LANCE: Piercing to the Heart of Instance Matching Tools

Tzanina Saveta[1], Evangelia Daskalaki[1], Giorgos Flouris[1], Irini Fundulaki[1], Melanie Herschel[2], and Axel-Cyrille Ngonga Ngomo[3]

[1] Institute of Computer Science-FORTH[⋆]Greece,
[2] IPVS - University of Stuttgart, Germany
[3] IFI/AKSW, University of Leipzig, Germany

Abstract. One of the main challenges in the Data Web is the identification of instances that refer to the same real-world entity. Choosing the right framework for this purpose remains tedious, as current instance matching benchmarks fail to provide end users and developers with the necessary insights pertaining to how current frameworks behave when dealing with real data. In this paper, we present Lance, a domain-independent instance matching benchmark generator which focuses on benchmarking instance matching systems for Linked Data. Lance is the first Linked Data benchmark generator to support complex *semantics-aware* test cases that take into account expressive OWL constructs, in addition to the standard test cases related to structure and value transformations. Lance supports the definition of matching tasks with varying degrees of difficulty and produces a weighted gold standard, which allows a more fine-grained analysis of the performance of instance matching tools. It can accept *any* linked dataset and its accompanying schema as input to produce a target dataset implementing test cases of varying levels of difficulty. We provide a comparative analysis with Lance benchmarks to assess and identify the capabilities of state of the art instance matching systems as well as an evaluation to demonstrate the scalability of Lance's test case generator.

1 Introduction

Instance matching (IM), refers to the problem of identifying instances that describe the *same real-world object* (alternative names include entity resolution [1], duplicate detection [2], record linkage [3] and object identification in the context of databases [4]). 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] it is crucial to develop IM techniques adapted to this setting that is characterized by an unprecedented number of sources across which to detect matches, a high degree of heterogeneity both at the schema and at the

⋆ This work was partially supported by the EU FP7 projects LDBC (FP7-ICT-2011-8 #317548) and DIACHRON (ICT-2011.4.3, #601043).

[1] http://linkeddata.org/

instance level, and rich semantics that accompany schemas defined in terms of expressive languages such as OWL, OWL 2, and RDFS. For such data, many IM techniques have recently been proposed (e.g., [5,6], survey in [7]).

Clearly, the large variety of IM techniques requires their comparative evaluation to determine which technique is best suited for a given application. Performing such an assessment generally requires *well-defined and widely accepted benchmarks* to determine the weak and strong points of the methods or systems and to motivate the development of better systems in order to overcome identified weak points. Hence, suited benchmarks help push the limit of existing systems, advancing both research and technology. A number of benchmarks have already been proposed, both for relational and XML data [8] and, more recently, for RDF data, the type of data prevalent in the Web of Data [9,10,11,12].

This paper presents the *Linked Data Instance Matching Benchmark Generator*[2] (Lance), a novel IM *benchmark generator* for assessing IM techniques for RDF data with an associated schema. The main features of Lance are:

Wider set of test cases. Lance supports a set of *test cases* based on *transformations* that distinguish different types of matching entities. Similarly to existing IM benchmarks, Lance supports the *value-based* (typos, date/number formats, etc.) and *structure-based* (deletion of classes/properties, aggregations, splits, etc.) test cases. Lance is the *first benchmark generator* to support *explicitly* advanced *semantics-aware* test cases that go beyond the standard RDFS constructs. These test cases test the use of RDFS/OWL semantics to identify matches, and include tests involving *instance (in)equality*, class and property *equivalence* and *disjointness*, *property constraints*, as well as *complex class definitions*. Lance also supports *simple combination (SC)* test cases (implemented using the aforementioned transformations applied on different triples pertaining to the same instance), as well as *complex combination (CC)* test cases (implemented by combinations of individual transformations on the same triple).

Similarity score and fine-grained evaluation metrics. Lance provides an enriched, *weighted gold standard* and related evaluation metrics, which allow a more fine-grained analysis of the performance of systems for tests with varying difficulty. In particular, the ground truth (or gold standard, i.e., pairs consisting of an entity in the source dataset and its matching entity in the target dataset) is enriched with annotations specific to the test case that generated each pair, i.e., the type of test case it represents, the property on which a transformation was applied, and a *similarity score* (or *weight*) that essentially quantifies the difficulty of finding a particular match. This detailed information, which is not provided by previous benchmarks, allows Lance to adopt more detailed views and evaluation metrics to assess the completeness, soundness, and overall matching quality of an IM system. In particular, Lance uses the *average similarity score* of the gold standard in combination with the *standard deviation* of the weight of each pair from the average score in order to asses the benchmark's level of difficulty. This fine-grained analysis allows Lance users to more easily identify the reasons

[2] http://www.ics.forth.gr/isl/lance

underlying the obtained performance results, and thereby supports IM systems' debugging and extension.

High level of customization and scalability testing. LANCE provides the ability to build a benchmark on top of any dataset, thereby allowing the implementation of diverse test cases for different domains, dataset sizes and morphology. This makes LANCE highly customizable and domain-independent. Perhaps more importantly, this feature allows also systematic scalability testing of IM systems, a feature which is not available in most state-of-the-art IM benchmarks.

The rest of the paper is structured as follows: in Section 2, we discuss related work; Section 3 describes the different components of our benchmark generator; Section 4 demonstrates the suitability of our benchmark generator in assessing and identifying the capabilities of an IM system; Section 5 concludes the paper.

2 Related Work

Several benchmarks have been proposed for testing the performance of IM systems for Linked Data. These benchmarks were the first to consider structure-based test cases, as previous benchmarks for relational and XML data primarily focused on value-based ones. A summary of the benchmarks relevant for LANCE is shown in Table 1; a more complete survey can be found in [13].

Benchmark	VOL	VAL	STR	SEM	ML
Synthetic IM benchmarks					
IIMB (2009) [14]*	2K	√	√	ltd	
IIMB (2010) [15]*	14K	√	√	ltd	
PR (2010) [16]*	9K	√	√		
IIMB (2011) [17]*	4K	√	√	ltd	
Sandbox (2012) [18]*	4K	√			
IIMB (2012) [18]*	2K	√	√	ltd	
RDFT (2013) [19]*	4K	√	√		√
ID-REC (2014) [20]*	3K	√			
ONTOBI (2010) [10]	14K	√	√	ltd	√
LANCE (2015)	√	√	√	√	√
Real IM benchmarks					
ARS (2009) [14]*	1M	√	√		
DI (2010) [15]*	6K	√	√		
DI (2011) [17]*	N/A	√	√		

Table 1. IM benchmark summary showing dataset size (VOL), supported test cases (value-based (VAL), structure-based (STR), semantics-aware (SEM)) and support for multilinguality (ML). A star (*) indicates a benchmark proposed by OAEI [9].

Our approach is based on the test cases proposed by our previous work SPIMBENCH [12] but unlike SPIMBENCH, LANCE is a domain-independent benchmark generator.

OAEI. The most important initiative regarding IM benchmarks is the *Ontology Alignment Evaluation Initiative (OAEI)* [9] that organizes a related annual track since 2009. OAEI proposes benchmarks based on both real and synthetic datasets. Synthetic datasets are mostly small (up to a few thousand instances) but allow a more accurate evaluation of the matching quality of IM systems, since they provide an accurate gold standard that is automatically constructed. Real datasets are much larger

(millions of instances) and allow evaluating the scalability of IM frameworks; nevertheless, the provided gold standard is error-prone, as it is practically infeasible to identify the complete and correct set of matches either manually (ARS [14]) or semi-automatically (DI 2010 [15], DI 2011 [17]). Thus, evaluating the ability of IM methods to scale comes at the price of a less accurate evaluation of matching quality. LANCE avoids this trade-off, as it generates the datasets along with the gold standard containing the matched instances. Most OAEI benchmarks consider both value-based and structure-based test cases (see Table 1). The support for semantics-aware test cases is limited to the IIMB benchmarks: IIMB 2009 considered only simple features such as class hierarchies and the OWL `sameAs`, whereas later versions used the SWING benchmark data generator [21] to support more complex cases, but still in a limited fashion compared to LANCE. Multilinguality (an important feature in practice) supported by LANCE, is considered by RDFT [19] only.
ONTOBI. ONTOBI is a synthetic IM benchmark that uses the DBpedia ontology (v.3.4) to propose 16 different test cases that include spelling mistakes, suppressed comments, change in date and number formats, deleted data types, language modifications, random names changes, synonym-based changes, disjunct dataset and flattening/expansion of the structure. ONTOBI is a domain-specific benchmark that supports mainly value and structure based test cases, as well as a limited amount of semantics-aware ones. It considers larger datasets than OAEI, but still in the range of a few thousand triples.
SWING. The SWING benchmark data generator [21] provides a general framework for creating IM benchmarks; it supports various test cases based on value and structure transformations at the instance level. The semantics-aware test cases are built upon class and property subsumption hierarchies, class disjointness and inverse properties. LANCE builds on SWING to implement most of the value-based test cases, but is also applying some novel value-based transformations, as well as a richer set of structure-based and semantics-aware test cases. SWING generates an artificial benchmark (without size limitations) and the corresponding gold standard, based on a given schema, thus allowing the creation of domain-independent benchmarks suitable for both scalability and matching quality evaluation. However, unlike LANCE, SWING does not support weighted gold standards and thus provides less insights for developers to debug or improve their IM system.

3 LANCE

3.1 Transformation-based Test Cases

In LANCE we propose a set of *value-based*, *structure-based*, and *semantics-aware* test cases. The former two are implemented using *transformations* as proposed in Ferrara et. al [21] on *data* and *object type* properties respectively; the last refers to the use of a subset of OWL *semantic* constructs. Value and structure-based test cases are produced by applying the appropriate transformation(s) on a *source* instance to obtain a *target* instance. The same principle holds in the case of semantics-aware test cases, with the difference that appropriate instance-level triples are constructed and added in the target dataset to consider the respective OWL constructs. This pair of instances is then used as input for the instance matching system (along with the gold standard) to test its performance.

Value-based Test Cases refer to scenarios implemented using *transformations* on *instance data type properties* that consider mainly typographical errors and the use of different data formats. In LANCE we extended the transformations of SWING [21], by adding antonyms, country abbreviations and multilinguality. Table 2 presents the transformations implemented in LANCE. Each transformation takes as input a *data type property* and a *severity* that determines how severe this modification is. Transformations VT1-VT3 can be perceived as different cases of *misspellings*. VT4 addresses the use of different date formats; Abbreviations are addressed by VT5: LANCE supports abbreviations that are very common in texts (such as "United States of America" vs "USA"), as well as those of SWING. VT6 refers to the use of synonyms and antonyms taken from Wordnet[3]. Stemming is applied using transformation VT7. LANCE also supports *multilinguality* (transformation VT8) from English to 64 languages.

VT1	Blank char. Addition/Deletion
VT2	Random char. Addition/Deletion/Modification
VT3	Token Addition/Deletion/Shuffle
VT4	Date Formats
VT5	Country & Simple Abbreviations
VT6	Synonym/Antonym
VT7	Stem of a Word
VT8	Multilinguality

Table 2. LANCE value-based transformations

Structure-based Test Cases are based on transformations applied on *object and data type* properties of instances such as *splitting*, *aggregation*, *deletion* and *addition*. Splitting refers to expanding properties whereas aggregation refers to merging a number of properties to a single oneaddition to property aggregation we support all the structure-based transformations that are proposed and implemented in SWING. These transformations are a superset of those considered in other IM benchmarks (see Section 2).

Semantics-aware Test Cases are primarily used to examine if the matching systems take into consideration OWL and OWL 2 axioms to discover matches between instances that can be found only when considering schema information. The axioms that we consider in LANCE are:

- *class* and *property equivalence* (`equivalentClass`, `equivalentProperty`)
- *instance (in)equality* (`sameAs`, `differentFrom`)
- *class* and *property disjointness* (`disjointWith`, `AllDisjointClasses`, `property-DisjointWith`, `AllDisjointProperties`)
- *class* and *property* hierarchies (`subClassOf`, `subPropertyOf`)
- *property constraints* (`FunctionalProperty`, `InverseFunctionalProperty`)
- *complex class definitions* (`unionOf`, `intersectionOf`)

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

	SOURCE DATASET	TARGET DATASET	SCHEMA TRIPLES	GS
ltSubC	$(u_1, \texttt{rdf:type}, C_1)$	$(u'_1, \texttt{rdf:type}, C'_1)$	$(C_1, \texttt{subClassOf}, C'_1)$	(u_1, u'_1)
ltEqC	$(u_1, \texttt{rdf:type}, C_1)$	$(u'_1, \texttt{rdf:type}, C'_1)$	$(C_1, \texttt{equivalentClass}, C'_1)$	(u_1, u'_1)
ltSameAs1	$(u_1, \texttt{rdf:type}, C_1)$ $(u_2, \texttt{rdf:type}, C_1)$	$(u'_1, \texttt{rdf:type}, C_1)$ $(u'_2, \texttt{rdf:type}, C_1)$ $(u'_1, \texttt{sameAs}, u'_2)$		(u_1, u'_1) (u_1, u'_2) (u_2, u'_2) (u_2, u'_1)
ltDiff	$(u_1, \texttt{rdf:type}, C_1)$	$(u'_1, \texttt{rdf:type}, C_1)$ $(u''_1, \texttt{rdf:type}, C_1)$ $(u'_1, \texttt{differentFrom}, u''_1)$		(u_1, u'_1)
ltDisjC	$(u_1, \texttt{rdf:type}, C_1)$	$(u'_1, \texttt{rdf:type}, C'_1)$	$(C_1, \texttt{disjointWith}, C'_1)$	—
ltFuncP	(u_1, p_1, o_1)	(u_1, p_1, o'_1)	$(p_1$, `rdf:type` `FunctionalProperty`$)$	(o_1, o'_1)
ltInvFuncP	(u_1, p_1, o_1)	(o_1, p_1, u'_1)	$(p$, `rdf:type`, `InverseFunctionalProperty`$)$	(u_1, u'_1)
ltUnionOf	$(u_1, \texttt{rdf:type}, C_1)$	$(u'_1, \texttt{rdf:type}, C'_1)$	$(C'_1, \texttt{unionOf}, \{C_1, C_2, \ldots)$	(u_1, u'_1)
ltIntersect1	$(u_1, \texttt{rdf:type}, C_1)$	$(u'_1, \texttt{rdf:type}, C'_1)$	$(C_1, \texttt{intersectionOf}, S)$ $(C'_1, \texttt{intersectionOf}, S)$	(u_1, u'_1)
ltIntersect2	$(u_1, \texttt{rdf:type}, C_1)$	$(u'_1, \texttt{rdf:type}, C'_1)$	$(C_1, \texttt{intersectionOf}, S)$ $(C'_1, \texttt{intersectionOf}, S')$ $S' \subset S$	(u_1, u'_1)

Table 3. Semantics-aware test cases

Table 3 shows some of these semantics-aware test cases: column SCHEMA TRIPLES refers to *schema triples* that the instance matcher under test should take into consideration when performing the matching tasks and GS shows the pairs of matches (u, u') that will be included in the gold standard. In all the tables we write u to refer interchangeably to an RDF instance and its URI. The rules in Table 3 should not be viewed as inference rules, but as *hints* for a system to derive that a match holds.

Class Hierarchy & Equivalence: test cases **ltSubC**, **ltEqC** shown in Table 3 consider the `subClassOf` and `equivalentClass` constructs respectively. Given a source URI u_1, instance of class C_1, we create target URI u'_1, instance of class C'_1 by copying all the properties of u_1 (except `rdf:type` triples) to create u'_1. For **ltSubC**, C_1 is a subclass of C'_1 (schema triple (C_1, `subClassOf`, C'_1)) and u_1 and u'_1 are considered as *matches* in the gold standard since they are of similar type due to the `subClassOf` semantics that specify that a class contains all instances of its subclasses. For **ltEqC**, C_1 and C'_1 are equivalent classes (schema triple (C_1, `equivalentClass`, C'_1)), so the two instances are considered matches since they are of the same type due to the semantics of class equivalence, according to which two equivalent classes have the same set of instances.

The rationale for properties is exactly the same since `subPropertyOf` and `equivalentProperty` axioms have similar semantics as their class counterparts. Test cases for `subClassOf` and `subPropertyOf` hierarchies are supported by the IM benchmarks that provide a limited support for this type of tests [10,14,15,17,18].

Instance (in)equality: test case **ltSameAs1** shown in Table 3 is a complex test for OWL construct `sameAs`; for this case we consider two source URIs u_1 and u_2 instances of the same class C_1; for u_1 and u_2, we create target instances u'_1 and u'_2. These are added in the target dataset along with triple (u'_1, `sameAs`, u'_2). A matcher that understands the semantics of `sameAs` should report all possible four matches between instances u_1, u'_1, u_2 and u'_2, otherwise it will report matches (u_1, u'_1) and (u_2, u'_2). OWL construct `differentFrom` is used to explicitly state that two resources refer to different real world objects. Test case **ltDiff** shown in Table 3 follows the same lines as the test case for `sameAs` construct: for a source instance u_1, we create two target instances u'_1 and u''_1 by copying all the properties of u_1 (including the `rdf:type` property). Target instance u''_1 is obtained by applying additional value and structure transformations to u'_1. Triple (u'_1, `differentFrom`, u''_1) is also added in the target dataset. If the matcher does not take under consideration the `differentFrom` construct it should produce a match between instances u_1 and u''_1 when it should not, since there is an *explicit* statement that these two instances refer to a different real world object (u_1, `differentFrom`, u''_1). Note that for all the test cases concerning `sameAs` and `differentFrom` OWL constructs, we assume that the source and target instances are of the same *type* (i.e., belong to the same class).

Class Disjointness: test case **ltDisjC** shown in Table 3 addresses class disjointness. To implement this test we produce target instance u'_1 from source instance u_1 as discussed before; these are instances of two *disjoint* classes C_1 and C'_1 - schema triple (C_1, `disjointWith`, C'_1) - respectively. In this case, the matcher should not return any match since according to the OWL semantics, two disjoint classes *cannot* share the same set of instances. Disjointness of properties follows the same rationale as disjointness of classes. Test cases for `AllDisjointClasses` and `AllDisjointProperties` follow the same principles for `disjointWith` and `propertyDisjointWith` respectively.

Functional & Inverse Functional Properties: A functional property is a property that can have only one (unique) value y for each instance x. Inverse functional properties are useful to denote values that uniquely identify an entity. Note that, due to the fact that the semantics of OWL do not include the Unique Name Assumption, 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 declared by the OWL semantics. **ltFuncP** test case shown in Table 3 considers `FunctionalProperty`: for a source instance u_1, subject of triple (u_1, p_1, o_1) with p_1 being a functional property (schema triple (p_1, `rdf:type`, `FunctionalProperty`)) we produce a triple (u_1, p_1, o'_1) in the target dataset, where o'_1 is obtained by applying a set of value and structure based transformations. If the matcher takes into consideration the fact that p_1 is a functional property, then it should produce a match between instances o_1 and o'_1 since according to the semantics of `FunctionalProperty` if a property p is declared as functional, then an instance u cannot have two properties p with different values. The same rationale is followed for **ltInvFuncP** test case that addresses `InverseFunctionalProperty`.

Complex Class Definitions: LANCE supports test cases for the `unionOf` and `intersectionOf` constructs shown in Table 3. As with all OWL constructs, the semantics of `unionOf` are intentional: `unionOf` implies a subsumption relationship between the constituents of the union, and the union itself. Therefore, if a class A is defined as a union of A_1, A_2, …, A_k then all instances that are known to be instances of any A_i,...,A_k will also be instances of their union. In **ltUnionOf** we assume that C'_1 is defined as a union of a set of classes $C_1, C_2, \ldots C_k$. For this test, we create for u_1 instance of class C_1 in the source dataset, u'_1 instance of class C'_1 in the target dataset. According to the `unionOf` semantics, u'_1 is an instance of class C'_1, and hence we go back to the `subClassOf` test case. Hence, we add (u_1, u'_1) as matched instances in the gold standard.

Similar to `unionOf`, `intersectionOf` semantics are also intentional: if a class A is defined as an intersection of A_1, A_2, …, A_k then A contains exactly those instances that are common to all classes. In addition, A is defined as the subclass of A_1, A_2, ... A_k. In **ltIntersect1**, u_1 is an instance of class C_1 in the source dataset and u'_1 is an instance of class C'_1 in the target dataset. C_1 and C'_1 are defined as the intersection of the *same* set of classes S. In that case, u_1 and u'_1 have the same type, and we include pair (u_1, u'_1) in the gold standard. In **ltIntersect2**, classes C_1 and C'_1 are defined as the intersection of two different sets of classes S and S', the latter being a subset of the former. Instances u_1 and u'_1 are again reported as matches in the gold standard since they have the same type (through the semantics of `intersectionOf`).

Simple and Complex Combination Test Cases In LANCE we consider combinations of the aforementioned test cases. We distinguish between *simple combination (SC)* test cases based on value, structure based and semantics-aware test cases, applied on different triples pertaining to one class instance. For example, for an instance u, we can perform a value-based transformation on its triple (u_1, p_1, o_1) where p_1 is a data type property and a structure-based transformation on its triple (u_1, p_2, o_2). We also consider *complex combination (CC)* test cases that are based on combinations of test cases applied to a *single* triple along with a transformation applied to the class of the instance. For instance, when a semantics-aware test case is considered, then for a triple (u_1, p_1, o_1) we can produce a triple (u_1, p'_1, o'_1) where p_1 is a subproperty of p'_1 and o'_1 is obtained by applying a value transformation on o_1.

3.2 Weighted Gold Standard

In the following, we present the weighted gold standard generated by LANCE. We begin by presenting how we store the transformations that were used to generate a target instance u'_i based on a source instance u_i. Thereafter, we present our approach to computing similarity scores for each pair (u_i, u'_i).

Computing the similarity scores To improve the debugging of instance matching tools and algorithms, we assign a similarity score (weight) to each pair of instances that should be matched. In essence, the weight of a match (u_i, u'_i) quantifies how similar the source and target instances are. We adopt an information-theoretical approach to compute the weight w of (u_i, u'_i) by measuring the information loss that results from applying transformations to the source data to generate the target data. The basic idea behind our approach is to apply a multi-relational learning (MRL) approach $\mathcal{L}$ to the input knowledge base K and the transformed knowledge base K'. By comparing the description of u_i in $\mathcal{L}(K)$ and u'_i in $\mathcal{L}(K')$, we should then be able to quantify how much information was lost through the transformation of K to K'. We implement this insight in the current version of LANCE by using RESCAL [22,23] as MRL approach.

The idea behind RESCAL is that each RDF graph K can be represented as a tensor $\mathcal{T}$ of order 3 and dimensions $|R| \times |R| \times |P|$, where R is the set of all RDF resources, P is the set of all RDF properties and $\mathcal{T}(i, j, k) = 1$ iff $< u_i, p_k, u_j > \in K$. Let $\mathcal{T}(\cdot, \cdot, k)$ be the kth $|R| \times |R|$-matrix that makes up $\mathcal{T}$, i.e., the matrix that is such that $\mathcal{T}(\cdot, \cdot, k)_{ij} = 1$ iff $< u_i, p_k, u_j > \in K$. RESCAL approximates the matrix A which minimizes the error $||\mathcal{T}(\cdot, \cdot, k) - X_k A X_k^\top||_F^2$ over all $\mathcal{T}(\cdot, \cdot, k)$ simultaneously. Based on A and the X_k matrices, we can approximate the whole of $\mathcal{T}$ to a tensor $\tilde{\mathcal{T}}$ with $\tilde{\mathcal{T}}(\cdot, \cdot, k) = X_k A X_k^\top$. As shown in previous work [23], each matrix $\tilde{\mathcal{T}}(i, \cdot, \cdot)$ contains all predicted relations of the resource u_i. Hence, it can be regarded as a complete description of u_i. The similarity in information content of u_i and u'_i can thus be computed by using the squared cosine of the angle between the matrices $\tilde{\mathcal{T}}(i, \cdot, \cdot)$, $\tilde{\mathcal{T}}'(i, \cdot, \cdot)$, where $\tilde{\mathcal{T}}'(i, \cdot, \cdot)$ is the tensor that results from applying the transformations above to the input (K):

$$cos^2(\tilde{\mathcal{T}}(i, \cdot, \cdot), \tilde{\mathcal{T}}'(i, \cdot, \cdot)) = \frac{\sum\limits_{jk} \tilde{\mathcal{T}}(i, j, k) \tilde{\mathcal{T}}'(i, j, k)}{||\tilde{\mathcal{T}}(i, \cdot, \cdot)||_F^2 ||\tilde{\mathcal{T}}'(i, \cdot, \cdot)||_F^2}. \tag{1}$$

A squared cosine value close to 1 suggests that u_i and u'_i contain similar information and that the information loss due to the transformation was small. Hence, it should be easier for an instance matching framework to detect this match than a match with a smaller squared cosine similarity.

On the hardware used for our evaluation of LANCE (see Section 4), RESCAL's performance grew linearly with the size of the benchmark. In particular, the approach required approximately 6 minutes to compute $\tilde{\mathcal{T}}$ for 10^4 triples. While the corresponding waiting times are acceptable for up to medium-sized datasets (i.e., data sets in orders of magnitude up to 10^5 triples), they are too large to be used when generating large benchmarks with more than 10^6 triples. We thus extended the approach above to be used on larger data sets by using sampling.

The idea behind our sampling approach is to partition the input knowledge base K into n partitions $K_1 \ldots K_n$ of the same size and run the approach above on user-selected partitions. Now for each pair of resources (u_i, u'_i) from the gold standard that belongs to the user-chosen partitions, we can compute a weight where instance u_i, and its transformed instance u_i are stored in partitions K_i

(input knowledge base) and K_i' (transformed knowledge base). In addition, we know how many transformations of which type were used to generate u_i' out of u_i. Based on this information, we can compute how much each transformation contributes to the information loss that occurs when generating u_i out of u_i' by solving the corresponding linear regression problem. Note that the matrices generated when applying our approach are commonly degenerate and that we thus use a numerical solver based on gradient descent to detect an approximate solution.

3.3 Metrics

The performance metric(s) in a benchmark determine the *effectiveness* and *efficiency* of the IM systems and tools. Traditionally, IM benchmarks focus on the quality of the output in terms of standard metrics such as *precision*, *recall* and *f-measure* [24]. In LANCE, opportunities for more sophisticated metrics arise due to the use of a *weighted gold standard*, which records, for each match, the *similarity* (or *weight*) of its source and target instances that is in the range $0 \dots 1$, and essentially quantifies the *difficulty* for an IM system to find this match.

In particular, a weight close to 1 means that the two instances are similar i.e., practically no transformations were applied to the source instances in order to generate the target instance; this match can be discovered relatively "easily" by an IM system and is hence considered a *low-difficulty* match. On the other hand, a weight close to 0 means that the target instance was obtained by applying complex transformations such as changing the topology of the graph through semantics-aware test cases (i.e., changing the class type of an instance), together with structure-based ones. This is a *difficult match* to discover for an IM system, since it needs to use effective similarity algorithms and be aware of possibly complex constraints or lower the employed threshold consequently affecting negatively precision.

In particular, by knowing the similarities of the matched instances recorded in the gold standard of a LANCE benchmark (say w_i), we can compute its *average similarity score* and the *standard deviation of its similarities.* These two numbers describe the average "difficulty" of the matched instances (i.e., of the test cases implemented in the benchmark) and the spread of the similarity scores (difficulty) in their range $(0 \dots 1)$.

A benchmark with a *high average similarity score* contains matched instances that are easier to find (easier cases have weights close to 1); a benchmark with a high standard deviation means that the weights are spread out from the average, so there is a larger variety of weights in the gold standard. The formulas are:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} w_i \qquad \sigma = \frac{1}{N}\sum_{i=1}^{N}(w_i - \mu)^2$$

In a similar fashion, we can compute the average and standard deviation of the *true positives* of a tested IM system (returned matches that are also in the gold standard). By comparing these numbers with the corresponding numbers for the benchmark, we can get a more fine-grained understanding of the system's effectiveness. In particular, comparing the averages, we can determine whether the IM system was able to find the easier or the more difficult matches; comparing the standard deviations gives an indication of whether the system is good for a specific range of transformations (as indicated by a deviation that is smaller than the benchmark's standard deviation) or for many different ones.

4 Evaluation

Applicability and scalability Our evaluation focused on demonstrating the capability of our benchmark generator in assessing and identifying the strengths and weaknesses of instance matching systems. For this purpose, we evaluated LogMap Version 2.4 [25] using the MoRe [26] reasoner, OtO [27] and LIMES [6] running the EAGLE [28] algorithm (Section 4.1). We chose these tools because they are prototypical working instances of existing IM systems[4]. LogMap considers both schema and instance level matching; hence it should perform well on all variations of the benchmark. OtO on the other hand, needs to be configured manually to implement instance matching tasks, so we assume that it will perform well on tasks with value transformations. The same holds for EAGLE, which can learn specifications and focuses on instance matching tasks only; we expect EAGLE to have a hard time at finding matches when faced with semantic transformations. We also report on the *scalability* aspect of LANCE (Section 4.1). The purpose of this experiment is to show that LANCE can be used for source datasets of arbitrary size and can generate target datasets that implement a large number of test cases without any additional processing overhead.
Datasets We used as source datasets those generated by LDBC's[5] SPIMBENCH [12]. Nevertheless, various data generators can be used in order to produce the *source* datasets. Indicatively we name Berlin SPARQL Benchmark (BSBM) [31], the DBpedia SPARQL Benchmark [32] and UOBM [33]. Due to space constraints we only present results achieved when using SPIMBENCH datasets. We produced two datasets, one with 10K triples and around 500 instances, and a larger one with 50K triples and around 2500 instances. All experiments were conducted on an Intel(R) Core(TM) 2 Duo CPU E8400 @3.00GHz with 8G of main memory running Windows 7 (64-bit).

Implementation of LANCE LANCE[6] is a highly configurable instance matching benchmark generator for Linked Data that consists of two components : (i) an *RDF repository* that stores the *source datasets* and (ii) a *test case generator* (see Figure 1). The test case generator takes as input a *source* dataset and produces a *target* dataset that implements various *test cases* according to the specified

[4] Attempts to evaluate LANCE benchmarks with systems such as RiMOM-IM [29], COMA++ [30] and CODI [17] were not successful. We were not able to work with RiMOM-IM due to incomplete information regarding the use of the system; COMA++ supports instance-based ontology matching but does not aim for instance matching per se. Finally CODI is no longer supported by the development team.
[5] LDBC Semantic Publishing Benchmark: http://ldbcouncil.org/developer/spb
[6] The code of LANCE is available at https://github.com/jsaveta/Lance

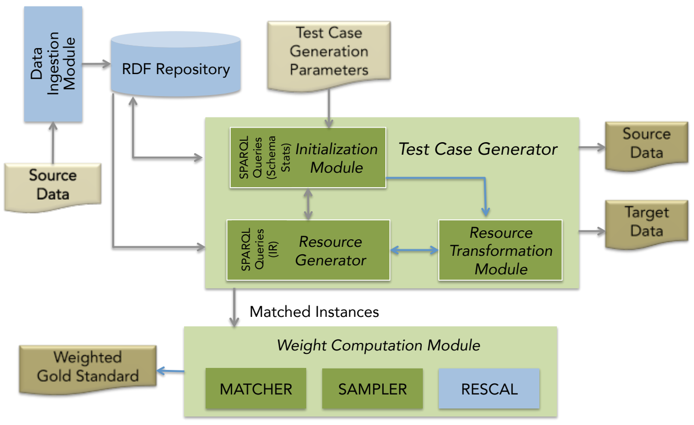

Fig. 1. LANCE System Architecture

configuration parameters to be used for testing instance matching tools. It consists of the *Initialization*, *Resource Generator* and the *Resource Transformation* modules. The first reads the test case generation parameters and retrieves by means of SPARQL queries the schema information (e.g., schema classes and properties) from the RDF repository that will be used for producing the target dataset. The *Resource Generator* uses this input to retrieve instances of those schema constructs from the RDF repository and passes those (along with the configuration parameters) to the *Resource Transformation Module*. The latter returns for a source instance u_i the transformed instance u'_i and stores this in the target dataset; this module is also responsible in producing an entry in the gold standard. Once LANCE has performed all the requested transformations, the *Weight Computation Module* calculates the similarity scores of the produced matches as discussed in Section 3.2. The configuration parameters specify the part of the schema and data to consider when producing the different test cases as well as the the percentage and the type of transformations to consider. More specifically, parameters for *value-based* test cases specify the kind and severity of transformation to be applied; for *structure* and *semantics-aware* test cases the parameters specify the type of transformation to be considered. The idea behind configuration parameters is to allow one to tune the benchmark generator into producing benchmarks of varying degrees of difficulty which test different aspects of an instance matching tool. LANCE is implemented in Java and in the current version we use OWLIM Version 2.7.3. as our RDF repository.

4.1 Experimental Results

Applicability of LANCE In order to show that LANCE is well suited to identify strong and weak points of state-of-the-art IM systems, we provided the tools at hand with difficult tasks and allowed the whole of the source dataset to be transformed so as to obtain the target dataset. Figure 2 reports the results for the different types of test cases and for datasets up to 10K and 50K triples. In all cases, we measured recall, precision, f-measure along with the *similarity score* and *standard deviation* we introduced in Section 3.3.

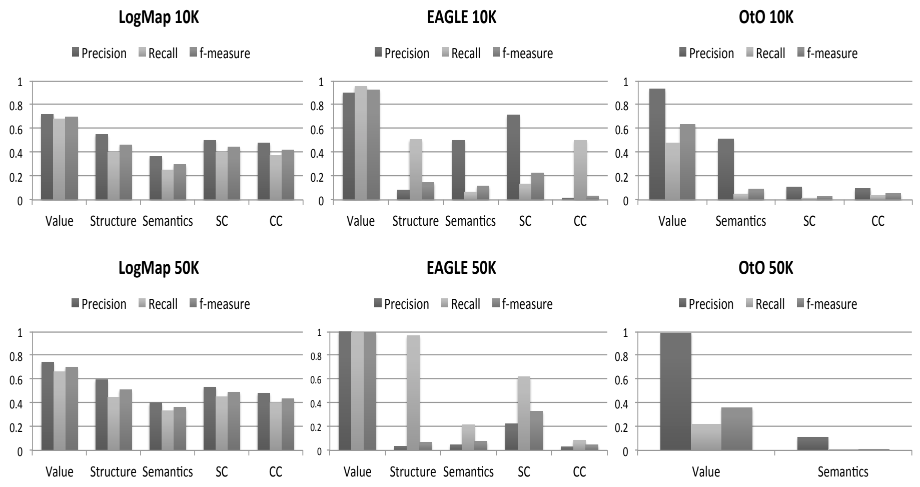

Fig. 2. Applicability experiments for LogMap, EAGLE and OtO

As expected, LogMap responds well to the value-based test cases having a high precision and recall (close to 0.75) but its performance degrades when the instances are involved in semantics-aware test cases with precision and recall (below 0.4). Still, the large number of transformations applied to the source dataset to generate the target dataset suggest that LogMap does indeed perform sufficiently well when faced with semantics-aware transformations. OtO gives very good precision results for the value-based test cases but faces many issues concerning all the others as in some cases is not able to find any match (recall is below 0.1). EAGLE also reacts as expected. The algorithm performs well when faced with syntactic transformations. Increasing changes to the topology of the underlying RDF graphs (the case of semantics-aware test cases) leads to a degradation of the performance of the algorithm. The performance of EAGLE is not consistent since it is non-deterministic and uses unsupervised learning. We ran EAGLE thrice for both datasets. The sim-

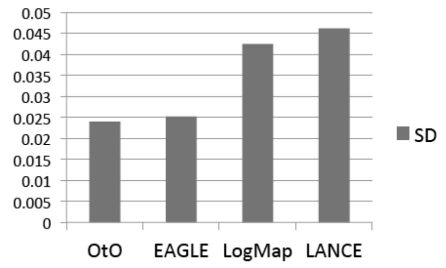

Fig. 3. Standard Deviation for LogMap, EAGLE, OtO, for 10K and semantics-aware test cases. The standard deviation of the gold standard is also shown (column LANCE).

ilarity scores as well as the standard deviation of the results returned by the instance matching systems provide insights on the ability of the systems to address the challenges proposed by Lance benchmarks. Figures 3 and 4 give the standard deviation and similarity scores for all three systems and for the semantics-aware test cases in the case of the 10K triples dataset. They also show the corresponding quantities for the benchmark itself for comparison. We can see that LogMap reports scores and standard deviation close to the ones given by Lance verifying that it can address the “difficult” test cases. EAGLE and OtO report lower similarity scores and standard deviation, meaning that they cannot address the challenges imposed by the, harder, semantics-aware test cases. In summary, we conclude that Lance is able to determine the capabilities of the IM systems and also reflect the difficulty of the test cases through the *weighted gold standard*.

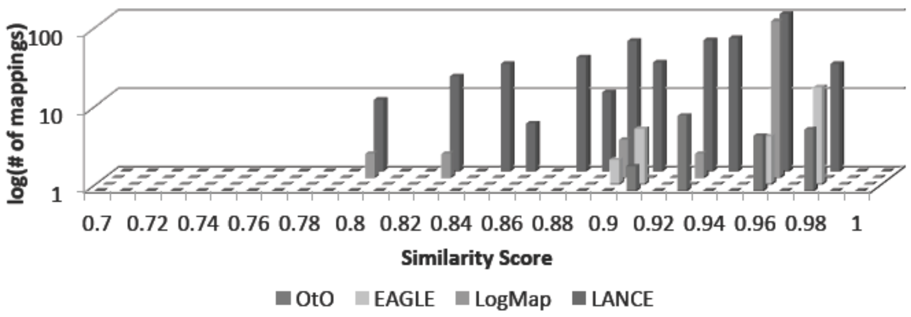

Fig. 4. Similarity score distribution for LogMap, EAGLE, OtO, for 10K and semantics-aware test cases. The similarity score of the benchmark is also shown (column Lance).

Scalability We also studied the scalability of the test case generator by measuring the runtime required by our framework to generate the *target* datasets for all different test cases, for various dataset sizes and percentages of source instances to be transformed. This time also includes the time required to retrieve the source instances from the RDF repository as previously discussed. We observe that the time for the data transformation is linear to the dataset size.

5 Conclusions

This paper presents Lance, an instance matching benchmark generator focusing on benchmarking instance matching systems for Linked Data. Lance is a domain-independent, highly modular and configurable generator that can accept as input *any* linked dataset and its accompanying schema to produce a target dataset implementing matching tasks of varying levels of difficulty. LANCE is the first Linked Data benchmark generator to support complex *semantics-aware* test cases that take into account expressive OWL constructs, in addition to the standard test cases related to structure and value transformations. The former type is largely absent in previous efforts. LANCE also produces a provably correct gold standard that allows a more fine-grained analysis of the performance of instance matching tools. This is in contrast to other benchmarks which are either based on manually generated gold standards, or based on gold standards produced semi-automatically (and are thus limited by the quality of the used approach, often producing inaccurate gold standards). Moreover, LANCE proposes the use of a *weighted gold standard* which records the similarity between a pair of matched instances as well as information on the type of transformation that was used to produce said match. This motivates the developers of IM systems to try benchmarks of varying levels of difficulty, and helps them identify the weak points of their systems, explain benchmark performance and more easily debug them. In the future, we plan to extend LANCE to work with spatial and streaming data; we also intend to work with datasets that include *blank nodes* thereby creating more challenging tasks for instance matching tools. Last, we plan to evaluate the frequency of appearance of the various types of transformations in existing datasets that would help us create realistic test cases.

References

1. I. Bhattacharya and L. Getoor. *Entity resolution in graphs. Mining Graph Data.* Wiley and Sons, 2006.
2. A. K. Elmagarmid, P.G. Ipeirotis, et al. Duplicate Record Detection: A Survey. *TKDE*, 19(1), 2007.
3. C. Li, L. Jin, et al. Supporting efficient record linkage for large data sets using mapping techniques. In *WWW*, 2006.
4. J. Noessner, M. Niepert, et al. Leveraging Terminological Structure for Object Reconciliation. In *ESWC*, 2010.
5. R. Isele, A. Jentzsch, et al. Silk Server - Adding missing Links while consuming Linked Data. In *COLD*, 2010.
6. A.-C. Ngonga Ngomo and Soren Auer. LIMES - A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data. *IJCAI*, 2011.
7. K. Stefanidis, V. Efthymiou, et al. Entity resolution in the web of data. In *WWW, Companion Volume*, 2014.
8. M. Weis, F. Naumann, et al. A Duplicate Detection Benchmark for XML and Relational Data. In *IQIS*, 2006.
9. Ontology Alignment Evaluation Initiative. http://oaei.ontologymatching.org/.
10. K. Zaiss, S. Conrad, et al. A Benchmark for Testing Instance-Based Ontology Matching Methods. In *KMIS*, 2010.
11. B. Alexe, W.-C Tan, et al. STBenchmark: Towards a benchmark for mapping systems. In *PVLDB*, 2008.
12. T. Saveta, E. Daskalaki, et al. Pushing the limits of instance matching systems: A semantics-aware benchmark for linked data. In *WWW, Companion Volume*, 2015.
13. E. Daskalaki, I. Fundulaki, et al. Instance Matching Benchmarks for Linked Data. ISWC (Tutorial), 2014.

14. J. Euzenat, A. Ferrara, et al. Results of the Ontology Alignment Evaluation Initiative 2009. In *ISWC Workshop on Ontology Matching (OM)*, 2009.
15. J. Euzenat. Results of the Ontology Alignment Evaluation Initiative. In *OM*, 2010.
16. OAEI Instance Matching. http://oaei.ontologymatching.org/2010, 2010.
17. J. Euzenat et. al. Final results of the Ontology Alignment Evaluation Initiative 2011. In *OM*, 2011.
18. J. L. Aguirre et. al. Results of the Ontology Alignment Evaluation Initiative 2012. In *OM*, 2012.
19. Z. Dragisic, K. Eckert, et al. Results of the Ontology Alignment Evaluation Initiative 2013. In *OM*, 2013.
20. Z. Dragisic, K. Eckert, et al. Results of the Ontology Alignment Evaluation Initiative 2014. In *OM*, 2014.
21. A. Ferrara, S. Montanelli, et al. Benchmarking Matching Applications on the Semantic Web. In *ESWC*, 2011.
22. D. Krompass, M. Nickel, et al. Non-Negative Tensor Factorization with RESCAL. In *TML*, 2013.
23. M. Nickel, V. Tresp, et al. Factorizing YAGO: Scalable Machine Learning for Linked Data. In *WWW*, 2012.
24. C. Goutte and E. Gaussier. A probabilistic interpretation of precision, recall, and F-score, with implication for evaluation. In *ECIR*, 2005.
25. E. Jiménez-Ruiz and B. C. Grau. Logmap: Logic-based and scalable ontology matching. In *ISWC*, 2011.
26. A. A. Romero, B.C. Grau, et al. MORe: a Modular OWL Reasoner for Ontology Classification. In *ORE*, pages 61–67, 2013.
27. E. Daskalaki and D. Plexousakis. OtO Matching System: A Multi-strategy Approach to Instance Matching. In *CAiSE*, 2012.
28. A.-C. Ngonga Ngomo and K. Lyko. EAGLE: Efficient Active Learning of Link Specifications using Genetic Programming. In *ESWC*, 2012.
29. J. Li, J. Tang, et al. Rimom: A dynamic multistrategy ontology alignment framework. *TKDE*, 21(8), 2009.
30. S. Massmann, S. Raunich, et al. Evolution of the COMA match system. *Ontology Matching*, 49, 2011.
31. C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *IJSWIS*, 5(2), 2009.
32. M. Morsey, J. Lehmann, et al. DBpedia SPARQL Benchmark - Performance assessment with real queries on real data. In *ISWC*, 2011.
33. L. Ma, Y. Yang, et al. Towards a Complete OWL Ontology Benchmark. In *European Semantic Web Conference (ESWC)*, 2006.