)$ $(u_2, \texttt{rdf:type}, C)$	$(u'_1, \texttt{rdf:type}, C)$ $(u'_2, \texttt{rdf:type}, C)$ $(u'_1, \texttt{owl:sameAs}, u'_2)$		$u_1 \sim u'_1$ $u_1 \sim u'_2$ $u_2 \sim u'_2$ $u_2 \sim u'_1$
LTSAMEAS2	$(u_1, \texttt{rdf:type}, C)$	$(u'_1, \texttt{rdf:type}, C)$ $(u''_1, \texttt{rdf:type}, C)$ $(u'_1, \texttt{owl:sameAs}, u''_i)$		$u_1 \sim u'_1$ $u_1 \sim u''_1$
LTDIFF	$(u_1, \texttt{rdf:type}, C)$	$(u'_1, \texttt{rdf:type}, C)$ $(u''_1, \texttt{rdf:type}, C)$ $(u'_1, \texttt{owl:differentFrom}, u''_1)$		$u_1 \sim u'_1$

Table 4.7: Tests for `owl:sameAs`, `owl:differentFrom`

LTSAMEAS1 is a more complex test: consider two instances u_i and u_j in source dataset SD. Those are transformed as discussed previously to u'_i and u'_j that are inserted in the target dataset TD along with triple $(u'_i, \texttt{owl:sameAs}, u'_j)$. A matcher that understands the semantics of `owl:sameAs` should report all possible matches between instances u_i, u'_i, u_j and u'_j (in total four matches).

OWL Construct `owl:differentFrom` is used to explicitly state that two resources refer to different real world objects. Test **LTDIFF** shown in Table 4.7 follows the same lines as the ones for `owl:sameAs` construct: for an instance u_i in source dataset SD, we create two instances u'_i and u''_i by copying all the properties of u_i and we add triple $(u'_i, \texttt{owl:differentFrom}, u''_i)$. The creation of u''_i is done with very few transformations. If the matcher does not consider the `owl:differentFrom` construct it should produce a match between instances u_i and u''_i, when it should not since there is an explicit statement that these two instances refer to a different real world object (rule EQ-DIFF1, Table 3.5).

Another kind of test we introduce refers to *disjointness* of classes and properties (see Table 4.8). Take for instance **LTDISJC** where we produce target instance u'_i from source instance u_i, instances of *disjoint* classes C' and C - schema triple $(C, \texttt{owl:disjointWith}, C')$ - respectively. In this case, the matcher should not return any match (rule CAX-DW, Table 3.1). Again, u'_i can be obtained from u_i by copying the properties of the former.

	SD	TD	SCHEMA TRIPLES	GS
LTDISJC	$(u_1, \texttt{rdf:type}, C)$	$(u'_1, \texttt{rdf:type}, C')$	$(C, \texttt{owl:disjointWith}, C')$	
LTDISJP	$(u_1, \texttt{rdf:type}, C)$ (u_1, p_1, o_1)	$(u'_1, \texttt{rdf:type}, C)$ (u'_1, p_2, o_1)	$(p_1, \texttt{owl:propertyDisjointWith}, p_2)$	

Table 4.8: Tests for `owl:disjointWith`, `owl:propertyDisjointWith`

Disjointness of properties follows the same rationale as disjointness of classes: in test **LTDISJP** (Table 4.8) we produce an instance u'_i from instance u_i, the former participating in triple (u_i, p_1, o_1) and the latter in triple (u'_i, p_2, o_1). The matcher in this case should not return a match since the two instances cannot share disjoint properties (rule PRP-PDW, Table 3.2).

Other interesting modifications are the ones regarding the use of *functional* (**LTFUNCP**) and *inverse functional* (**LTINVFUNCP**) properties (`owl:FunctionalProperty` and `owl:InverseFunctionalProperty`) shown in Table 4.9. In the case of the former, for an instance u_i in the source dataset SD, subject of triple (u_i, p_i, o_i) with p_i being a functional property, we produce a triple (u_i, p_i, o'_i) in TD. If the matcher takes into consideration the fact that p_i is a functional property, then it should produce a match between instances o_i and o'_i (rule PRP-FP, Table 3.2).

	SD	TD	SCHEMA TRIPLES	GS
LTFUNCP	$(u_1, \texttt{rdf:type}, C)$ (u_1, p, o_1)	$(u_1, \texttt{rdf:type}, C)$ (u_1, p, o_2)	$(p, \texttt{rdf:type}$ $\texttt{owl:FunctionalProperty})$	$o_1 \sim o_2$
LTINVFUNCP	$(u_1, \texttt{rdf:type}, C)$ (u_1, p, o_1)	$(u'_1, \texttt{rdf:type}, C)$ (o_1, p, u'_1)	$(p, \texttt{rdf:type},$ $\texttt{owl:InverseFunctionalProperty})$	$u_1 \sim u'_1$

Table 4.9: Tests for `owl:FunctionalProperty`, `owl:InverseFunctionalProperty`

The test for *inverse functional* properties follows the same rationale (**LTINVFUNCP**, rule PRP-IFP, Table 3.2). For an instance u_i in the source dataset SD, subject of triple (u_i, p_i, o_i) with p_i being an inverse functional property, we produce a triple (o_i, p_i, u'_i) in TD. If the matcher takes into consideration the fact that p_i is an inverse functional property, then it should produce a match between instances u_i and u'_i (rule PRP-IFP, Table 3.2).

In addition to the above, we defined tests, shown in Table 4.10, for *complex class expressions* using constructs `owl:unionOf` and `owl:intersectionOf`. Consider test **LTUNIONOF**: for a source object u_i instance of class C, we create a target object u'_i, instance of class C'. Suppose that C' is defined as a union of a set of classes $C, C_1, \ldots C_k$. According to OWL semantics, `owl:unionOf` can be expressed in terms of `rdfs:subClassOf` (rule SCM-UNI, Table 3.4), hence the problem can be reduced to the case of `rdfs:subClassOf`. u'_i is obtained from u_i by copying the modified (or not) properties of the former. Therefore, u_i and u'_i have the same properties.

	SD	TD	SCHEMA TRIPLES	GS
LTUNIONOF	$(u_1, \texttt{rdf:type}, C)$	$(u'_1, \texttt{rdf:type}, C')$	$(C', \texttt{owl:unionOf}, \{C, C_1, \ldots)$	$u_1 \sim u'_1$
LTINTERSECT1	$(u_1, \texttt{rdf:type}, C)$	$(u'_1, \texttt{rdf:type}, C')$	$(C, \texttt{owl:intersectionOf}, S)$ $(C', \texttt{owl:intersectionOf}, S)$	$u_1 \sim u'_1$
LTINTERSECT2	$(u_1, \texttt{rdf:type}, C)$	$(u'_1, \texttt{rdf:type}, C')$	$(C, \texttt{owl:intersectionOf}, S)$ $(C', \texttt{owl:intersectionOf}, S')$ $S' \subset S$	$u_1 \sim u'_1$

Table 4.10: Tests for `owl:unionOf`, `owl:intersectionOf`

The same principle holds for the `owl:intersectionOf` construct (tests **LTINTERSECT1**, **LTINTERSECT2**); the rules considered for those transformations are SCM-INT (Table 3.4) and CLS-INT1, CLS-INT2 (Table 3.3).

Examples of the modifications presented before are shown in Tables 4.11,4.12, 4.13, 4.14, 4.15 and 4.16.

MODIFICATION LTSUBC	
SD	cwork:id1 rdf:type cwork:NewsItem .
TD	cwork:id1234 rdf:type cwork:CreativeWork .
SCHEMA TRIPLES	cwork:NewsItem rdfs:subClassOf cwork:CreativeWork .
GOLD STANDARD	cwork:id1 ∼ cwork:id1234
MODIFICATION LTEQC	
SD	dbpedia:Harry_Foreman rdf:type foaf:Person .
TD	core:Harry_Foreman rdf:type core:Person .
SCHEMA TRIPLES	foaf:Person owl:equivalentClass core:Person .
GOLD STANDARD	dbpedia:Harry_Foreman ∼core:Harry_Foreman

Table 4.11: Examples for `rdfs:subClassOf`, `owl:equivalentClass`

MODIFICATION LTEQP	
SD	dbpedia:Harry_Foreman rdf:type foaf:Person . dbpedia:Harry_Foreman foaf:name "Harry Foreman" .
TD	dbpedia:Harry_Foreman rdf:type foaf:Person . dbpedia:Harry_Foreman core:name "Harry Foreman" .
SCHEMA TRIPLES	foaf:name owl:equivalentProperty core:name .
GOLD STANDARD	dbpedia:Harry_Foreman (in SD) ~ dbpedia:Harry_Foreman (in TD)
MODIFICATION LTSUBP	
SD	cwork:id1 cwork:about dbpedia:Andrew_Tyrie .
TD	cwork:id231 cwork:tag dbpedia:Andrew_Tyrie .
SCHEMA TRIPLES	cwork:about rdfs:subPropertyOf cwork:tag .
GOLD STANDARD	cwork:id1 (in SD) ~ cwork:id231 (in TD)

Table 4.12: Examples for `rdfs:subPropertyOf`, `owl:equivalentProperty`

MODIFICATION LTSAMEAS1	
SD	cwork:id1 rdf:type cwork:BlogPost . cwork:id2 rdf:type cwork:BlogPost .
TD	cwork:id128 rdf:type cwork:BlogPost . cwork:id457 rdf:type cwork:BlogPost . cwork:id128 owl:sameAs cwork:id457 .
SCHEMA TRIPLES	none
GOLD STANDARD	cwork:id1 ∼ cwork:id128 cwork:id1 ∼ cwork:id457 cwork:id2 ∼ cwork:id457 cwork:id2 ∼ cwork:id128
MODIFICATION LTSAMEAS2	
SD	cwork:id1 rdf:type cwork:Programme .
TD	cwork:id80 rdf:type cwork:Programme . cwork:id537 rdf:type cwork:Programme . cwork:id80 owl:sameAs cwork:id537 . (not necessary)
SCHEMA TRIPLES	none
GOLD STANDARD	cwork:id1 ∼ cwork:id80 cwork:id1 ∼ cwork:id537
MODIFICATION LTDIFF	
SD	cwork:id1 rdf:type cwork:NewsItem .
TD	cwork:id50 rdf:type cwork:NewsItem . cwork:id286 rdf:type cwork:NewsItem . cwork:id50 owl:differentFrom cwork:id286 .
SCHEMA TRIPLES	none
GOLD STANDARD	cwork:id1 ∼ cwork:id50

Table 4.13: Examples for `owl:sameAs`, `owl:differentFrom`

MODIFICATION LTDISJC	
SD	cwork:id1 rdf:type cwork:NewsItem .
TD	cwork:id1493 rdf:type cwork:BlogPost .
SCHEMA TRIPLES	cwork:NewsItem owl:disjointWith cwork:BlogPost .
GOLD STANDARD	
MODIFICATION LTDISJP	
SD	cwork:id1 core:facebook ''Desmond Swayne''
TD	cwork:id123 core:twitter''Desmond Swayne'' .
SCHEMA TRIPLES	core:facebook owl:propertyDisjointWith core:twitter .
GOLD STANDARD	

Table 4.14: Examples for `owl:disjointWith`, `owl:propertyDisjointWith`

modification ltFuncP	
SD	cwork:id1122 core:primaryTopic dbpedia:Andrew_Tyrie .
TD	cwork:id1122 core:primaryTopic ldbc:Andrew_Tyrie .
SCHEMA TRIPLES	core:primaryTopic rdf:type owl:FunctionalProperty .
GOLD STANDARD	dbpedia:Andrew_Tyrie ∼ ldbc:Andrew_Tyrie .
modification ltInvFuncP	
SD	cwork:id1 bbc:primaryContentOf cwork:id100 .
TD	cwork:id100 bbc:primaryContentOf cwork:id1123 .
SCHEMA TRIPLES	bbc:primaryContentOf rdf:type owl:InverseFunctionalProperty
GOLD STANDARD	cwork:id1 ∼ cwork:id123

Table 4.15: Examples for `owl:FunctionalProperty`, `owl:InverseFunctional-Property`

modification ltUnionOf	
SD	foaf:Andrew_Tyrie rdf:type foaf:Person .
TD	ldbc:Andrew_Tyrie rdf:type ldbc:Thing .
SCHEMA TRIPLES	ldbc:Thing owl:unionOf (foaf:Person dbpedia:Event dbpedia:Organization dbpedia:Place dbpedia:Theme)
GOLD STANDARD	foaf:Andrew_Tyrie ∼ ldbc:Andrew_Tyrie
modification ltIntersect1	
SD	dbpedia:William_McWilliams rdf:type ldbc:Person_Organisation .
TD	ldbc:William_McWilliams rdf:type ldbc:Individual_Corporation .
SCHEMA TRIPLES	ldbc:Person_Organisation owl:intersectionOf (foaf:Person dbpedia:Organisation) . ldbc:Individual_Corporation owl:intersectionOf (foaf:Person dbpedia:Organisation) .
GOLD STANDARD	dbpedia:Williams_McWilliams ∼ ldbc:Williams_McWilliams
modification ltIntersect2	
SD	core:id1 rdf:type ldbc:Event_Place_Theme .
TD	ldbc:id1 rdf:type ldbc:Happening_Spot .
SCHEMA TRIPLES	ldbc:Event_Place_Theme owl:intersectionOf (dbpedia:Event dbpedia:Place dbpedia:Theme) ldbc:Happening_Spot owl:intersectionOf (dbpedia:Event dbpedia:Place) .
GOLD STANDARD	core:id1 ∼ ldbc:id1

Table 4.16: Tests for `owl:unionOf`, `owl:intersectionOf`

4.3.4 Simple and Complex Transformations

In SPIMBench we consider combinations of the aforementioned transformations, that we call *simple transformations* to apply to different triples pertaining to one creative work. For instance, we can perform a value transformation on triple (s, p, o) where p is a data type property and a structural transformation on (s, p', o') (triple deletion). We also consider *complex transformations* that are combinations of the aforementioned ones that are applied to a *single* triple. For instance, when logical transformations are considered, then for a triple (s, p, o) we can produce a triple (s', p', o') where p is a subproperty of p' and o' is obtained by applying a lexical transformation on o. We focus on complex transformations (for the same triple) that combine lexical with logical or structural modifications as those were presented above, but not combinations of structural with logical modifications. The reason is that in schema of SPIMBench we do not have the meaningful properties for this kind of transformations. Nevertheless, our general framework does not forbid one to perform this kind of transformations.

LTSUBC AND PROPERTY AGGREGATION	
SD	dbpedia:Wendy_Crewson rdf:type foaf:Person .
	dbpedia:Wendy_Crewson foaf:givenName Wendy .
	dbpedia:Wendy_Crewson foaf:surname Crewson .
TD	core:Wendy_Crewson rdf:type core:Thing .
	core:Wendy_Crewson core:fullName Wendy_Crewson .
SCHEMA TRIPLES	foaf:Person rdfs:subClassOf core:Thing .
GOLD STANDARD	dbpedia:Wendy_Crewson ∼core:Wendy_Crewson

Table 4.17: Example for rdfs:subClassOf, and property aggregation

4.4 Data Generator

The generator of SPIMBench extends the one proposed by the Semantic Publishing Benchmark SPB [32]. The SPB data generator produces RDF descriptions of *creative works* that are valid instances of the BBC ontologies presented in Section 4.1. As discussed before, a creative work is described by a number of *data value prop-*

erties such as *title*, *description*, and *object value properties* such as *primaryTopicOf* and *primaryContent* among others, that refer to other resources. Recall that creative works have also properties that link them to resources defined in *reference* datasets: those are the *about* and *mentions* properties, and their values can be *person names*, *locations* and *events*. In this way, a creative work is linked to one or more resources. One of the purposes of the data generator is to produce large synthetic (in the order of billions of triples) datasets in order to check the ability of the engines to *scale*. The synthetic data generation is done in such a way that guarantees that the *distributions* used, emulate the real datasets provided by BBC. The SPB data generator [32] models three types of relations in the data:

- CLUSTERING OF DATA The clustering effect is produced by generating *creative works* about a *single entity from reference* datasets and for a *fixed period of time*. More precisely, the number of creative works starts with a high peak at the beginning of the chosen clustering period and follows a smooth decay towards its end. The data generator produces sets of creative works of different sizes: by default five major and one hundred smaller sets of creative works are produced for one year period.
- CORRELATIONS OF ENTITIES This correlation effect is produced by generating *creative works about two or three entities from the reference datasets* in a *fixed period of time*. Concretely, each of the entities is tagged by creative works *solely* at the beginning and the end of the specified correlation period whereas in the middle of the period, both entities are used as tags for the same creative work. As in the case of data clustering, the data generator models by default, fifty correlations between entities for one year period.
- RANDOM TAGGING OF ENTITIES Random data distributions are created with a *bias towards popular entities* created when the *tagging* is performed (when values are assigned to *about* and *mentions* creative work properties). This is achieved by *randomly* selecting a 5% of all the resources from reference data and mark them as *popular* when the remaining ones are marked as *regular*. When creating creative works, 30% percent of them are tagged with *randomly selected* popular resources and the remaining 70% are linked to the *regular*

ones. In addition to values for *mentions* and *about* properties, the values for data and object properties are *randomly generated*. More specifically, data value properties, such as creative work's *title*, and *description* are created randomly from DBpedia text. Creative works properties related to their *creation* and *modification* date are randomly created within a range of one year specified by a fixed seed year value. The classification of a creative work as a *blog post*, *news item* or *programme* follows user defined distributions. The value of property *audience* depends on the type of creative work; the same rationale is followed for properties *primaryFormat* whereas the value for property *thumbnail* is a randomly generated URI (same for property *primaryContentOf*).

The SPB data generator operates in a sequence of phases:

1. ontologies and reference datasets are *loaded* in an RDF repository
2. all instances of the domain ontologies that exist in the reference datasets are retrieved by means of predefined SPARQL queries that will be used as values for the *about* and *mentions* properties of creative works
3. from the previous set of instances, the *popular* and *regular* entities are selected
4. the generator produces the creative works according to the three principles discussed previously

SPIMBench extends the data generation process of SPB as follows:

- during the creation of *source instances*, that is instances of creative works, we produce a set of creative works, i.e., *target instances* by applying the transformations discussed in Section 4.3.
- the *(i)* percentage of *source* instances (over the total number of produced instances) to be transformed to obtain the *target* instances and *(ii)* the percentage of the different kinds of transformations (value, structural, logical) to be applied on the source instances are specified by the user. A large percentage of instances to be transformed and a large percentage of the types of transformations to be applied will produce a very diverse target dataset hence, testing the ability of the matcher to work with highly heterogeneous datasets;

in addition, having user defined parameters allows one to produce datasets that accommodate different instance matching scenarios. For instance, one could produce target datasets that use only value, lexical or logical transformations or a mix thereof; depending on the defined percentages one could emphasize on a specific kind of transformations and downplay on others.

- the creative works to be transformed are chosen *randomly* making sure that the percentage of instances to be transformed is respected.
- to determine the transformation to be applied on a source instance we are using a random generator that will produce a random double value in the range of $[0; 1]$. If we need to perform $x\%$, $y\%$ and $z\%$ of value, structural and logical transformations on the chosen instances respectively, then if the random value obtained is between $[0; 0.x]$ then a value transformation is applied; if the value is between $[0.x; 0.x + 0.y]$ a structural one is performed. Similarly for a logical one. JAVA's random number generator guarantees an even distribution of all generated values in that range. So there is no round down or up for the produced values.
- the percentage of the *kind of a specific transformation* to be performed is also *user-specified*; recall that SPIMBench supports 8 different types of value, 3 types of structural and 12 types of logical transformations. At the current version of the SPIMBench data generator the properties of creative work instances to which value and structural transformations are applied, are explicitly specified in the generator. This is the case with the logical transformations in order to make sure that those performed are meaningful: for instance, to test disjointness of classes, the data generator will produce instances of classes that are defined as disjoint. The same rationale for user-specified percentages for the number of instances to be transformed applies for here.
- to accommodate logical transformations the SPIMBench data generator produces instances of the classes/properties for which we have specified in the extended SPIMBench schema the appropriate constraints (`owl:unionOf`, `owl:intersectionOf`, `owl:disjointWith`, etc.). We have to mention here

that the modifications that we perform on instances we obtain from external ontologies namely FOAF, DBpedia and Travel are not applied on existing properties and values thereof, but on properties that are introduced to support the modifications (especially the logical ones) covered by SPIMBench.

- for each pair of *(source, target)* instances produced, one *match* triple and one *transformation* triple per employed transformation (see Figure 4.6) that records the transformation type, the property on which the transformation is applied are added in the gold standard. Once the gold standard is produced, we determine the weight for each pair of *(source, target)* instances as discussed in Section 4.5.

4.5 Gold Standard

The gold standard that we propose in SPIMBench records for each pair of instances u and u' the set of transformations applied for obtaining the latter from the former. Figure 4.5 shows the schema that we are using to represent the instances that we consider as matches. For each pair of source instance u and its transformed target instance u', we keep an object, instance of class `spimbench:Match`, that stores the instances (using properties `spimbench:source` and `spimbench:target` respectively) and the *transformation* (property `spimbench:transformation` that takes its values in class `spimbench:Transformation`) applied to transform the former to the latter; we also store a *weight* that records the *information loss* obtained by a transformation applied on a source instance u to obtain the target instance u'.
`spimbench:Transformation` is a super-class of `spimbench:ValueTransf`, `spimbench:StructuralTransf` and `spimbench:LogicalTranf` for the value/lexical, structural and logical transformations supported in our framework. The specific kinds of *value* VTI, *structural* STJ and *logical* LTK transformations are defined as *subclasses* of their respective classes. In addition to the above information we also store the schema property or predicate on which the specific transformation has been applied using property `spimbench:onProperty`. The weight is recorded using property `spimbench:weight`. To capture the fact that a transformation is a complex or simple one, then it is instantiated in *multiple* transformation classes. This

detailed gold standard can be used by instance matching systems for *debugging* purposes. An instance of the gold standard for a value transformation is shown in Figure 4.6. In the future we will explore the possibility of representing the proposed Gold Standard ontology in terms of the OAEI EDOAL ontology alignment language[7] and PROV-O[8] provenance ontology.

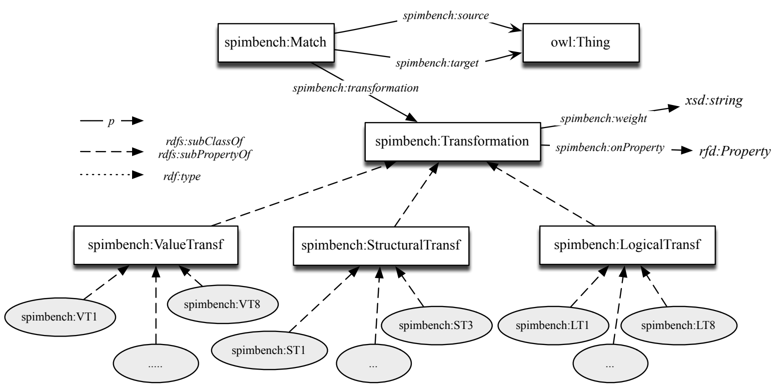


Figure 4.5: Gold Standard Ontology

```
@prefix spimbench: <http://www.spimbench/trans/>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

spimbench:match0 rdf:type spimbench:Match .
spimbench:match0 spimbench:source <http://www.bbc.co.uk/things/1#id> .
spimbench:match0 spimbench:target <http://www.bbc.co.uk/things/859547602#id> .
spimbench:match0 spimbench:transf spimbench:transf14 .

spimbench:transf14 rdf:type spimbench:VT1.
spimbench:transf14 spimbench:onProperty cwork:CW_id .
spimbench:transf14 spimbench:weight 0.86
```

Figure 4.6: Example: Gold Standard Instance

Computing the weights

We discuss in this section the two solutions we designed and implemented for computing the weight w for a pair of source and target instances u and u' respectively of the gold standard. Let $\{\mathsf{sd}_1, \mathsf{sd}_2, \ldots, \mathsf{sd}_\mathsf{n}\}$ be a partition of the source dataset

[7]http://alignapi.gforge.inria.fr/edoal.html
[8]http://www.w3.org/TR/prov-o/

sd where each $\mathsf{sd_i}$, $i = 1, \ldots n$ contains the same number of triples. Let td be the target dataset where for $\{\mathsf{td_1}, \mathsf{td_2}, \ldots, \mathsf{td_n}\}$ each $\mathsf{td_i}$ is the target dataset for $\mathsf{sd_i}$. Finally, let gs be the gold standard and $\mathsf{gs_i}$ the part of the gold standard (i.e., sets of triples that are valid instances of the schema presented in Figure 4.5 for the pair $(\mathsf{sd_i}, \mathsf{td_i})$, $i = 1, \ldots, n$. For both solutions we use the RESCAL [33, 34] tool that actually computes the information loss between two instances u and u'.

RESCAL is a tensor factorization for large-scale relational learning from Linked Data, multi-relational data and large multigraphs. It offers state-of-the-art relational learning results combined with high scalability, such that it can be applied to data consisting of millions of entities, hundreds of relations, and billions of known facts. Due to its latent variable structure, RESCAL does not require deep domain knowledge and therefore can be easily applied to most domains.

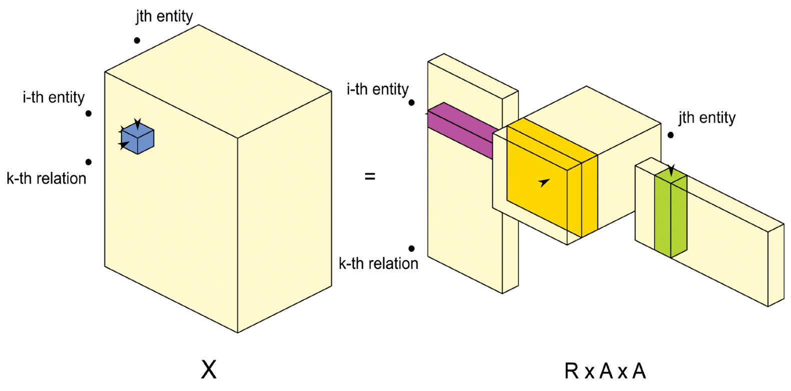


Figure 4.7: RESCAL factorization model

In the first, naive solution, we run through all pairs of instances u, u' in each source and target file $\mathsf{sd_k}$, $\mathsf{td_k}$ and compute the information loss using the RESCAL approach. It is evident that for a large number of source files that would be non-scalable and actually our experiments showed that we need 6 minutes for RESCAL to compute the score for 10^4 triples. The time increases linearly, hence we would need approximately 6×10^2 minutes for 10^6 triples.

Given that computing the score for each pair of instances in the gold standard is a process that requires a large number of computations and consequently renders the computation of the weights non scalable for large datasets (in the order of millions or billions of triples). We propose a second solution based on *sampling* the pairs of source and target data and using these samples we consider the score for each of applied transformations. We provide below a more detailed discussion.

In this approach, we would like to compute the weight per *type of transformation* and not per *transformation*. More specifically, let $\mathsf{T}^1, \mathsf{T}^2, \dots \mathsf{T}^{\mathsf{m}}$ be the types of transformations supported by our framework. We store, for each pair $(\mathsf{sd_i}, \mathsf{td_i})$ using their gold standard $\mathsf{gs_i}$, a vector $\mathsf{F_i}$ that contains the number of transformations of the same type that were employed on $\mathsf{sd_i}$ to obtain $\mathsf{td_i}$: $\mathsf{F_i} =<|| \mathsf{T}^1_{\mathsf{i}} ||, \dots || \mathsf{T}^{\mathsf{m}}_{\mathsf{i}} ||>$. In a vector $\mathsf{F_i}$, if a transformation $\mathsf{T}^{\mathsf{k}}_{\mathsf{i}}$ is not employed to obtain target instances in $\mathsf{td_i}$, then the value in position k is zero.

$$\text{score}(\mathsf{T}^{\mathsf{i}}) = \frac{\sum_{j=1}^{||\mathsf{gs}||} || \mathsf{F}^{\mathsf{i}}_{\mathsf{j}} ||}{|| \mathsf{gs} ||}$$

Vector E is the vector that contains the average number of appearances of each transformation type, that is $\mathsf{E} =< \mathsf{score}(\mathsf{T}^1), \mathsf{score}(\mathsf{T}^2), \dots, \mathsf{score}(\mathsf{T}^{\mathsf{m}}) >$. Once E is obtained, we compute the squared cosine of E and each $\mathsf{F_i}$: $\cos^2(\mathsf{F_i}, \mathsf{E})$. We obtain then the λ files with the F_k's with the smallest cosine, that is the F_k's that include the highest number of transformations.

We calculate λ using Jensen–Shannon divergence [35]. Jensen–Shannon divergence is a popular method of measuring the similarity between two probability distributions. It is based on the Kullback–Leibler divergence [36], with some notable (and useful) differences, including that it is symmetric and it is always a finite value.

$$D_{\text{JS}} = \tfrac{1}{2} D_{\text{KL}} \left(P \| M \right) + \tfrac{1}{2} D_{\text{KL}} \left(Q \| M \right)$$

where M is the average of the two distributions,

$$M = \tfrac{1}{2}(P + Q)$$

In our case, Q is the probability distribution of the transformation types in all files and P the probability distribution of λ first files with the best distribution. If the value of λ chosen by the user is not satisfactory, we recommend the use of the λ value calculated with the assistance of Jensen–Shannon divergence.

Given the top-λ pairs of source and target instances ($\mathsf{sd_i}$, $\mathsf{td_i}$), as well as their respective $\mathsf{gs_i}$, we give those files as input to RESCAL that returns a matrix A for every URI we are interested in. Then we calculate the cosine similarity between the matrices of every pair of the URIs we consider as a match. Those cosines correspond to a score of the combination $\mathbb{C} = \{\mathsf{T}^1, \mathsf{T}^2, \dots \mathsf{T}^j\}$ of the transformations the URIs have undergone.

$$\mathsf{score}(\mathbb{C}) = \mathsf{score}(\mathsf{T}^1) \times \mathsf{score}(\mathsf{T}^2) \times \dots \mathsf{score}(\mathsf{T}^j)$$

Given the score for each combination $\mathbb{C}$ we can obtain the score for each individual transformation using linear regression [37].

Linear regression is an approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables denoted X. Given a data set $\{y_i,\, x_{i1}, \ldots, x_{ip}\}_{i=1}^n$ of n statistical units, a linear regression model assumes that the relationship between the dependent variable y_i and the p-vector of regressors x_i is linear. This relationship is modeled through a disturbance term or error variable ε_i — an unobserved random variable that adds noise to the linear relationship between the dependent variable and regressors. Thus the model takes the form

$$y_i = \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\rm T}\boldsymbol\beta + \varepsilon_i, \qquad i = 1, \ldots, n,$$

where T denotes the transpose, so that $\mathbf{x}_i^{\rm T}\boldsymbol\beta$ is the inner product between vectors $\mathbf{x}_i$ and $beta$.

Often these n equations are stacked together and written in a vector form as

$$\mathbf{y} = \mathbf{X}\boldsymbol\beta + \boldsymbol\varepsilon,$$

where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} \mathbf{x}_1^{\mathrm{T}} \\ \mathbf{x}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{x}_n^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

After all, for every pair of URIs, we sum the weights of the transformations applied to the URI from the source dataset and calculate the final weight as follows:

$$finalWeight = 1 - \sum_{i=0}^{i=n} (\mathsf{T_0}, \cdots, \mathsf{T_i})$$

We have to note here that if the final weight is greater than 1.0 we consider it as 1.0 apart from `owl:disjointWith`, and `owl:propertyDisjointWith`that we consider equal to 0.0.

4.6 Evaluation

In this Section we are going to discuss the implementation of SPIMBench (Section 4.6.1), also to show the experiments that we conducted for testing the ability of the SPIMBench generator to scale (Section 4.6.2). Our study on *weight distribution*(Section 4.6.3) intends to show the difficulty of each of the proposed transformations of SPIMBench. We also ran experiments to test the capacity of instance matching systems to address the challenges set by the datasets produced by SPIMBench and the capability of the instance matching systems to find the correct matches as those are reported by the gold standard (Section 4.6.4).

4.6.1 Implementation

The Semantic Publishing Benchmark (SPB) is developed in the context of LDBC (`http://www.ldbc.eu`) for testing the performance of RDF engines inspired by the Media/Publishing industry. The benchmark offers a data generator that uses real reference data to produce datasets of various sizes and tests the scalability aspect of RDF systems. In addition, it the benchmark's workload consists of (a) editorial operations that add new data, alter or delete existing data (b) aggregation operations that retrieve content according to various criteria.

In SPIMBench[9] we used and extended SPB's data generator to produce the *target* dataset to be used along with the source data as input to an instance matching tool. The datasets are expressed in RDF `Turtle` format[10] and are produced in parallel. Different sizes of datasets allows us to test the ability of the instance matching systems to scale.

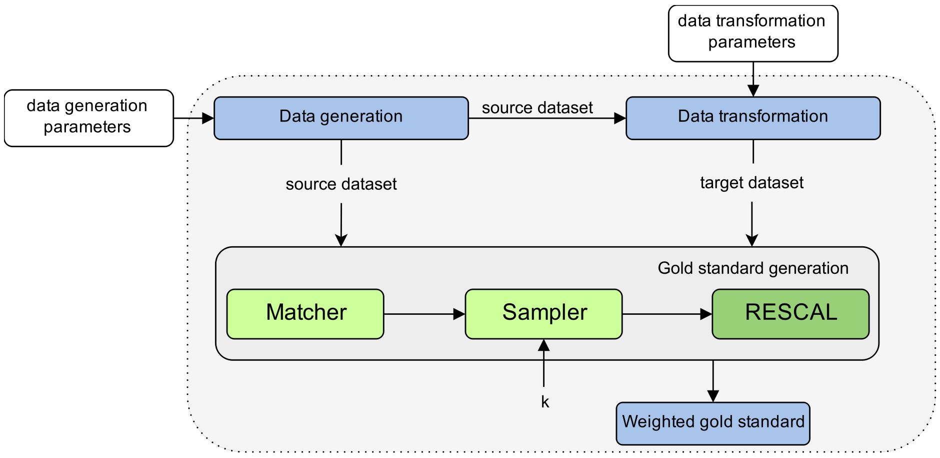


Figure 4.8: SPIMBenchModel

We discuss SPIMBench 's implementation in Figure 4.8. SPIMBench takes as input a configuration file that contains data transformation and data generation parameters. Guided by the configuration file *source*, *target* data and an intermediate *gold standard* are produced.

The most complex task is the generation of the final *weighted gold standard*. The weights are calculated through the RESCAL factorization model. In order to be as scalable as possible SPIMBench uses a sampling technique (Jensen–Shannon divergence) and keeps only the *target* data to which an important number of transformations has been applied. Each selected *target* data along with the *source* data are given as input to RESCAL. The result is a *detailed weighted gold standard* file

[9]The code of SPIMBench is available at `https://github.com/jsaveta/SPIMBench`
[10]`http://www.w3.org/TR/turtle/`

that records for each pair of (source, target) instances an entry that stores (a) the type of transformation applied, (b) the property on which it is applied and (c) the weight for the individual transformation. There is also a simpler *weighted gold standard* that records for each pair of (source, target) instances the weight of the distance between them.

4.6.2 Scalability

We ran all experiments on a two eight-core Intel Xeon CPUs (2.30GHz) with 384GB of main memory using Debian Wheezy on a Linux Kernel 3.12.3.

In order to test the performance of SPIMBench regarding its ability to produce large synthetic datasets to be used by instance matching systems, we produced datasets of 10K, 50K, 100K, 1M, 10M, 100M, 250M and 500M triples using the benchmark's data generator. In the experiments, we considered that 5%, 10%, 15% and 25% of the produced source triples related to *creative works* were transformed.

Data set	I1	I2 (15%)	I3 (25%)
10K	400	60	100
50K	2K	300	500
100K	4K	600	1K
1M	40K	6K	10K
100M	4M	600K	1M
250M	10M	1.5M	2.5M
500M	20M	3M	5M

Table 4.18: Data sets and the instances involved in transformations

Table 4.18 shows the number of creative work instances(column I1) for each of the produced data sets. Columns I2, I3 give the number of creative work instances that are involved in the transformations when 15% and 25% respectively of the instances in I1 are considered. We show just 15% and 25% indicatively.

For our experiments, all the transformations that implement lexical, structural and logical test cases are used with the same frequency.

We chose this high percentage(25%) (compared to other state-of-the-art works) to show that data generation scales even when the amount of work necessary to produce the target data set is large. Note that these percentages merely indicate an upper bound of the actual percentage of test cases observed in the target data. Indeed, for *lexical* transformations with a given probability, the triple selected randomly to be transformed may actually not support a given value transformation (e.g., change date format can only be applied to dates but not to numbers, selected property is not an object property but a data type property as discussed in Section 4.1).

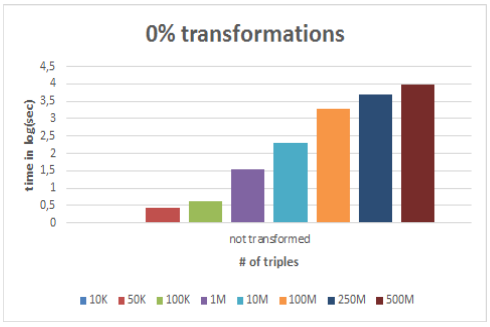
A similar observation can be made for *structural* transformations, e.g., because not all properties can be aggregated. In the generated test dataset, we observe that the probability of actually obtaining a lexical transformation is approximately 0.87, whereas the probability of a structural transformation is 0.82. For *logical* transformations, this probability decreases to 0.42, explained by the fact that there is a limited number of cases where meaningful logical transformations can be produced (see Section 4.3). For *complex* combination of transformations we observe a probability of 0.6 whereas the probability in the case of a *simple* combination of transformations is related to the probabilities of the individual transformations is defined by. In the sequel, we refer to these probabilities as *success probabilities* (*sp*). In summary, when we will refer to 25% of transformations of a given type, this means that 25% of our instances are selected for transformation. Each triple of such an instance has a probability *sp* to be considered by a transformation.

LEX	STR	LOG	SIM	COM
0.87	0.82	0.42	$x \times 0.87 + y \times 0.82 + z \times 0.42$	0.6

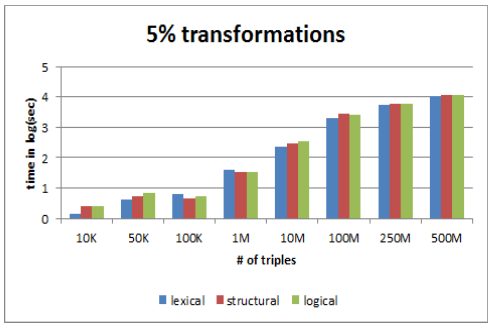
Table 4.19: Success Probabilities

Table 4.19 shows the success probabilities for each of the lexical (LEX), structural (STR), logical (LOG) transformations, simple combination (SIM), and complex combination(COM)of transformations. In the case of simple combination, x, y, and z correspond to the probability of selecting a triple for LEX, STR and LOG transformations.

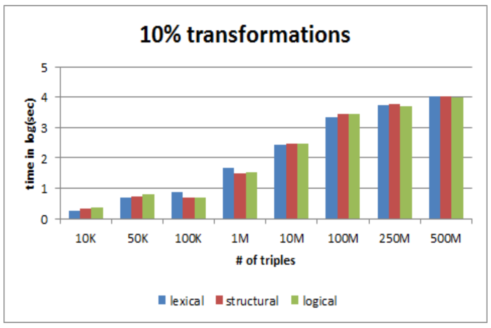
Figures 4.9 and 4.10 show the time required to produce the target datasets for the input source datasets and for the aforementioned configurations. More specifically Figure 4.9 shows the time in seconds required to perform value, structural and logical transformations for 5%, 10%, 15% and 25% transformation percentages.



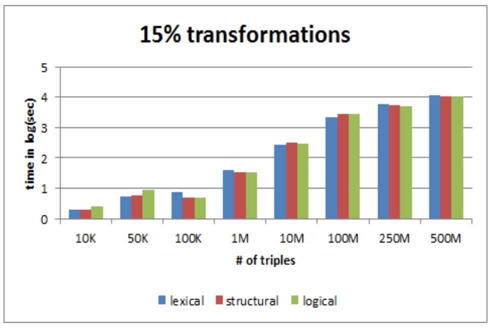
(a) 0% Transformations



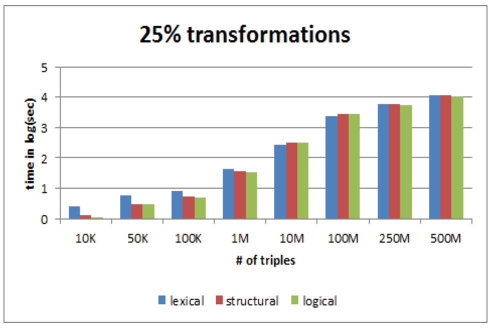
(b) 5% Transformations



(c) 10% Transformations



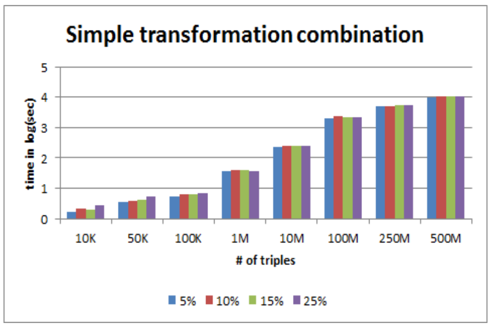
(d) 15% Transformations



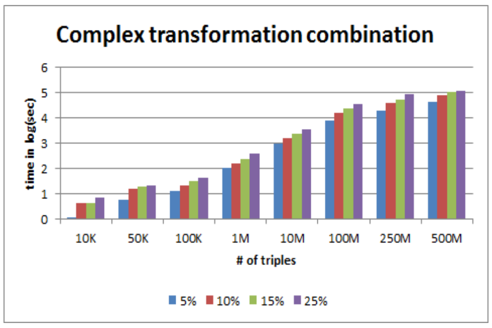
(e) 25% Transformations

Figure 4.9: Scalability results for the SPIMBench Data Generator

Figure 4.10 shows the time needed to compute the target instances when only complex and simple transformations are considered. We can observe that the transformations *do not* introduce any overhead (except in the case of complex ones) when compared to the Figure 4.9(a) that shows the time required to produce the source datasets with zero transformations. Note though that the time needed to perform complex transformations is one order of magnitude higher than the time needed to perform the simple ones. Nevertheless, we do not consider this to be an important problem since this process is done once and offline to produce the source and target datasets to be used by an instance matching system.



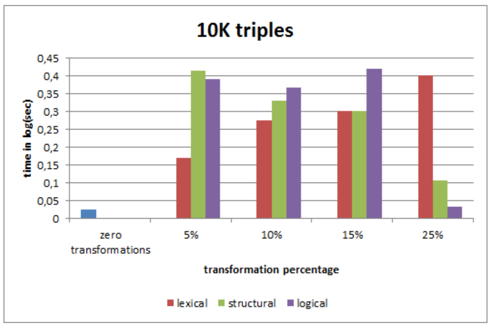
(a) Simple Transformations



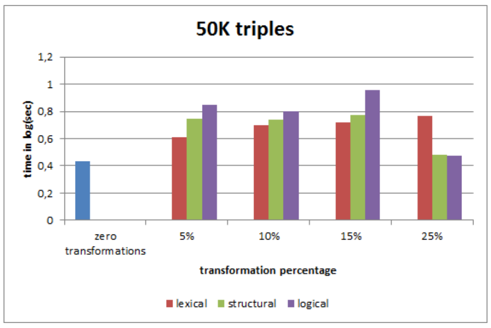
(b) Complex Transformations

Figure 4.10: Simple and Complex Transformations

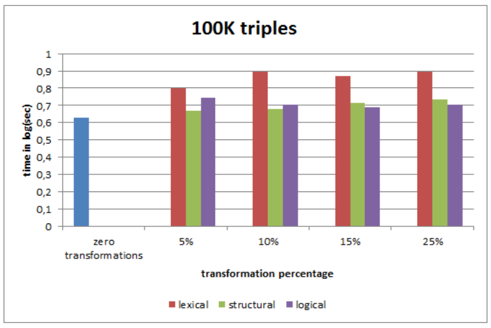
Figure 4.11 shows the time in seconds required to produce the target instances with 0%, 5%, 10%, 15% and 25% of *value*, *structural* and *logical* transformations for all different size of datasets.



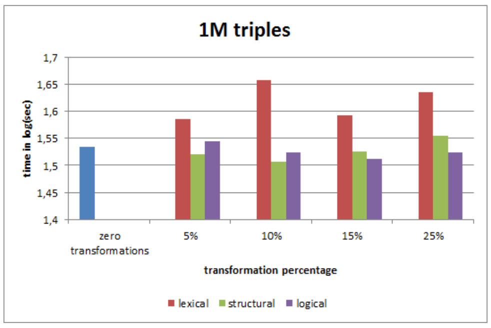
(a) 10K of triples



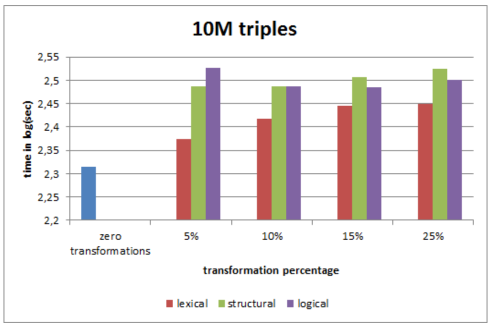
(b) 50K of triples



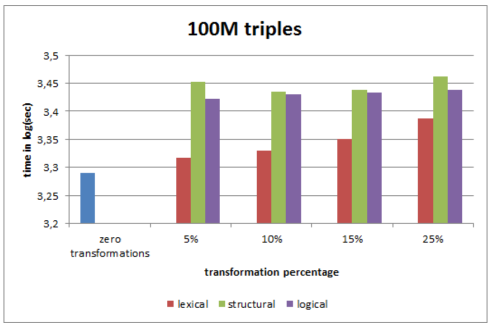
(c) 100K of triples



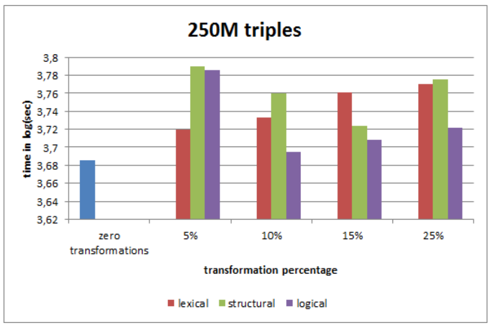
(d) 1M of triples



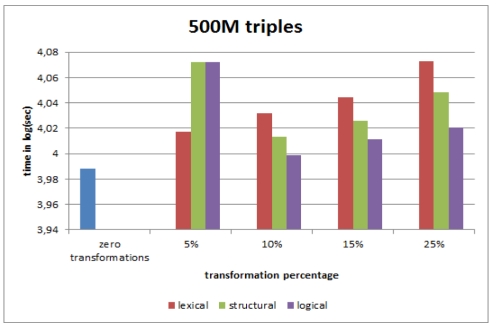
(e) 10M of triples



(f) 100M of triples



(g) 250M of triples



(h) 500M of triples

Figure 4.11: Scalability results for the SPIMBench Data Generator for different sizes of datasets

4.6.3 Weight Distribution

To determine how the *weights* are distributed for the different test cases, we generated 1M triples (approximately 40K creative work instances). Again, we produced all test cases for the 25% of source instances, and with the same configuration for the transformations (or test cases) as discussed previously in (Section 4.6.2).

The weight distribution was intended to represent the difficulty for each of the test cases. This is also confirmed by the results that we obtained from LogMap [38] (see Section 4.6.4). Figure 4.12 shows the average weight for different test cases.

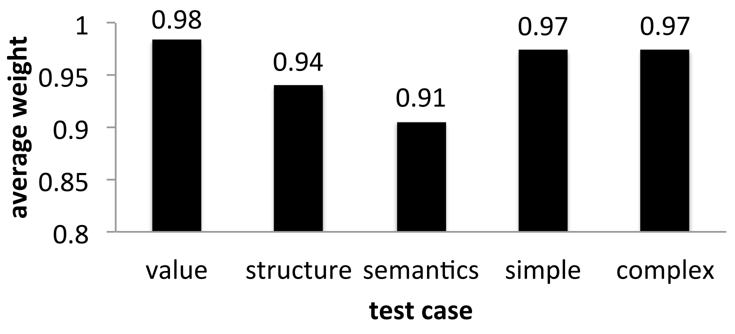


Figure 4.12: Average weights for different test cases

One can see that lexical(or value)transformations have a higher weight than the structural, and logical (or semantics-aware) ones. This means that they should be less challenging for instance matching systems. On the other hand, and as expected, logical transformations are the most difficult since those change the topology of the graph of the instance. Structural transformations also change the topology of the graph but these concern only properties.

For a more in-depth analysis of the weight distribution, and in particular to explain the high average weight for simple and complex combination of transformations, we further studied the distribution of weights in each transformation, as depicted in Figure 4.13. Clearly, high weights represent the majority of weights for lexical transformations, whereas more than half of the weights are in a lower range for logical transformations. Now looking at simple and complex combination of transformations, we observe that the majority of weights is again in the highest range. This distribution relates to the success probability *sp* we previously mentioned.

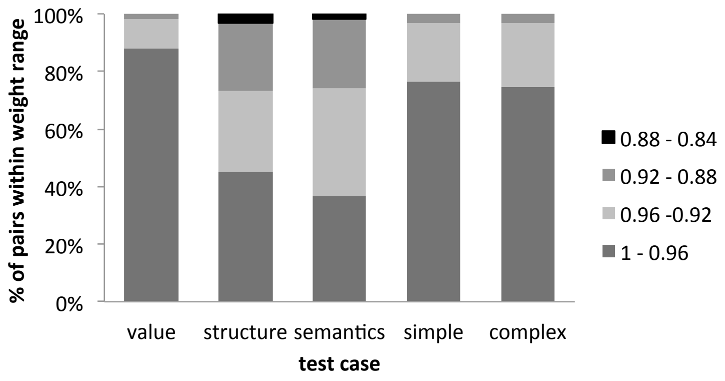


Figure 4.13: Weight distribution for different test cases

Indeed, for the simple combination case, we have seen that the success probability of a lexical transformation is 0.87 whereas the probability of obtaining a logical transformation is 0.42. So, overall, lexical transformations will dominate the weight distribution.

For the complex combination case, the weight is high because in our experiments we consider transformations only regarding the instance properties. These have a higher weight than the structural transformations since, as discussed earlier, the success probability *sp* of the latter is 0.82 while for the former is 0.60.

4.6.4 Performance of instance matching systems

We demonstrate the applicability of SPIMBench by using it to evaluate Logic-based and Scalable Ontology Matching (LogMap) [38] with different data set sizes and different test cases.LogMap is an ontology matching system with 'built-in' reasoning capabilities. LogMap can deal with semantically rich ontologies containing tens (and even hundreds) of thousands of classes. We ran the experiments using the MoRe [39] and HermiT [40] reasoners. Given an OWL file, HermiT can determine whether or not the ontology is consistent, identify subsumption relationships between classes and much more. While, MORe outputs the classification hierarchy entailed by the terminological part of the hierarchy.

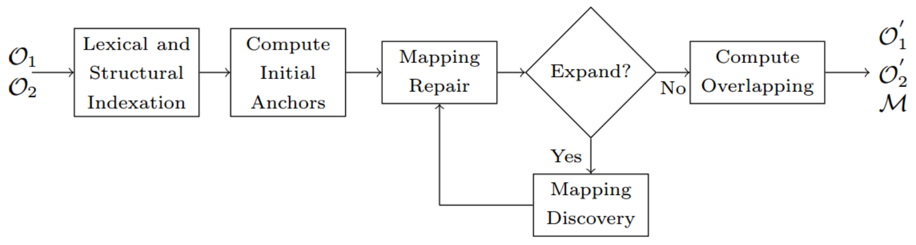


Figure 4.14: LogMap

The results and the runtimes were identical for the two reasoners so we present only HermiT's results. Once again, we used 25% of transformations for our experiments. We restricted the sizes of the input data sets to 10K, 25K and 50K triples. We chose to evaluate SPIMBench with LogMap as it is a continuously evolving system and has also demonstrated very good results.

In all cases, we measured recall, precision, and f-measure. Figures 4.15, 4.16, and 4.17 report our results for the different types of transformations and all the aforementioned datasets. Note that the results reported there concern only the instances that were used for the test cases and not all the instances that were generated.

The obtained results confirm the difficulty of the test cases as shown from the weight distribution. We notice that LogMap responds optimally regarding the precision as it does not find many matches that are not actually a match. On the

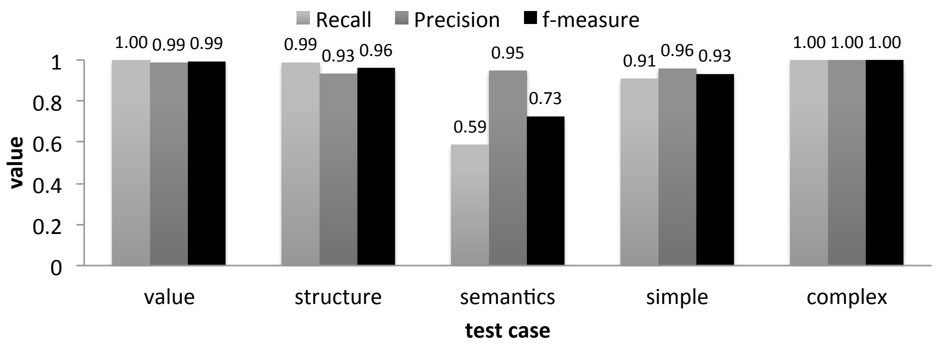


Figure 4.15: Recall, precision, and f-measure for test cases on 10K dataset

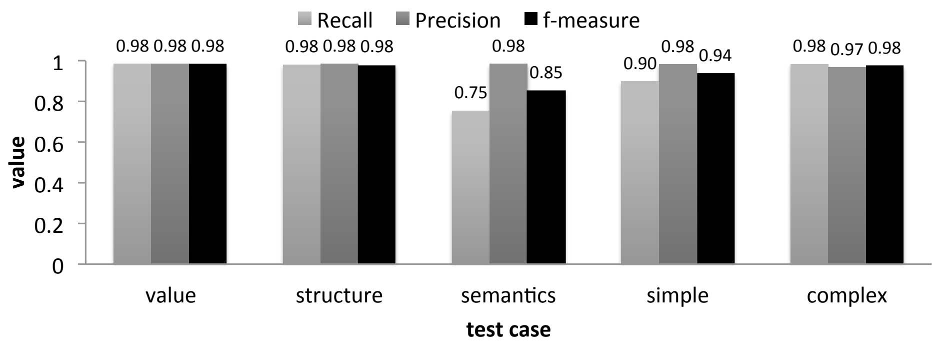


Figure 4.16: Recall, precision, and f-measure for test cases on 25K dataset

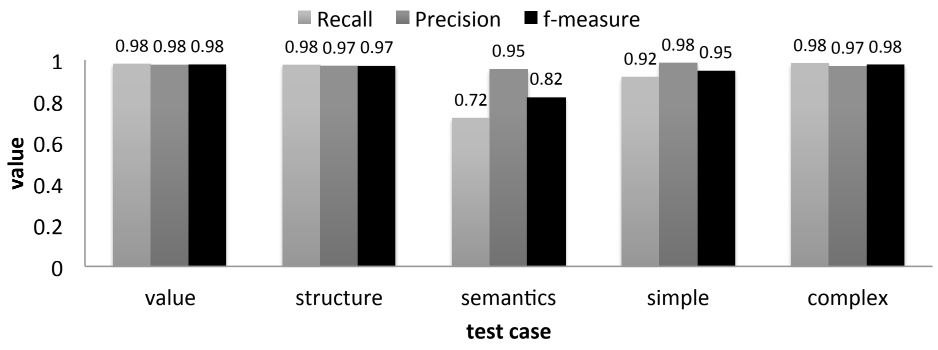


Figure 4.17: Recall, precision, and f-measure for test cases on 50K dataset

other hand, LogMap fails to find matches when the instance is involved in multiple semantics-aware test cases (hence, the lower recall for those).

Regarding runtime, LogMap needs approximately 100 seconds for the dataset size of 10K triples. What we observed was that the time needed for the matching was not increasing in a linear way. Sometimes 25K triples required more time than 50K triples. This inconsistency might be derived from the network overhead as we ran LogMap through the web application. The reason we did not choose to run it through the jar file that they provide was that we needed to define the properties that were going to be matched.

The second instance matching system we unsuccessfully tried to use for the evaluation task was Community Edition of Combining Matchers (COMA) 3.0 [41]. The reason for the failure of the experiment was that COMA focusses on matching schemas and ontologies. It has support for instance-based ontology matching but does not aim for instance matching per se.

In summary, we conclude that SPIMBench generates a variety of test cases that is well suited to identify strong points and weak points of state-of-the-art instance matching systems such as LogMap in terms of matching quality.

Chapter 5

Conclusions

In this thesis we presented the *Semantic Publishing Instance Matching Benchmark*, in short, SPIMBench, a benchmark inspired from the *Semantic Publishing Benchmark* SPB. SPIMBench, like SPB, is based on the BBC (`http://www.bbc.com/`) ontologies, which lie in the *Semantic Publishing* domain.

The differentiator of SPIMBench with the existing instance matching benchmarks is that it is, to the best of our knowledge, *the first benchmark proposing a data generator, a gold standard, and test cases that take into consideration expressive OWL constructs that go beyond the usual RDFS constructs.*

SPIMBench proposes and implements a *scalable data generator* that produces synthetic *source* and *target* data consistent with the extended SPIMBench schema to be used for testing the performance of instance matching systems; SPIMBench also proposes and implements a set of *transformations* on source data to obtain the target data. The set of transformations supported by SPIMBench includes *value* and *structural* ones as those have been proposed in a large number of representative instance matching benchmarks; and finally the *logical* ones that go beyond the standard RDFS constructs and include expressive OWL constructs, namely *instance (in)equality*, *equivalence* of classes and properties, *property constraints* and *complex class definitions.*

The SPIMBench data generator also produces a *weighted gold standard* that records for each pair of (source, target) instances an entry that stores (a) the type of transformation applied, (b) the property on which it is applied (in the case of structural and lexical transformations) and (c) the weight that records the distance between the two instances. The detailed gold standard can be used for debugging instance matching systems since we explicitly store the transformations applied to a source to obtain a target instance as well as their degree of similarity. In the future, we plan to define new metrics of precision and recall that take into account the computed weights.

The experimental evaluation showed that SPIMBench scales for a very large number of triples and modifications. In fact, our experiments showed that the generation of the target data and the gold standard does not introduce any additional processing overhead.

An objective for future research is to customize SPIMBench in order to be fully configurable and domain independent. Hence, the user will be able to use the domain of his choice rather than the predefined one. In addition, we will aim to use different schemas for source and target datasets. Last but not least,we plan to further pursue this idea by defining new precision/recall metrics that take into account weights.

Bibliography

[1] I. Bhattacharya and L. Getoor, *Entity resolution in graphs. Mining Graph Data.* Wiley and Sons, 2006.

[2] A. Elmagarmid, P. Ipeirotis, and V. Verykios, "Duplicate Record Detection: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 1, 2007.

[3] C. Li, L. Jin, and S. Mehrotra, "Supporting efficient record linkage for large data sets using mapping techniques," in *WWW*, 2006.

[4] J. Noessner, M. Niepert, C. Meilicke, and H. Stuckenschmidt, "Leveraging Terminological Structure for Object Reconciliation," in *ESWC*, 2010.

[5] M. Dean and G. Schreiber, "OWL Web Ontology Language Reference," http://www.w3.org/TR/owl-ref, 2004.

[6] F. Manola, E. Miller, and B. McBride, "RDF Primer," www.w3.org/TR/rdf-primer, February 2004.

[7] R. Isele, A. Jentzsch, and C. Bizer, "Silk Server - Adding missing Links while consuming Linked Data," in *COLD*, 2010.

[8] A.-C. N. Ngomo and S. Auer, "LIMES - A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data," *IJCAI*, 2011.

[9] M. Weis, F. Naumann, and F. Brosy, "A Duplicate Detection Benchmark for XML and Relational Data," in *IQIS*, 2006.

[10] J. Euzenat and et. al., “ Results of the Ontology Alignment Evaluation Initiative 2009,” in *OM*, 2009.

[11] “ISLAB, Instance Matching Benchmark,” http://islab.dico.unimi.it/iimb/.

[12] “Oaei instance matching,” http://oaei.ontologymatching.org/2010/, 2010.

[13] “Oaei instance matching,” http://www.instancematching.org/oaei/imei2011.html, 2011.

[14] B. C. Grau and et. al., “Results of the Ontology Alignment Evaluation Initiative,” in *OM*, 2013.

[15] K. Zaiss, S. Conrad, and S. Vater, “A Benchmark for Testing Instance-Based Ontology Matching Methods,” in *KMIS*, 2010.

[16] K. Zaiss, “Instance-Based Ontology Matching and the Evaluation of Matching Systems,” Ph.D. dissertation, Mathematisch-Naturwissenschaftlichen Fakultaet der Heinrich-Heine-Universitaet Dusseldorf, 2010.

[17] B. Alexe and et. al., “STBenchmark: Towards a benchmark for mapping systems,” in *PVLDB*, 2008.

[18] M. Neiling, S. Jurk, H.-J. Lenz, and F. Naumann, “Object identification quality,” in *DQCIS*, 2003.

[19] H. Köpcke, A. Thor, and E. Rahm, “Comparative evaluation of entity resolution approaches with FEVER,” in *VLDB*, 2009, demo Track.

[20] D. Brickley and R. Guha, “RDF Vocabulary Description Language 1.0: RDF Schema,” www.w3.org/TR/2004/REC-rdf-schema-20040210, 2004.

[21] A. Ferrara, D. Lorusso, S. Montanelli, and G. Varese, “Towards a Benchmark for Instance Matching,” in *OM*, 2008.

[22] “Ontology Alignment Evaluation Initiative,” http://oaei.ontologymatching.org/.

[23] A. Ferrara, S. Montanelli, J. Noessner, and H. Stuckenschmidt, “Benchmarking Matching Applications on the Semantic Web,” in *ESWC*, 2011.

[24] D. Barbosa, A. O. Mendelzon, J. Keenleyside, and K. Lyons, “ToXgene: an extensible template-based data generator for XML,” in *WebDB*, 2002.

[25] J. Euzenat, P. Shvaiko, and et. al., *Ontology matching.* Springer, 2007, vol. 18.

[26] E. Ioannou, N. Rassadko, and Y. Velegrakis, “On generating benchmark data for entity matching,” *Journal on Data Semantics*, vol. 2, no. 1, pp. 37–56, 2013.

[27] I. Fundulaki, “D1.1.1 Overview and analysis of existing benchmark frameworks,” Linked Data Benchmark Council, Tech. Rep., 2013, available at http://ldbc.eu/results/deliverables.

[28] W3C OWL Working Group, “OWL 2 Web Ontology Language,” http://www.w3.org/TR/owl2-overview/, 2012.

[29] P. Hayes, “RDF semantics,” http://www.w3.org/TR/rdf-mt/, 2004, W3C Recommendation, 10 February 2004.

[30] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz, “OWL 2 Web Ontology Language Profiles (Second Edition),” http://www.w3.org/TR/owl2-profiles/, w3C Recommendation 11 December 2012.

[31] D. L. McGuinness and F. van Harmelen, “OWL Web Ontology Language,” http://www.w3.org/TR/owl-features/, 2004.

[32] I. Fundulaki, N. Martinez, R. Angles, B. Bishop, and V. Kotsev, “D2.2.2 Data Generator,” Linked Data Benchmark Council, Tech. Rep., 2013, available at http://ldbc.eu/results/deliverables.

[33] D. Krompass, M. Nickel, X. Jiang, and V. Tresp, “ Non-Negative Tensor Factorization with RESCA,” in *ECML/PKDD*, 2013, workshop on Tensor Methods for Machine Learning.

[34] M. Nickel, V. Tresp, and H.-P. Kriegel, "Factorizing YAGO: Scalable Machine Learning for Linked Data," in *WWW*, 2012.

[35] B. Fuglede and F. Topsoe, "Jensen-shannon divergence and hilbert space embedding," in *IEEE International Symposium on Information Theory*, 2004, pp. 31–31.

[36] J. M. Joyce, "Kullback-leibler divergence," in *International Encyclopedia of Statistical Science*. Springer, 2011, pp. 720–722.

[37] S. Weisberg, *Applied linear regression*. John Wiley & Sons, 2014.

[38] E. Jiménez-Ruiz and B. C. Grau, "Logmap: Logic-based and scalable ontology matching," in *ISWC*. Springer, 2011, pp. 273–288.

[39] A. A. Romero, B. Grau, I. H. Ian, and E. Jiménez-Ruiz, "More: a modular owl reasoner for ontology classification." in *ORE*, 2013, pp. 61–67.

[40] R. Shearer, B. Motik, and I. Horrocks, "Hermit: A highly-efficient owl reasoner." in *OWLED*, vol. 432, 2008.

[41] S. Massmann, S. Raunich, D. Aumuller, P. Arnold, and E. Rahm, "Evolution of the coma match system," in *The Sixth International Workshop on Ontology Matching*. Springer, October 2011, pp. 146–171.