REPORT ON DELIVERABLE D3.1

Initial release of the POI interlinking software

SLIPO

PROJECT NUMBER: 731581
START DATE OF PROJECT: 01/01/2017
DURATION: 36 months

Horizon 2020

SLIPO is a research project funded by European Commission's Horizon 2020 Programme.

Dissemination Level	Public
Due Date of Deliverable	Month 18, 30/06/2018
Actual Submission Date	27/06/2018
Work Package	WP3 - Data Integration and Quality Assurance
Tasks	3.1 Interlinking
Type	Other
Approval Status	Submitted for approval
Version	1.0
Number of Pages	56
Filename	D3.1_Initial_release_of_the_interlinking_software.pdf

Abstract

This document presents LIMES, our software framework for interlinking POIs. First, we briefly describe the current state of the art in geospatial data interlinking, as well as the initial versions of LIMES, which comprised the starting point of our work. Then, we present LIMES v1.0.0, the current version optimized for scalable and quality assured interlinking of POIs. Apart from implementation and deployment information, we describe the POI-specific link discovery algorithms we developed and tested against real-world, commercial POI datasets. Finally, we present our evaluation experiments assessing both the interlinking effectiveness and the scalability of the software.

History

Version	Date	Reason	Revised by
0.1	15/05/2018	Deliverable structure and initial content	Mohamed Sherif
0.2	22/05/2018	Added content on LIMES components and architecture section	Mohamed Sherif
0.3	30/05/2018	First draft	Mohamed Sherif
0.4	4/06/2018	Multiple revisions and improvements	Mohamed Sherif
0.5	12/06/2018	Multiple revisions and improvements	Mohamed Sherif, Kevin Dreßler
0.6	18/06/2018	Reviewed version	Giorgos Giannopoulos, Spiros Athanasiou
1.0	27/06/2018	Final version	Mohamed Sherif, Kevin Dreßler, Axel Ngonga

Author list

Organization	Name	Contact information
INFAI	Mohamed Sherif	sherif@informatik.uni-leipzig.de
INFAI	Axel Ngonga	ngonga@informatik.uni-leipzig.de
INFAI	Kevin Dreßler	dressler@informatik.uni-leipzig.de
ATHENA RC	Spiros Athanasiou	spathan@imis.athena-innovation.gr
ATHENA RC	Giorgos Giannopoulos	giann@imis.athena-innovation.gr
ATHENA RC	Dimitrios Skoutas	dskoutas@imis.athena-innovation.gr

Executive Summary

This document presents LIMES, our framework for the scalable and quality assured interlinking of Point of Interest (POI) datasets. Interlinking, which is generally referred to as Link Discovery (LD) in literature, is the process of finding related entities in (not necessarily distinct) knowledge bases. An interlinking task generates a set of mappings among such knowledge bases as its output. In the rest of this deliverable, we use the terms of *interlinking* and *link discovery* interchangeably.

LIMES implements a series of measures and distance functions for link discovery among POI resources, as well as machine learning algorithms for automatic discovery of links among POI resources. Further, LIMES incorporates mechanisms for interlinking validation and quality statistics/indicators extraction to facilitate and assure the quality of the interlinking process. The presented version of LIMES extends the initial framework (LIMES v0.6.0), which was developed for dealing with traditional resources commonly described using string of characters, and with no geospatial representations as the ones available in POI resources tackled by SLIPO. The current version, LIMES v1.0.0, developed in the context of SLIPO, has been extended to specialize on the effective and scalable interlinking of POI entities.

The layout of document is the following.

In Section 1, we introduce the setting of the interlinking task. We first describe the objectives of interlinking in the frame of SLIPO project, and then we provide some background knowledge, briefly presenting existing works on interlinking of geospatial entities. Finally, we briefly report our achievements until M18 of the project and the delivery of LIMES v1.0.0.

In Section 2, we review the roadmap for the development of LIMES, and present the evolution of the software from its initial version (v0.6.0) at the beginning of SLIPO. Also, we summarize the major advancements of the software during the first period of the project, and the key features of the current version LIMES v1.0.0, as well as the planed extensions towards LIMES v2.0.0.

In Section 3, we present a complete user's guide for LIMES v1.0.0, including building and installation instructions, configuration settings, and a short demonstration of the usage of the software.

Finally, in Section 4, we evaluate the new POI interlinking approaches implemented within LIMES.

Abbreviations and Acronyms

API	Application Programming Interface
CRS	Coordinate Reference System
KPI	Key Performance Indicator
ML	Machine Learning
RDF	Resource Description Framework
UI	User Interface
VM	Virtual Machine
LD	Link Discovery
LS	Link Specification
LIMES	Link Discovery Framework for Metric Spaces
LSL	LIMES Specification Language

Table of Contents

1. Introduction

In this section, we discuss the basic concepts and goals of POI data Interlinking. Then, we present the current state of the art in POI interlinking approaches and frameworks, and finally, we briefly present our goals and achievements regarding the scalable, effective, and quality-assured interlinking of POIs.

1.1. POI data Interlinking

Establishing links between knowledge bases is one of the key steps of the Linked Data publication process[1]. A plethora of approaches has thus been devised to support this process [1]. In this document, we present the LIMES framework, which was designed to accommodate a large number of link discovery approaches within a single extensible architecture. LIMES was designed as a declarative framework (i.e., a framework that processes link specifications, to address two main challenges.

1. **Time-efficiency:** The mere size of existing knowledge bases (e.g., 30+ billion triples in LinkedGeoData [2], 20+ billion triples in LinkedTCGA [3]) makes efficient solutions indispensable to the use of link discovery frameworks in real application scenarios. LIMES addresses this challenge by providing time-efficient approaches based on the characteristics of metric spaces [4, 5], orthodromic spaces [6] and on filter-based paradigms [7].
2. **Accuracy:** Efficient solutions are of little help if the results they generate are inaccurate. LIMES accommodates solutions that allow the generation of links between knowledge bases with a high accuracy. These solutions abide by paradigms such as batch and active learning [8, 9, 10], unsupervised learning [10] and even positive-only learning [11].

1.1.1. The Link Discovery Problem

Interlinking, which is generally referred to as **Link Discovery (LD)** in literature, is the process of finding related entities in (not necessarily distinct) knowledge bases. An interlinking task generates a set of mappings among such knowledge bases as its output.

Formally, given two sets S, T of source and target resources respectively, as well as a relation R, the goal of interlinking is to find the set $M \subseteq S \times T$ of pairs $(s, t) \in S \times T$ such that $R(s, t)$. Note that, S and T are two not necessarily distinct sets of instances. One way to automate this discovery is to compare the $s \in S$ and $t \in T$ based on their properties using a (in general complex) similarity metric. Two entities are then considered to be linked via R if their similarity is superior to a threshold θ. If R is `owl:sameAs`, then we are faced with a deduplication task. The direct computation of the pairs for which R holds is commonly very tedious if at all possible. Thus, most frameworks for LD resort to approximating the set of pairs for which R holds by using Link Specifications (LS). A LS can be regarded as a classifier C_{ij} that maps each element of the knowledge bases' cartesian product $K_i \times K_j$ to one of the classes of $Y = \{+1, -1\}$, where K_i is called the set of source

[1] https://www.w3.org/DesignIssues/LinkedData.html

instances while K_j is the set of target instances. $(s, t) \in K_i \times K_j$ is considered by C_{ij} to be a correct link when $C_{ij}(s, t) = +1$. Otherwise, (s, t) is considered not to be a potential link.

The formal specification of LD adopted herein is akin to that proposed in [12]. Given two (not necessarily distinct) sets S, T of source and target resources respectively, as well as a relation R, the goal of LD is to find the set $M = \{(s,t) \in S \times T : R(s, t)\}$ of pairs $(s, t) \in S \times T$ such that $R(s, t)$. In most cases, computing M is a non-trivial task. Hence, a large number of frameworks (e.g., SILK [13], LIMES [12] and KnoFuss [14]) aim to approximate M by computing the **mapping** $M' = \{(s,t) \in S \times T : \sigma(s,t) \geq \theta\}$, where σ is a similarity function and θ is a similarity threshold. For example, one can configure these frameworks to compare the dates of birth, family names and given names of persons across census records to determine whether they are duplicates. We call the equation which specifies M' a **link specification** (short **LS**; also called linkage rule in the literature, see e.g., [13]). Note that the LD problem can be expressed equivalently using distances instead of similarities as follows: Given two sets S and T of resources, a (complex) distance measure δ and a distance threshold $\theta \in [0,\bullet[$, determine $M' = \{(s,t) \in S \times T : \delta(s,t) \leq \tau\}$. Note that a distance function δ can always be transformed into a normed similarity function σ by setting $\sigma(x, y) = (1+\delta(x, y))^{-1}$. Hence, the distance threshold τ can be transformed into a similarity threshold θ by means of the equation $\theta = (1 + \tau)^{-1}$. Consequently, the concepts of distance and similarity are used interchangeably within this document.

Under this so-called **declarative paradigm**, two entities s and t are then considered to be linked via R if $\sigma(s, t) \geq \theta$. Naïve algorithms require $O(|S||T|) \in O(n^2)$ computations to output M'. Given the large size of existing POI knowledge bases, time-efficient approaches able to reduce this runtime are hence a central component of LIMES. In addition, note that the choice of appropriate σ and θ is central to ensure that M is approximated correctly by M'. Approaches that allow the computation of accurate σ and θ are thus fundamental for the LIMES framework [1]).

1.1.2. Formal Overview

Several approaches can be chosen when aiming to define the syntax and semantics of LSs in detail [9, 13, 14]. In LIMES, we chose a grammar with a syntax and semantics based on set semantics. This grammar assumes that LSs consist of two types of atomic components:

- **similarity measures** *m*, which allow the comparison of property values or portions of the concise bound description of two resources and
- **operators** *op*, which can be used to combine these similarities into more complex specifications.

Similarity measures could be *atomic* or *complex*. Without loss of generality, an **atomic similarity measure** is function that maps an input pair of resources (sets S, T of source and target resources respectively) into a similarity score between zero and one inclusive. Formally, we define an *atomic similarity measure a* as a function $a : S \times T \rightarrow [0,1]$. An example of an atomic similarity measure is the *edit* similarity. We define the edit similarity of two strings str_1 and str_2 as $(1 + lev(str_1, str_2))^{-1}$, where *lev* stands for the *Levenshtein* distance.). **A complex measure** m combines measures m_1 and m_2 using measure operators such as *min* and *max*. We use mappings $M \subseteq S \times T$ to store the results of the application of a similarity measure to $S \times T$ or subsets thereof. We denote the set of all mappings as $\mathbb{M}$.

We define a filter f as a function $f(m, \theta)$, where m is a measure (either simple of complex) and θ is the filter threshold. The output of the filter f is the subset of mappings M (generated from the application of the

measure *m*) with similarity values above or equal the filter threshold θ. We call a specification *atomic* when it consists of exactly one filter function. For example, the upper part of Figure 1 represent the atomic link specification *f(edit(:socId, :socId), 0.5).* For such atomic LS, the filter *f* will generate the subset of mappings between source and target resources with social security numbers' *edit distance* less than of equal *0.5*.

A *complex specification* can be obtained by combining two specifications L_1 and L_2 through an operator that allows the merging of the results of L_1 and L_2. Here, we use the operators *mappings-union* $\sqcap$, *mappings-intersect* $\sqcup$ and *mappings-difference* \ as they are complete w.r.t. the Boolean algebra and frequently used to define LSs. We denote the set of all LSs as $\mathbb{L}$.

An example of a complex LS is given in Figure 1. The result of such complex LS would be the *mappings-union* ($\sqcup$) of the mappings generated by applying the two atomic LSs *f(edit(:socId, :socId), 0.5) and f(trigrams(:name, :label), 0.5).*

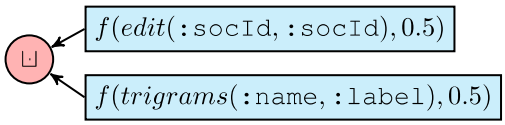


Figure 1: Example of a complex LS. The filter nodes are rectangles while the operator nodes are circles. :socID stands for social security number.

LS	$[[LS]]_M$
$f(m, \theta)$	$\{(s,t) \mid (s,t) \in M \wedge m(s,t) \geq \theta\}$
$L_1 \sqcap L_2$	$\{(s,t) \mid (s,t) \in [[L_1]]_M \wedge (s,t) \in [[L_2]]_M\}$
$L_1 \sqcup L_2$	$\{(s,t) \mid (s,t) \in [[L_1]]_M \vee (s,t) \in [[L_2]]_M\}$
$L_1 \backslash L_2$	$\{(s,t) \mid (s,t) \in [[L_1]]_M \wedge (s,t) \notin [[L_2]]_M\}$

Table 1: Link Specification Syntax and Semantics

We define the semantics $[[L]]_M$ of a link specification *L* w.r.t. a mapping *M* as given in Table 1. Those semantics are similar to those used in languages like SPARQL, i.e., they are defined extensionally through the mappings they generate. The mapping *[[L]]* of a link specification *L* with respect *to S x T* contains the links that will be generated by L.

1.2. State of the art

There has been a significant amount of work pertaining to the two key challenges of LD: **efficiency** and **accuracy**. Here, we focus on frameworks for link discovery. Dedicated approaches and algorithms can be found in the related work sections of [4, 5, 12, 9, 10, 15, 19, 25, 21]. As described in [1], most LD frameworks address the **efficiency challenge** by aiming to discover unnecessary computations of the similarity or distance measures efficiently. The accuracy challenge is commonly addressed using machine learning techniques.

One of the first LD frameworks is **SILK** [39], which uses a multi-dimensional blocking technique (MultiBlock [13]) to optimize the linking runtime through a rough index pre-matching. To parallelize the linking process,

SILK relies on *MapReduce*. Like LIMES, SILK configuration supports input files in XML and is also able to retrieve RDF data by querying SPARQL endpoints. Both frameworks also allow user-specified link types between resources as well as *owl:sameAs* links. SILK incorporates element-level matchers on selected properties using string, numeric, temporal and geo-spatial similarity measures. SILK also supports multiple matchers, as it allows the comparison of different properties between resources that are combined together using match rules. To this end, SILK implements supervised and active learning methods for identifying LS for linking.

KNOFUSS [14] incorporates blocking approaches derived from databases. Like LIMES, it supports unsupervised machine learning techniques based on genetic programming. However, in contrast to both aforementioned tools, KNOFUSS supports no other link types apart from `owl:sameAs` and implements only string similarity measures between matching entities properties. Additionally, its indexing technique for time complexity minimization is not guaranteed to achieve result completeness.

ZHISHI.LINKS [40] is another LD framework that achieves efficiency by using an index-based technique that comes at the cost of effectiveness. Similar to KNOFUSS, it only supports `owl:sameAs` links but implements geo-spatial relations between resources. In comparison with all previous three tools, it supports both property-based and semantic-based matching, using knowledge obtained by an ontology.

SERIMI [41] is a LD tool that retrieves entities for linking by querying SPARQL endpoints. In contrast to the previous frameworks, it only supports single properties to be used for linking resources. However, it incorporates an adaptive technique that weighs differently the properties of a knowledgebase and chooses the most representative property for linking.

SLINT+ [42] is a LD tool that is very similar to SERIMI but supports comparisons between multiple properties. Finally, there are a set of frameworks (**RIMOM** [43], **AGREEMENTMAKER** [44], **LOGMAP** [45], **CODI** [46]) that initially supported ontology matching and then evolved to support LD between resources.

RIMOM is based on Bayesian decision matching in order to link ontologies and transforms the problem of linking into a decision problem. Even though RIMOM utilizes dictionaries for ontology matching, it does not support them in entity linking.

Similar to RIMOM, **AGREEMENTMAKER** also uses dictionaries in ontology level. AGREEMENTMAKER incorporates a variety of matching methods based on different properties considered for comparison and the different granularity of the components.

LOGMAP is an efficient tool for ontology matching that scales up to tens (even hundreds) of thousands of different classes included in an ontology. It propagates the information obtained from ontology matching to link resources, by using logical reasoning to exclude links between resources that do not abide by the restrictions obtained from ontology matching.

CODI is a probabilistic-logical framework for ontology matching based on the syntax and semantics of *Markovlogic*. It incorporates typical matching approaches that are joined together to increase the quality of ontology alignment.

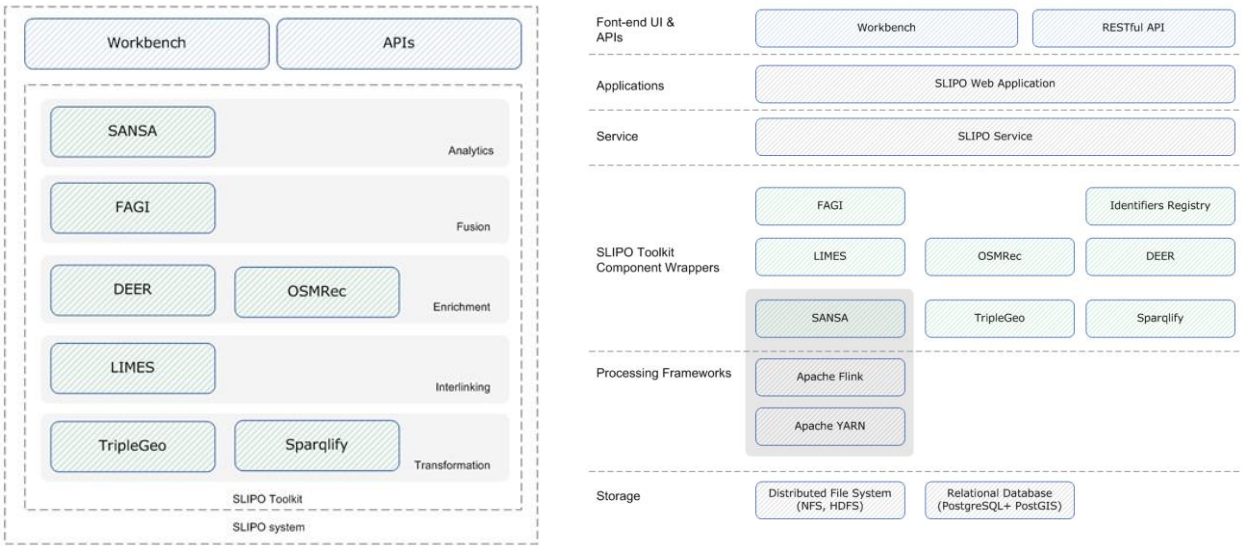
The basic difference between the previous LD frameworks and **LIMES** is that LIMES provides both theoretical and practical guarantees for efficient and scalable LD. LIMES is guaranteed to lead to exactly the same matching as a brute force approach while at the same time reducing significantly the number of

comparisons. The approaches incorporated in LIMES facilitate different approximation techniques to compute estimates of the similarity between instances. These estimates are then used to filter out a large number of those instance pairs that do not suffice the mapping conditions. By these means, LIMES can reduce the number of comparisons needed during the mapping process by several orders of magnitude. LIMES supports the first planning technique for link discovery, HELIOS, that minimizes the overall execution of a link specification, without any loss of completeness. As shown in our previous works (see [27, 19, 47]), LIMES is one of the most scalable link discovery frameworks currently available.

1.3. Interlinking in the SLIPO lifecycle

The SLIPO POI integration lifecycle is realized through the SLIPO Workbench, a platform for defining, executing and managing POI integration workflows (*see Deliverable D1.3 "Beta SLIPO Integrated System"*). These workflows combine all the components of the SLIPO Toolkit, supporting the integrated execution of all four core POI integration steps: *transformation, interlinking, enrichment and fusion*. Additionally, the SLIPO system prescribes a set of value-added services on top of integrated POI datasets (Figure 2 (a)).

The goal of Task 3.1 is to deliver LIMES, the interlinking framework of SLIPO. LIMES incorporates many algorithms for performing efficient interlinking among POI resources. In the context of SLIPO, LIMES receives as input two RDF POI datasets conforming to the SLIPO ontology. Thus, LIMES's input POI data are first transformed by *TripleGeo*, into the proper RDF format and schema. Further, apart from the two input POI datasets, LIMES requires as input a configuration file containing the LIMES configuration parameters. LIMES's output consists in a single file, which contains the mapping (i.e. *links*) between POI entities from both input POI datasets. The output of LIMES is essential for running the next SLIPO tools (i.e., *DEER* and *LIMES*). Figure 2 shows an over view of the SLIPO architecture.



(a) High level architecture (b) Logical architecture

Figure 2: SLIPO Architecture

1.3.1. Achievements - LIMES v1.0.0

LIMES v1.0.0 is the first version of LIMES that has been developed in the context of the SLIPO project and focuses on POI-specific interlinking. One of the major goals of the project is to abstract as much complexity as possible from the end users of the SLIPO Workbench. So, in order to keep user interaction at a minimum and requiring no knowledge of Linked Data technologies and concepts, we aimed at adapting and fine-tuning LIMES's functionality specifically for POI data, as well as at automating the interlinking process as much as possible. To this end, we emphasized on the development of the backend of the platform, aiming to enrich and specialize the core linking functionality of the framework. Next, we enumerate the new features and functionality of LIMES, as a result of our work during the first 18 months of SLIPO.

- **POI-specific point-set distance.** New point-set distances based on the vector representations of the POI resources (e.g., Hausdorff, mean, surjection and sumOfMin). Altogether, we implemented a set of 10 point-set distance functions based on our survey published on [18].
- **Topological relation discovery** based on the vector representations of the POI resources (e.g. one POI resource contains/crosses/touches another POI resource). For instance, find all the parking locations within shopping malls. Our novel algorithm, RADON [21], for rapid discovery of topological relation among POI resources with 2D geometries. Our evaluation prove that RADON is able to overperform state-of-the-art approaches up to 3 orders of magnitude while maintaining a precision and a recall of one.
- **Temporal relation discovery** based on the temporal timestamps within the POI resources (e.g., one POI take place after/before/during another POI). For example, a specific area is used as a parking location only during a football match. We were able to propose AEGLE [19], a novel approach for the efficient computation of links between POIs' temporal representations according to Allen's interval algebra. Our evaluations of the runtime of AEGLE show that AEGLE outperforms the state of the art by up to 4 orders of magnitude while maintaining a precision and a recall of one.
- **Combining the new techniques with the one already in LIMES.** In LIMES v.1.0.0 we integrated the novel algorithms for the 10 POI-point-set distances as well as RADON and AEGLE into the LIMES core. In particular, a new mapper is implemented for each of the new relation types. Such mappers are combined with the already existing mappers for efficient link discovery of the new types of relation specific for POI resources.
- **Integration with the SLIPO Workbench**. LIMES v.1.0.0 realizes two deployment modes: (a) standalone, as an individual software that accepts as input linked POI datasets and provides as output a mapping file contains the links between the input POI datasets; (b) deployment within the SLIPO Workbench, where LIMES serves as an integral component of the *SLIPO Toolkit* and is loosely integrated by the SLIPO Workbench with the other software components into forming POI integration workflows.
- **Scalability**. In LIMES v1.0.0 we have implemented a simple, yet efficient in our setting, distributed execution scheme, that functions independently of core interlinking of LIMES. Specifically, we have 2 implementations based on SPARK and FLIK frameworks. Currently, we run an intensive evaluation for both frameworks to find the pros and cons of each.

A detailed presentation of LIMES is provided in the following Section.

2. The LIMES Framework

LIMES, the **L**ink Discovery Framework for **Me**tric **S**paces, is a framework for discovering links between entities contained in Linked Data sources. LIMES is a hybrid framework that combines the mathematical characteristics of metric spaces as well prefix-, suffix- and position filtering to compute pessimistic approximations of the similarity of instances. These approximations are then used to filter out a large amount of those instance pairs that do not suffice the mapping conditions. By these means, LIMES can reduce the number of comparisons needed during the mapping process by several orders of magnitude and complexity without losing a single link.

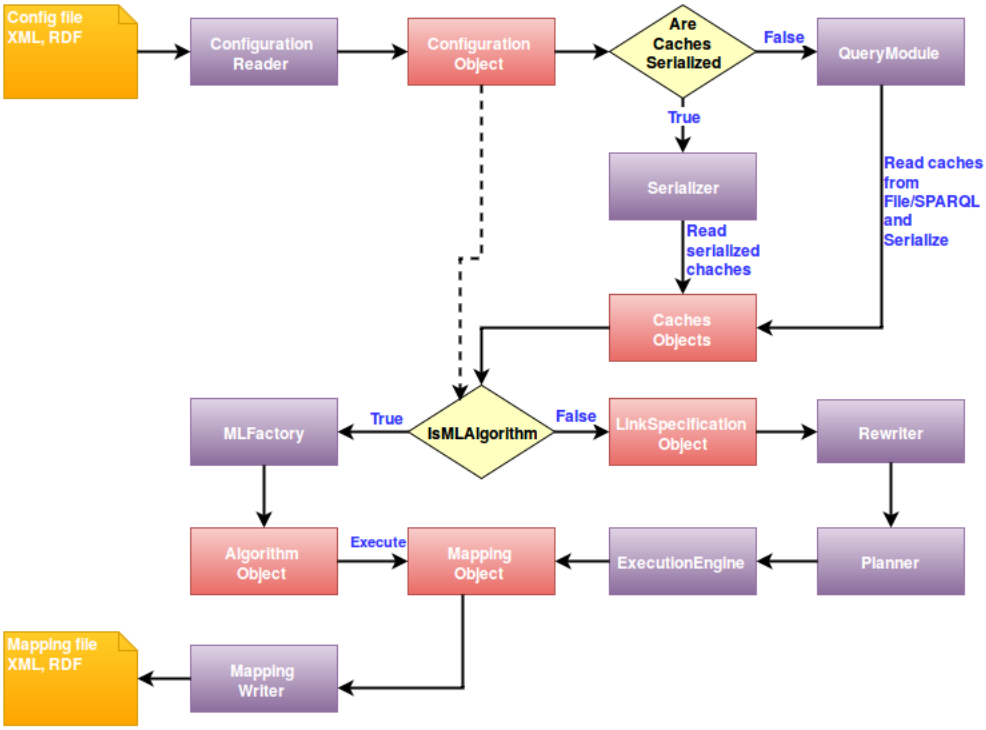


Figure 3: General architecture of LIMES

As shown in Figure 3, the LIMES framework consists of eight main modules of which each can be extended to accommodate new or improved functionality. The central module of LIMES is the **controller** module, which coordinates the matching process. The matching process is carried out as follows: First, the **controller** calls the **configuration** module, which reads the configuration file and extracts all the information necessary to carry out the comparison of instances, including the URL of the SPARQL-endpoints of source (S) and the target (T) knowledge bases, the restrictions on the instances to map (e.g., their type), the expression of the metric to be used and the threshold to be used.

Given that the configuration file is valid w.r.t. the **LIMES Specification Language (LSL)**, the **query** module is called. This module uses the configuration for the target and source knowledge bases to retrieve instances and properties from the SPARQL-endpoints of the source and target knowledge bases that adhere to the

restrictions specified in the configuration file. The query module writes its output into a file by invoking the **cache** module. Once all instances have been stored in the cache, the controller chooses between performing Link Discovery or Machine Learning. For Link Discovery, LIMES will re-write, plan and execute the Link Specification (LS) included in the configuration file, by calling the **rewriter**, **planner** and **engine** modules respectively. The main goal of LD is to identify the set of links (mapping) that satisfy the conditions opposed by the input LS. For Machine Learning, LIMES calls the **machine learning** algorithm included in the configuration file, to identify an appropriate LS to link S and T. Then it proceeds in executing the LS. For both tasks, the mapping will be stored in the output file chosen by the user in the configuration file. The results are finally stored into an RDF or a XML file.

The advantages of LIMES are manifold. First, it implements **highly time-optimized** mappers, making it a complexity class faster than other Link Discovery Frameworks. Thus, the larger the problem, the faster LIMES is w.r.t. other Link Discovery Frameworks. Secondly, LIMES supports a large set of string, numeric, topological and temporal similarity metrics, that provide the user with the opportunity to perform various comparisons between resources. In addition, **LIMES is guaranteed to lead to exactly the same matching as a brute force approach while at the same time reducing significantly the number of comparisons**. In addition, LIMES supports a **large number of input and output formats** and can be extended very easily to fit new algorithms, new datatypes, and new pre-processing functions, thanks to its modular architecture.

In general, LIMES can be used to set links between two data sources, e.g., a novel data source created by a data publisher and existing data source such as *DBpedia*. This functionality can also be used to detect duplicates within one data source for knowledge curation. The only requirement to carry out these tasks is a simple XML-based or TURTLE-based configuration file.

2.1. LIMES v0.6.0 to v1.0.0

The initial version of LIMES (v0.6.0) allows the interlinking of RDF resources from the same or different knowledgebases. LIMES v0.6.0 was not able to handle the new 5D representations of POI datasets. i.e., it was not able to deal with spatial, temporal and granularity-based representations of resources such as the ones in current POI data sets. In particular, LIMES v0.6.0 was not able to discover point-set (e.g., as *Hausdorff* distance) and topological relations (e.g., *CoveredBy* relation). Also, LIMES v0.6.0 was not able to discover temporal relations such as the concurrency of two events. In order to enable the handling of 5D POI datasets, in the current version of LIMES (v1.0.0) we extended LIMES within the SLIPO project to include:

- **New point-set distances** based on the vector representations of the POI resources (e.g. Hausdorff, mean, surjection and sumOfMin). Altogether, we implemented a set of 10 point-set distance function based on our survey published on [18].
- **Topological relation discovery** based on the vector representations of the POI resources (e.g. contains, disjoints, crosses and touches). Our proposed algorithm, RADON [21], outperforms the state of the art by up to three orders of magnitude.
- **Temporal relation discovery** based on the temporal timestamps within the POI resources (e.g., after, before, during). We were able to propose an efficient algorithm (AEGLE [19]) for the discovery of such relations based on *Allen algebra*.

In addition to the new measures, we divided the old LIMES project into limes-core and limes-gui in order to realize a more efficient LIMES installation process. Moreover, in LIMES v1.0.0 we introduced LIMES HTTP server mode. Finally, we fixed several bugs and carried out many performance optimizations improvements.

2.2. Towards LIMES v2.0.0

In this section, we describe the planned extensions on LIMES in the context of the SLIPO project, towards evolving it into a platform for efficiently and effectively interlinking large datasets of linked POI entities. Towards this, we have followed our original development roadmap (see D1.2 Architecture, Section 2.3.2), where our final version of the interlinking software will be LIMES v2.0.0. Within this deliverable, we introduce LIMES v1.0.0 as an interim version of the software. This reflects the dynamic development of the software and the multitude advances we have managed to introduce already in the project's lifecycle. Overall, the proposed extensions focus on the following development pathways.

- **Linking of 5D POI Resources**. Using the LIMES framework, we will harness the novel paradigm of 5D modelling for POIs by dealing with 2D geospatial coordinates, time and scale (*also known as Level of Detail, resolution or granularity*). The fifth dimension, granularity, will be employed for the optimization of the spatial interlinking approaches implemented in the LIMES framework.

  In the current release, LIMES v1.0.0 is able to discover new types of relations based on the 5D modelling of the POI resources. In particular, using the coordinates of resources, LIMES is now able to discover proximity of POI resources in the map as well as topological relation among them. For example, LIMES can discover the nearest gas station for each hotel in the given datasets. Dealing with granularity of data is still a current work under development.
- **Learning of class-expression-specific specifications**. In this line of work, we aim at discovering POI specific relations. In particular, the learning of class-expression-specific specifications will allow for the tuning of geospatial proximity thresholds based on POI types/categories. An initial work is currently under development. We expect to have an initial result by M24 of the SLIPO project.
- **Temporal Relations Discovery**. In the current version, we extended the scalable approaches for *Euclidean* spaces implemented in LIMES to the discovery of temporal relations (according to *Allen's* interval algebra and extensions thereof) between POIs such as for example temporal overlaps between POIs. In [19], we introduced AEGLE, a novel approach for the efficient computation of links between events according to *Allen's* interval algebra. We studied Allen's relations and showed that we can reduce all thirteen relations to eight simpler relations. We then present an efficient algorithm with a complexity of $O(n \log n)$ for computing these eight relations. Our evaluation of the runtime of our algorithms shows that we outperform the state of the art by up to four orders of magnitude while maintaining a precision and a recall of 1.

  In LIMES v2.0.0, we will extend the scalability of our approach by providing dedicated solutions for load balancing within a parallel execution setting. Moreover, we will study the incremental computation of temporal links on streams of data.
- **Combining the new techniques in LIMES**. In LIMES v2.0.0, we will aim more optimize the new functionalities with the other mappers within LIMES using set theory operators to allow for hybrid similarity functions and the configurable weighting of specific POI metadata. In particular, we will

apply load-balancing approaches for better usage of available resources as well machine learning techniques for minimizing the configurations effort of LIMES' users.

- **Scalability**. In LIMES, we aim to develop scalable approaches for de-duplicating and interlinking massive, heterogeneous, and incomplete POI data at a world-scale. These will be scalable to handling billions of RDF triples and optimized according to the developed schema and ontology for managing POI data. We will also tackle the challenges arising from the multiple and inherent sources of ambiguity in POI data (spatial, temporal, semantic), as well as the inherent multi-linguality of location-related information. Scalability in linking 5D POI resources will be achieved as follows: By computing the error engendered by the usage of low-resolution data, we will be able to discard a large number of similarity computations when running our interlinking approaches and therewith achieve significantly better scalability than the state of the art. We will aim at achieving interlinking of 10 Million POIs in less than 10 minutes.

  Currently, we develop two main implementations of the core algorithms in LIMES using the FLINK and SPARK frameworks for Big Data processing. Our preliminary experiments promise a huge speedup when dealing with big POI datasets.

2.3. Libraries and Frameworks

LIMES has dependencies to the following open-source tools/libraries:

- Apache Jena[2]: A Java framework for building Semantic Web applications.
- Java Topology Suite[3]: An API of 2D spatial predicates and functions conforming to the OGC Simple Features Specification for SQL.
- Google Guava[4]: A set of libraries that includes collection types (such as multimap and multiset), immutable collections, a graph library, functional types, an in-memory cache, and APIs/utilities for concurrency, I/O, hashing, primitives, reflection and string processing.
- Apache Commons Text[5]: A library focused on algorithms working on strings.
- Apache Commons Lang[6]: Provides a host of helper utilities for the java.lang API, notably String manipulation methods, basic numerical methods, object reflection, concurrency, creation and serialization and System properties. Additionally it contains basic enhancements to java.util.Date and a series of utilities dedicated to help with building methods, such as hashCode, toString and equals.
- Google code json-simple[7]: A simple Java toolkit for JSON. It is used to encode or decode JSON text.

Other libraries include:

- algorithms.edjoin

[2] https://jena.apache.org/

[3] https://github.com/locationtech/jts

[4] https://github.com/google/guava

[5] https://commons.apache.org/proper/commons-text/

[6] https://commons.apache.org/proper/commons-lang/

[7] https://code.google.com/archive/p/json-simple/

- commons-fileupload
- commons-lang
- com.googlecode.lanterna
- com.ibm.icu
- com.vividsolutions
- eu.medsea.mimeutil
- fr.ign.cogit
- jgraphx
- junit
- net.sf.ehcache
- net.sf.jgap
- nz.ac.waikato.cms.weka
- org.aksw.jena-sparql-api
- org.apache.logging.log4j
- org.apache.maven.plugins
- org.fusesource.jansi
- uk.ac.shef.wit

2.4. License

LIMES is an open source software and is available from GitHub[8] under the terms of the GNU AFFERO GENERAL PUBLIC LICENSE[9].

2.5. Documentation

For a detailed documentation of LIMES, one can consult the following resources:

- LIMES website: http://cs.uni-paderborn.de/ds/research/research-projects/active-projects/limes/
- User manual: http://dice-group.github.io/LIMES/user_manual/
- Developer manuals: http://dice-group.github.io/LIMES/developer_manual/
- Online DEMO: http://limes-webui.aksw.org/

[8] https://github.com/dice-group/LIMES/blob/master

[9] https://github.com/dice-group/LIMES/blob/master/LICENSE

3. Usage Manual for LIMES

In this Section, we provide the usage manual for LIMES v1.0.0. First, we give details on building the application from the Java source code. Next, we provide instructions on both the manual- and machine-learning-based- configuration of LIMES. Finally, we present a short demonstration example on configuring and running LIMES.

3.1. Building Installation

LIMES v1.0.0 is publicly available (see https://github.com/SLIPO-EU/limes), offering the entire Java source code as well as indicative configurations. Java SDK 1.8[10] (or later) as well as Maven 3.3.3[11] (or later) must be installed and properly configured in order to compile and execute DEER. The `pom.xml` file contains the project's configuration in Maven and has been successfully tested in Mac OS, Microsoft Windows and Linux environments. The following building instructions assume that Git is also installed.

3.1.1. Generating Jar File (Headless)

In order to build the command line version from source, first the master branch of LIMES must be cloned to a preferred location by running:

```
git clone -b master --single-branch https://github.com/SLIPO-EU/LIMES.git LIMES
```

It is recommended to use the `--single-branch` parameter to save some time and avoid pulling the whole history of the project.

Then, from the root directory of the project (LIMES) the following command needs to be executed:

```
mvn clean install
```

Creating the runnable jar file including the dependencies use:

```
mvn clean package shade:shade -Dcheckstyle.skip=true -Dmaven.test.skip=true
```

After a successful installation, a target directory should have been created containing the `LIMES-VERSION-SNAPSHOT.jar` (version depending on POM configuration).

3.1.2. Generating Jar File (GUI)

Optionally, LIMES provides a graphical user interface (GUI) which helps the user build complicated link specifications without having to manually define rules in XML files. The GUI installation assumes that the command line version is already installed and uses the command-line version as a library.

Similarly, the LIMES-web repository needs to be cloned to a preferred location on your system by running:

```
git clone -b master --single-branch https://github.com/SLIPO-EU/LIMES.git LIMES
```

[10] http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

[11] https://maven.apache.org/docs/3.3.3/release-notes.html

Switch to `limes-gui` and use:

```
mvn jfx:jar -Dcheckstyle.skip=true -Dmaven.test.skip=true
```

The `.jar` will be placed in `limes-gui/target/jfx/app/limes-GUI-VERSION-SNAPSHOT-jfx.jar`

The `limes-gui/target/jfx/app/lib` folder needs to be in the same folder as the `.jar` for the `.jar` to work.

3.2. Configuration Settings

A LIMES configuration file consists of ten parts, some of which are optional.

3.2.1. Metadata

The `metadata` tag always consists of the following bits of XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE LIMES SYSTEM "limes.dtd">
<LIMES>
```

3.2.2. Prefixes

Defining a prefix in a LIMES file demands setting two values: The `namespace` that will be addressed by the prefix's `label`

```
<PREFIX>
    <NAMESPACE>http://www.w3.org/1999/02/22-rdf-syntax-ns#</NAMESPACE>
    <LABEL>rdf</LABEL>
</PREFIX>
```

Here, we set the prefix `rdf` to correspond to `http://www.w3.org/1999/02/22-rdf-syntax-ns\.` A LIMES link specification can contain as many prefixes as required.

3.2.3. Data Sources

LIMES computes links between items contained in two Linked Data sources, dubbed source and target. Both source and target need to be configured using the same tags. An example of a configuration for both data sources is shown below.

```
<SOURCE>
    <ID>mesh</ID>
    <ENDPOINT>http://mesh.bio2rdf.org/sparql</ENDPOINT>
    <VAR>?y</VAR>
    <PAGESIZE>5000</PAGESIZE>
    <RESTRICTION>?y rdf:type meshr:Concept</RESTRICTION>
    <PROPERTY>dc:title</PROPERTY>
    <TYPE>sparql</TYPE>
</SOURCE>
```

```
<TARGET>
    <ID>linkedct</ID>
    <ENDPOINT>http://data.linkedct.org/sparql</ENDPOINT>
    <VAR>?x</VAR>
    <PAGESIZE>5000</PAGESIZE>
    <RESTRICTION>?x rdf:type linkedct:condition</RESTRICTION>
    <PROPERTY>linkedct:condition_name</PROPERTY>
</TARGET>
```

Six properties need to be set.

Each data source must be given an ID via the tag `ID`.

The endpoint of the data source needs to be explicated via the `ENDPOINT` tag.

If the data is to be queried from a SPARQL end point, the `ENDPOINT` tag must be set to the corresponding SPARQL endpoint URI.

In case the data is stored in a local file (CSV, N3, TURTLE, etc.), `ENDPOINT` tag must be set to the absolute path of the file containing the data.

The `VAR` tag describes the variable associated with the aforementioned endpoint. This variable is also used later, when specifying the metric used to link the entities retrieved from the source and target endpoints.

The fourth property is set via the `PAGESIZE` tag. This property must be set to the maximal number of triples returned by the SPARQL endpoint. For example, the *DBpedia endpoint*[12] returns a maximum of 1000 triples for each query. If `PAGESIZE` tag is set, the SPARQL query module in LIMES will be able to retrieve all relevant instances by iteratively asking the SPARQL end point for the relevant instances using the *offset* and *limit* features of the SPARQL query language. If the SPARQL endpoint does not limit the number of triples it returns or if the input is a file, the value of `PAGESIZE` should be set to -1.

The restrictions on the queried data can be set via the `RESTRICTION` tag. This tag allows to constrain the entries that are retrieved by the LIMES' query module. In this particular example, we only use instances of MESH concepts. Additionally, multiple `RESTRICTION` tags are allowed per data source.

The `PROPERTY` tag allows to specify the properties that will be used during the linking. It is important to note that the property tag can also be used to specify the pre-processing on the input data. For example, setting `rdfs:label AS nolang`, one can ensure that the language tags get removed from each `rdfs:label` before it is written in the cache. Pre-processing functions can be piped into one another by using `->`. For example, `rdfs:label AS nolang->lowercase` will compute `lowercase(nolang(rdfs:label))`. If you are not sure if all your entities have a certain property you can use the `OPTIONAL_PROPERTY` tag instead of `PROPERTY`. Additionally, multiple `PROPERTY` tags are allowed per data source.

Additionally, optional properties can be set to segment the requested dataset:

The graph of the endpoint can be specified directly after the `ENDPOINT` tag using the `GRAPH` tag.

The limits of the query can be set with the `MINOFFSET` and `MAXOFFSET` tags directly after the `PAGESIZE` tag. The resulting query will ask about the statements in the interval [`MINOFFSET`, `MAXOFFSET`]. Note that

[12] http://dbpedia.org/sparql

`MINOFFSET` must be smaller than `MAXOFFSET`! If both `SOURCE` and `TARGET` are restricted, a warning is generated.

3.2.3.1. Pre-processing Functions

Currently, LIMES supports the following set of pre-processing functions:

- `noLang` for removing language tags
- `lowerCase` for converting the input string into lower case
- `upperCase` for converting the input string into upper case
- `number` for ensuring that only the numeric characters, "." and "," are contained in the input string
- `replace(String a,String b)` for replacing each occurrence of `a` with `b`
- `regexReplace(String x,String b)` for replacing each occurrence the regular expression `x` with `b`
- `cleanIri` for removing all the prefixes from IRIs
- `celsius` for converting Fahrenheit to Celsius
- `fahrenheit` for converting Celsius to Fahrenheit
- `removeBraces` for removing the braces
- `regularAlphabet` for removing nun-alphanumeric characters
- `uriAsString` returns the last part of an URI as a String.

Sometimes, generating the right link specification might either require merging property values (for example, the `dc:title` and `foaf:name` of MESH concepts) or splitting property values (for example, comparing the label and `foaf:homepage` of source instances and the `foaf:homepage` of target instances as well as `foaf:homepage AS cleanIri` of the target instances with the `rdfs:label` of target instances. To enable this, LIMES provides the `RENAME` operator which simply stores either the values of a property or the results of a pre-processing function into a different property field. For example, `foaf:homepage AS cleanIri RENAME label` would store the homepage of an object without all the prefixes in the name property. The user could then access this value during the specification of the similarity measure for comparing source and target instances. Note that the same property value can be used several times. Thus, the following specification fragment is valid and leads to the `dc:title` and `foaf:name` of individuals) of MESH concepts being first cast down to the lowercase and then merged to a single property.

```
<SOURCE>
    <ID>mesh</ID>
    <ENDPOINT>http://mesh.bio2rdf.org/sparql</ENDPOINT>
    <VAR>?y</VAR>
    <PAGESIZE>5000</PAGESIZE>
    <RESTRICTION>?y rdf:type meshr:Concept</RESTRICTION>
    <PROPERTY>dc:title AS lowercase RENAME name</PROPERTY>
    <PROPERTY>foaf:name AS lowercase RENAME name</PROPERTY>
    <TYPE>sparql</TYPE>
</SOURCE>
```

In addition, the following allows splitting the values of `foaf:homepage` into the property values `name` and `homepage`.

```
<SOURCE>
```

```
    <ID>mesh</ID>
    <ENDPOINT>http://mesh.bio2rdf.org/sparql</ENDPOINT>
    <VAR>?y</VAR>
    <PAGESIZE>5000</PAGESIZE>
    <RESTRICTION>?y rdf:type meshr:Concept</RESTRICTION>
    <PROPERTY>foaf:homepage AS lowercase RENAME homepage</PROPERTY>
    <PROPERTY>foaf:homepage AS cleaniri->lowercase RENAME name</PROPERTY>
    <TYPE>sparql</TYPE>
</SOURCE>
```

In addition, a source type can be set via `TYPE`. The default type is set to `SPARQL` (for a SPARQL endpoint) but LIMES also supports reading files directly from the hard-drive. The supported data formats are:

- `CSV:` Character-separated file can be loaded directly into LIMES. Note that the separation character is set to `TAB` as a default. The user can alter this setting programmatically.
- `N3` (which also reads `NT` files) reads files in the `N3` language.
- `N-TRIPLE` reads files in W3C's core N-Triples format[13].
- `TURTLE` allows reading files in the `Turtle` syntax[14].

Moreover, if you want to download data from a SPARQL endpoint, there is no need to set the `<TYPE>` tag. Instead, if you want to read the source (or target) data from a file, you should fill `<ENDPOINT>` tag with the absolute path of the input file, e.g. `<ENDPOINT>C:/Files/dbpedia.nt</ENDPOINT>`, and you should also set the `<TYPE>` tag with the type of the input data, for example `<TYPE>NT</TYPE>`.

3.2.4. Metric Expression for Similarity Measurement

One of the core improvements of the newest LIMES kernels is the provision of a highly flexible language for the specification of complex metrics for linking (set by using the `METRIC` tag as exemplified below).

```
<METRIC>
    trigrams(y.dc:title, x.linkedct:condition_name)
</METRIC>
```

In this example, we use the `trigrams` metric to compare the `dc:title` of the instances retrieved from the *source* data source (with which the variable `y` is associated) with the `linkedct:condition` (with which the variable `x` is associated). While such simple metrics can be used in many cases, complex metrics are necessary in complex linking cases. LIMES includes a formal grammar for specifying complex configurations of arbitrary complexity. For this purpose, two categories of binary operations are supported: Metric operations and Boolean operations.

3.2.4.1. Metric operations

Metric operations allow to combine metric values. They include the operators `MIN`, `MAX` and `ADD` e.g. as follows:

```
MAX(trigrams(x.rdfs:label,y.dc:title)|0.3,
    euclidean(x.lat|long,y.latitude|longitude)|0.5).
```

[13] http://www.w3.org/TR/rdf-testcases//#ntriples

[14] http://www.w3.org/TR/turtle

This specification computes the maximum of:

1. The trigram similarity of x's `rdfs:label` and y's `dc:title` filtered by 0.3. i.e. LIMES only returns resources with trigram similarities above or equal 0.3.
2. The 2-dimension Euclidean distance of `x`'s `lat` and `long` with `y`'s `latitude` and `longitude`, i.e., `((x.lat-y.latitude)2+(x.long-y.longitude)2) \sqrt{((x.lat- y.latitude)^2 + (x.long - y.longitude)^2)} √((x.lat-y.latitude)2+(x.long-y.longitude)2.` Note that, Like in the case of trigrams, LIMES filter the results for Euclidian similarities greater or equal to 0.5.

Note that the Euclidean distance supports arbitrarily many dimensions. In addition, note that ADD allows to define weighted sums as follows:

```
ADD(0.3*trigrams(x.rdfs:label,y.dc:title)|0.3,
    0.7*euclidean(x.lat|x.long,y.latitude|y.longitude)|0.5).
```

We call `trigrams(x.rdfs:label,y.dc:title)|0.3` the left child of the specification and `euclidean(x.lat|long, y.latitude|longitude)|0.5` the right child of the specification. Both children specifications are simple specifications and combined with a metric operator, they create a complex specification. LIMES gives the user the opportunity to combine **exactly two LSs** (complex or simple) in order to create a new complex LS.

3.2.4.2. Boolean operations

Boolean operations allow to combine and filter the results of metric operations and include `AND`, `OR`, `DIFF`, e.g. as

```
AND(trigrams(x.rdfs:label,y.dc:title)|0.9,
    euclidean(x.lat|x.long,y.latitude|y.longitude)|0.7).
```

This specification returns all links such that:

1. the trigram similarity of `x`'s `rdfs:label` and `y`'s `dc:title` is greater or equal to 0.9 and
2. the 2-dimension Euclidean distance of `x`'s `lat` and `long` mit y's `latitude` and `longitude` is greater or equal to `0.7`.

We call `trigrams(x.rdfs:label,y.dc:title)|0.9` the left child of the specification and `euclidean(x.lat|x.long, y.latitude|y.longitude)|0.7` the right child of the specification. Both children specifications are simple specifications and combined with a Boolean operator, they create a complex specification. LIMES gives the user the opportunity to combine **exactly two** complex or simple speciation's to create a new complex specification. Note that each child specification must be accompanied by its own threshold.

3.2.4.3. Measure Packages

Measures are organized in packages, based on the type of resource they are designed to operate with. Several measure packages ship with LIMES, while and it is easy to extend it with custom packages from third parties.

The current version of LIMES ships with the following measure packages include:

1. *String measures* for computing the similarity values among string representations of POI resources. For example, string measures can be used to compare name of POI resources modelled using `rdf:label`.
2. *Vector space measures* for computing the proximity among point representations of POI resources. For example, find the nearest bus station for each hospital.
3. Point-set *measures* for computing the distance among point-set representations of POI resources. For example, it can be used to find the nearest hospital for each kindergarten.
4. Topological *measures* for computing the topological relation among geospatial representations of POI resources. For example, it can be used to find car park placed which located within shopping malls.
5. Temporal *measures* for computing the temporal relations among time stamps of POI resources. For example, find restaurants with the same opening hours.

More complex distance measures are being added continuously. We give more details about each of the measure type in the following sections.

3.2.4.3.1. String Measures

The string measures package consists of the following measures:

`Cosine:` Cosine string similarity is a measure of similarity between two non-zero vectors representations of the two input strings of an inner product space that measures the cosine of the angle between them. The outcome of the Cosine string similarity is neatly bounded in [0,1]

`ExactMatch:` Exact match string similarity is a measure of similarity between two input strings that returns one in case the two input strings were identical, zero otherwise.

- `Jaccard:` The *Jaccard index*, also known as Intersection over Union and the *Jaccard similarity coefficient* (originally coined coefficient de communauté by Paul Jaccard), is a statistic used for comparing the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets. In LIMES, we use trigrams of the input strings to generate our sample sets.
- `Overlap:` The *overlap coefficient* or *Szymkiewicz–Simpson coefficient*, is a similarity measure that measures the overlap between two sets. It is related to the *Jaccard index* and is defined as the size of the intersection divided by the smaller of the size of the two sets.
- `Jaro:` The *Jaro distance* between two strings is the minimum number of single-character transpositions required to change one string into the other.
- `JaroWinkler:` The *Jaro–Winkler distance* is a string metric for measuring the edit distance between two sequences. It is a variant proposed in 1990 by William E. Winkler of the Jaro distance metric. The Jaro–Winkler distance uses a prefix scale which gives more favourable ratings to strings that match from the beginning for a set prefix length. The lower the Jaro–Winkler distance for two strings is, the more similar the strings are. The score is normalized such that 0 equates to no similarity and 1 is an exact match. The *Jaro–Winkler similarity* in LIMES is given by 1 / (1 – Jaro–Winkler distance).

- `Levenshtein:` The *Levenshtein distance* is a string metric for measuring the difference between two strings. Informally, the *Levenshtein distance* between two strings is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other. It is named after the Soviet mathematician Vladimir Levenshtein, who considered this distance in 1965. Normalized *Levenshtein distance* is computed by dividing the *Levenshtein distance* by the length of the input string. The *Levenshtein similarity* in LIMES is given by 1 / (1 – normalized-*Levenshtein distance*).
- `MongeElkan:` The *Monge-Elkan similarity* measure is a type of hybrid similarity measure that combines the benefits of sequence-based and set-based methods. This can be effective for domains in which more control is needed over the similarity measure. In LIMES, we use trigrams of the input strings to generate our sample string subsets. In LIMES, we use trigrams of the input strings to generate our sample sets.
- `RatcliffObershelp` In Ratcliff/Obershelp, we compute the similarity of the two input strings as the number of matching characters divided by the total number of characters in the two strings. Matching characters are those in the longest common subsequence plus, recursively, matching characters in the unmatched region on either side of the longest common subsequence.
- `Soundex:` Soundex is a phonetic algorithm for indexing names by sound, as pronounced in English. The goal is for homophones to be encoded to the same representation so that they can be matched despite minor differences in spelling. The algorithm mainly encodes consonants, a vowel will not be encoded unless it is the first letter. In LIMES, we compute the Soundex distance as the reverse of the distance between the encoding of the two input strings.
- `Trigram`: A tri-gram is a group of three consecutive characters taken from a string. In LIMES, we measure the similarity of two input strings by counting the number of trigrams they share. Formally, we compute the trigram similarity as the normalized sum of absolute differences between tri-gram vectors of both the input strings.
- `Qgrams:` Same as trigram but using a group of four three consecutive characters for generating the q-gram vectors of the input strings.

Below, an example of an atomic LS that consists of the string measure `Trigram` and a threshold `theta = 0.8` is given:

```
trigram(x.label, y.title) | 0.8
```

where `label` and `title` are properties of the source and target KB respectively, whose values are strings.

3.2.4.3.2. Vector Space Measures

LIMES supports comparing numeric vectors representations of POI resources by using the vector space measures package consisting of the following measures:

- `Euclidean` *Euclidean* metric is the straight-line distance between two points in *Euclidean* space. With this distance, Euclidean space becomes a metric space. For example:

```
euclidean(a.wgs84:lat|wgs84:long,b.wgs84:lat|wgs84:long)
```

will compute the *Euclidean distance* between the point representations of each resource from the source and target datasets.

- `Geo_Orthodromic` The *great-circle distance* or *orthodromic distance* is the shortest distance between two points on the surface of a sphere, measured along the surface of the sphere (as opposed to a straight line through the sphere's interior). The distance between two points in *Euclidean* space is the length of a straight line between them, but on the sphere, there are no straight lines. In spaces with curvature, straight lines are replaced by geodesics. Geodesics on the sphere are circles on the sphere whose centres coincide with the canter of the sphere and are called great circles.
- `Geo_Great_Elliptic` The great ellipse distance is the length of the ellipse passing through two points on a spheroid and having the same centre as that of the spheroid. Equivalently, it is distance of the ellipse on the surface of a spheroid and cantered on the origin, or the curve formed by intersecting the spheroid by a plane through its centre. The great ellipse distance is confedered the most accurate distance between two point In the surface of the earth.

3.2.4.3.3. Point-Set Measures

The similarity between POI geometries can be measured by using the following point-set distances:

- `Geo_Hausdorff` The *Hausdorff* distance is a measure of the maximum of the minimum distances between pairwise points in the two input geometries.
- `Geo_Max` The idea behind this measure is to compute the overall maximal distance between pairwise points of the two input geometries.
- `Geo_Min` The idea of `Geo_Min` is akin to that of `Geo_Max` and is defined as minimal distance between pairwise points of the two input geometries.
- `Geo_Mean` The mean distance is one of the most efficient distance measures for point sets. First, a mean point is computed for each point set. Then, the distance be- tween the two means is computed by using the orthodromic distance.
- `Geo_Avg` For computing the average point sets distance function, the orthodromic distance measures between all the source-target geometries' points pairs is cumulated and divided by the number of points in the source-target geometries' point pairs.
- `Geo_Frechet` The Fréchet distance is a measure of similarity between curves (in our case geometries representations of the POI resources) that takes into account the location and ordering of the points along the curves.
- `Geo_Sum_Of_Min` First, the closest point from the source geometry to each point to the target geometry is computed. The same operation is carried out with source and target reversed. Finally, the average of the two values is then the distance value.
- `Geo_Naive_Surjection` The surjection distance function introduced defines the distance between two geometries as the minimum distance between the sum of distances of the surjection of the larger set to the smaller one. A main drawback of the surjection is being biased toward some points ignoring some others in calculations.
- `Geo_Fair_Surjection` In order to fix the bias of the `Geo_Naive_Surjection`, the fair-surjection distance maps the elements of source geometry as evenly as possible to the elements of the target geometry.

- `Geo_Link The Link` distance is defined as the minimum orthodromic distance between pairwise points of the source and target geometries that satisfy the bijection relation.

3.2.4.3.4. Topological Measures

The topological relations between spatial representations of POI resources can be found by using the following relations. In the following relations we assume that the first POI resource has a geospatial representation in a form of geometry *a* and the target POI resource has a geospatial representation in a form of geometry *b*:

- `Top_Contains` A geometry *a* contains geometry *b* if and only if no points of b lie in the exterior of a, and at least one point of the interior of b lies in the interior of a.
- `Top_Covers` A geometry *a* covers geometry *b* if and only if the geometry *b* lies in *a*. i.e. No points of *b* lie in the exterior of *a*, or Every point of *b* is a point of (the interior or boundary of) *a*.
- `Top_Covered_By` A geometry *a* is covered by a geometry *b* if and only if every point of *a* is a point of *b*, and the interiors of the two geometries have at least one point in common. Note that `Top_Covered_By` is the reverse relation of `Top_Covers`.
- `Top_Crosses` A geometry *a* crosses a geometry *b* if and only if they have some but not all interior points in common, and the dimension of the intersection is less than that of at least one of them.
- `Top_Disjoint` Two geometries *a* and *b* are disjoint if and only if they have no point in common. They form a set of disconnected geometries.
- `Top_Equals` Two geometries *a* and *b* are topologically equal if their interiors intersect and no part of the interior or boundary of one geometry intersects the exterior of the other.
- `Top_Intersects` A geometry *a* intersects A geometry *b* if and only if geometries a and b have at least one point in common.
- `Top_Overlaps` A geometry *a* overlaps a geometry *b* if and only if they have some but not all points in common, they have the same dimension and the intersection of the interiors of the two geometries has the same dimension as the geometries themselves
- `Top_Touches` Two geometries *a* and touched if they have at least one boundary point in common, but no interior points.
- `Top_Within` A geometry *a* is within a geometry *b* if and only if a lies in the interior of b.

3.2.4.3.5. Temporal Measures

The temporal relations between POI resources can be found by using the following relations:

- `Tmp_Concurrent`: given a source and a target KB, `Tmp_Concurrent` links the source and the target events that have the same begin date and were produced by the same machine. For example: `Tmp_Concurrent(x.beginDate1|machine1,y.beginDate2|machine2)|1.0`
- `Tmp_Predecessor:` given a source and a target KB, `vmp_Predecessor` links the source events to the set of target events that happen exactly before them. For example: `Tmp_Predecessor (x.beginDate1, y.beginDate2)|1.0`. If the `Tmp_Predecessor` measure is used in a complex LS, the `CANONICAL` planner should be used.

- `Tmp_Successor`: given a source and a target KB, `Tmp_Successor` links the source events to the set of target events that happen exactly after them. For example: Tmp_Successor `(x.beginDate1, y.beginDate2)|1.0`. If the `Tmp_Successor` measure is used in a complex LS, the `CANONICAL` planner should be used.

Moreover, LIMES support the following temporal relations between POI resources based on *Allen's* algebra:

- `Tmp_After` The first POI takes place *after* the second POI takes place.
- `Tmp_Before` The first POI takes place *before* the second POI takes place.
- `Tmp_During` The first POI take place *during* the second POI takes place.
- `Tmp_During_Reverse` The second POI take place *during* the first POI takes place. `Tmp_During_Reverse` is the reverse of `Tmp_During`
- `Tmp_Equals` Both first POI and the second take place concurrently. i.e. both POIs have *equal* timestamp.
- `Tmp_Finishes` The first POI *finishes in the same time* as the second POI finishes.
- `Tmp_Is_Finished_By` reverse of `Tmp_Finishes`
- `Tmp_Overlaps` Part of the first POI timestamp *overlaps* with the second POI time stamp.
- `Tmp_Is_Overlapped_By` reverse of `Tmp_Overlaps`
- `Tmp_Starts` The start first POI timestamp is tha same as the start of the second POI time stamp.
- `Tmp_Is_Started_By` reverse of `Tmp_Starts`
- `Tmp_Meets` The end first POI timestamp meets the start of the second POI time stamp.
- `Tmp_Is_xBy` reverse of `Tmp_Meets`

Below, an example of an atomic LS that consists of the temporal measure `Tmp_Finishes` and a threshold `theta = 1.0` is given:

```
Tmp_Finishes(x.beginDate1|endDate1, y.beginDate2|endDate2) | 0.8
```

where `beginDate1` and `beginDate2` are properties of the source and target KB respectively, whose values indicate the begin of a temporal event instance and `endDate1` and `endDate2` are properties of the source and target KB respectively, whose values indicate the end of a temporal event instance. Both begin and end properties for both source and target MUST be included in an atomic LS whose measure is temporal. Also, the acceptable values for all properties are in the format: `2015-04-22T11:29:51+02:00`.

3.2.5. Execution (optional)

Additional fine-tuning parameters can be set using the `Execution` tag. We recommend LIMES users to use the default parameters for each of the execution parameters. An advanced user needs to consult the developer's manual[15] for correct setting of the optional parameters. Three LIMES execution parameters can be set here:

- `REWRITER`: LIMES 1.0.0 implements the `DEFAULT` rewriter. **(recommended)**

[15] https://dice-group.github.io/LIMES/developer_manual/

- `PLANNER:` the user can choose between:
  - `CANONICAL:` It generates an immutable plan in a static manner.
  - `HELIOS:` It generates an immutable plan in a dynamic manner.
  - `DYNAMIC:` It generates a mutable plan in a dynamic manner.
  - `DEFAULT:` same as `CANONICAL`. **(recommended)**
- `ENGINE:` the user can choose between:
  - `SIMPLE:` It executes each independent part of the plan sequentially.
  - `DEFAULT:` same as `SIMPLE`. **(recommended)**
  - if not set, the `DEFAULT` value for each parameter will used be will

Machine Learning

In most cases, finding a good metric expression (i.e. one that achieves high F-Measure in interlinking POI entities) is not a trivial task. Therefore, in LIMES we implemented a number of machine learning algorithms for auto-generation of mappings among POI resources. For using a machine learning algorithm in your configuration file use the `MLALGORITHM` tag instead of the `METRIC` tag. For example:

```
<MLALGORITHM>
    <NAME>wombat simple</NAME>
    <TYPE>supervised batch</TYPE>
    <TRAINING>trainingData.nt</TRAINING>
    <PARAMETER>
        <NAME>max execution time in minutes</NAME>
        <VALUE>60</VALUE>
    </PARAMETER>
</MLALGORITHM>
```

In the above example, the following apply:

- The tag `NAME` contains the name of the machine learning algorithm. Currently, we have implemented the following algorithms:
  - `wombat simple`: First the `wombat simple` algorithm generates a set of initial atomic LSs. Then, it uses the three logical connectors ⊔, ⊓ and \ to append further atomic LS. `wombat simple` uses a naïve operator for LS generation based of concept of the generalisation via an upward refinement operator to traverse the space of link specification. The naïve refinement operator used by `wombat simple` is not complete (i.e., not able to reach all possible LSs). Our experiments show that `wombat simple` is faster and as accurate as the `wombat complete`.
  - `wombat complete`: Same as wombat simple but uses a more sophisticated complete refinement operator to be able to reach all the possible LSs. Our experiments show that `wombat complete` is slower and as accurate as the `wombat simple`.
  - `Eagle`: EAGLE is a genetic-algorithm-based approach for link specification learning. `EAGLE` is non-deterministic (i.e. running EAGLE for the same setting many times generates different results).

- The tag `TYPE` contains the type of the machine learning algorithm, which could take one of the values:
    - `supervised batch`: The user provides a file contains number of labelled examples for the different ML algorithms to learn from.
    - `supervised active`: The user iteratively will be asked by the ML algorithm for labelling feedback (i.e., user will be asked to mark one of more example link(s) as positive or negative at each iteration run of the ML algorithm).
    - `Unsupervised`: No training data needed from the user. Note that unsupervised linking is currently supported only for `owl:sameAs` links.
- The tag `TRAINING` contains the full path to the training data file. Note that this tag is not required in case of the supervised active and unsupervised learning algorithms. Usually a `*.ttl` file is used where the URIs of the entities are linked via `owl:sameAs` e.g.: `<http://sourceexample.org/entity1> <http://www.w3.org/2002/07/owl#sameAs><http://targetexample.org/entity1>`
- The tag `PARAMETER` contains the name (using the sub-tag `NAME`) and the value (using the sub-tag `VALUE`) of the used machine learning algorithm parameter. User can use as many `PARAMETER` tags as it is required. Note that LIMES uses the default values of all unspecified parameters.

Table 2 contains a list of implemented algorithms together with supported implementations and parameters.

Table 2: ML algorithms in LIMES

ML Algorithm	Supported types	Parameter	Default Value	Note
WOMBAT Simple	`supervised batch, supervised active and unsupervised`	`max refinement tree size`	2000	
		`max iterations number`	3	
		`max iteration time in minutes`	20	
		`max execution time in minutes`	600	
		`max fitness threshold`	1	Range 0 to 1
		`minimum property coverage`	0.4	Range 0 to 1
		`property learning rate`	0.9	Range 0 to 1
		`overall penalty weight`	0.5	Range 0 to 1

		children penalty weight	1	Range 0 to 1
		complexity penalty weight	1	Range 0 to 1
		verbose	false	
		atomic measures	jaccard, trigrams, cosine, qgrams	
		save mapping	true	
WOMBAT Complete	supervised batch, supervised active and unsupervised	Same as WOMBAT Simple		
EAGLE	supervised batch, supervised active and unsupervised	generations	10	Integer
		preserve_fittest	true	
		max_duration	60	[1,Inf)
		inquiry_size	10	[1,Inf)
		max_iterations	500	[1,Inf)
		max_quality	0.5	[0.0,1.0]
		termination_criteria	iteration	enum
		termination_criteria_value	0.0	[0.0,Inf)
		beta	1.0	[0.0,1.0]
		population	20	[1,Inf)
		mutation_rate	0.4	[0.0,1.0]
		reproduction_rate	0.4	[0.0,1.0]
		crossover_rate	0.3	[0.0,1.0]

3.2.6. Granularity (optional)

The user can choose positive integers to set the granularity of hyperspace tiling approaches used by LIMES by setting

```
<GRANULARITY>2</GRANULARITY>.
```

Our evaluations show that the default granularity of 2 achieves the best performance for most of the datasets. Therefore we recommend LIMES' users not to change the default value. An advanced user needs to consult the developer's manual[16] for correct setting of the granularity parameters.

3.2.7. Acceptance Condition

Filling the acceptance condition consists of setting the threshold value to the minimum value that two instances must have in order to satisfy a relation. This can be carried out as exemplified below.

```
<ACCEPTANCE>
    <THRESHOLD>0.98</THRESHOLD>
    <FILE>accepted.nt</FILE>
    <RELATION>owl:sameAs</RELATION>
</ACCEPTANCE>
```

By using the `THRESHOLD` tag, the user can set the minimum value that two instances must have in order to satisfy the relation specified in the `RELATION` tag, i.e., `owl:sameAs` in our example. Setting the tag `<FILE>` allows to specify where the links should be written. Currently, LIMES produces output files in the N3 format.Future versions of LIMES will allow to write the output to other streams and in other data formats.

3.2.8. Review Condition

Setting the condition upon which links must be reviewed manually is very similar to setting the acceptance condition as shown below.

```
<REVIEW>
    <THRESHOLD>0.95</THRESHOLD>
    <FILE>reviewme.nt</FILE>
    <RELATION>owl:sameAs</RELATION>
</REVIEW>
```

All instances that have a similarity between the threshold set in `REVIEW` (0.95 in our example) and the threshold set in `ACCEPTANCE` (0.98 in our example) will be written in the review file and linked via the relation set in `REVIEW`.

The LIMES configuration file should be concluded with `</LIMES>`

3.2.9. Output Format

The user can choose between `TAB` (for CSV records) and `N3` (for RDF triples) as output format by setting:

```
<OUTPUT>N3</OUTPUT>
```

3.3. Configuration File Examples

The following listing shows the whole configuration file for LIMES explicated in the sections above.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE LIMES SYSTEM "limes.dtd">
<LIMES>
    <PREFIX>
```

[16] https://dice-group.github.io/LIMES/developer_manual/

```
        <NAMESPACE>http://geovocab.org/geometry#</NAMESPACE>
        <LABEL>geom</LABEL>
    </PREFIX>
    <PREFIX>
        <NAMESPACE>http://www.opengis.net/ont/geosparql#</NAMESPACE>
        <LABEL>geos</LABEL>
    </PREFIX>
    <PREFIX>
        <NAMESPACE>http://linkedgeodata.org/ontology/</NAMESPACE>
        <LABEL>lgdo</LABEL>
    </PREFIX>
    <SOURCE>
        <ID>linkedgeodata</ID>
        <ENDPOINT>http://linkedgeodata.org/sparql</ENDPOINT>
        <VAR>?x</VAR>
        <PAGESIZE>2000</PAGESIZE>
        <RESTRICTION>?x a lgdo:RelayBox</RESTRICTION>
        <PROPERTY>geom:geometry/geos:asWKT RENAME polygon</PROPERTY>
    </SOURCE>
    <TARGET>
        <ID>linkedgeodata</ID>
        <ENDPOINT>http://linkedgeodata.org/sparql</ENDPOINT>
        <VAR>?y</VAR>
        <PAGESIZE>2000</PAGESIZE>
        <RESTRICTION>?y a lgdo:RelayBox</RESTRICTION>
        <PROPERTY>geom:geometry/geos:asWKT RENAME polygon</PROPERTY>
    </TARGET>
    <METRIC>geo_hausdorff(x.polygon, y.polygon)</METRIC>
    <ACCEPTANCE>
        <THRESHOLD>0.9</THRESHOLD>
        <FILE>lgd_relaybox_verynear.nt</FILE>
        <RELATION>lgdo:near</RELATION>
    </ACCEPTANCE>
    <REVIEW>
        <THRESHOLD>0.5</THRESHOLD>
        <FILE>lgd_relaybox_near.nt</FILE>
        <RELATION>lgdo:near</RELATION>
    </REVIEW>

    <EXECUTION>
        <REWRITER>default</REWRITER>
        <PLANNER>default</PLANNER>
        <ENGINE>default</ENGINE>
    </EXECUTION>

    <OUTPUT>TAB</OUTPUT>
</LIMES>
```

The following configuration file uses the machine learning algorithm of the Wombat simple to find the metric expression for the same example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE LIMES SYSTEM "limes.dtd">
<LIMES>
    <PREFIX>
        <NAMESPACE>http://geovocab.org/geometry#</NAMESPACE>
        <LABEL>geom</LABEL>
    </PREFIX>

    <PREFIX>
        <NAMESPACE>http://www.opengis.net/ont/geosparql#</NAMESPACE>
```

```
        <LABEL>geos</LABEL>
    </PREFIX>

    <PREFIX>
        <NAMESPACE>http://linkedgeodata.org/ontology/</NAMESPACE>
        <LABEL>lgdo</LABEL>
    </PREFIX>

    <SOURCE>
        <ID>linkedgeodata</ID>
        <ENDPOINT>http://linkedgeodata.org/sparql</ENDPOINT>
        <VAR>?x</VAR>
        <PAGESIZE>2000</PAGESIZE>
        <RESTRICTION>?x a lgdo:RelayBox</RESTRICTION>
        <PROPERTY>geom:geometry/geos:asWKT RENAME polygon</PROPERTY>
    </SOURCE>

    <TARGET>
        <ID>linkedgeodata</ID>
        <ENDPOINT>http://linkedgeodata.org/sparql</ENDPOINT>
        <VAR>?y</VAR>
        <PAGESIZE>2000</PAGESIZE>
        <RESTRICTION>?y a lgdo:RelayBox</RESTRICTION>
        <PROPERTY>geom:geometry/geos:asWKT RENAME polygon</PROPERTY>
    </TARGET>

    <MLALGORITHM>
        <NAME>wombat simple</NAME>
        <TYPE>supervised batch</TYPE>
        <TRAINING>trainingData.nt</TRAINING>
        <PARAMETER>
            <NAME>max execution time in minutes</NAME>
            <VALUE>60</VALUE>
        </PARAMETER>
    </MLALGORITHM>

    <ACCEPTANCE>
        <THRESHOLD>0.9</THRESHOLD>
        <FILE>lgd_relaybox_verynear.nt</FILE>
        <RELATION>lgdo:near</RELATION>
    </ACCEPTANCE>

    <REVIEW>
        <THRESHOLD>0.5</THRESHOLD>
        <FILE>lgd_relaybox_near.nt</FILE>
        <RELATION>lgdo:near</RELATION>
    </REVIEW>

    <EXECUTION>
        <REWRITER>default</REWRITER>
        <PLANNER>default</PLANNER>
        <ENGINE>default</ENGINE>
    </EXECUTION>

    <OUTPUT>TAB</OUTPUT>
</LIMES>
```

3.4. Running LIMES

Once the configuration file containing all the 10 elements detailed in the previous section is written, the last step consists in actually running the LIMES framework. Here, we detail how to run LIMES with an arbitrary configuration file (dubbed here as `config.xml`).

3.4.1. Running LIMES core

For running LIMES from command line, the following command needs to be executed:

```
java -jar LIMES.jar config.xml [OPTIONS...].
```

The following optional command line flags and options are available:

- `-f $format` sets the format of configuration file. Possible values for `$format` are `"XML"` (default) or `"RDF"`
- `-s` runs the LIMES server
- `-p $port` used to specify port of LIMES server; defaults to port 8080
- `-l $limit` limits the number of resources processed by LIMES server to `$limit`; defaults to -1 (no limit). **CAUTION:** Setting this option will compromise the correctness of LIMES and is only encouraged to reduce server load for demo purposes.
- `-h` prints out a help message
- `-o $file_path` sets the path of the logging file

In case your system runs out of memory, please use the `-Xmx` option (must appear before the -jar option) to allocate more memory to the Java Virtual Machine.

3.4.2. Running LIMES GUI

For running LIMES GUI, one should switch to the `limes-gui/` folder and run:

```
mvn jfx:jar -Dcheckstyle.skip=true -Dmaven.test.skip=true
```

The jar will be placed in `limes-gui/limes-gui/target/jfx/app/`

Note that, the `limes-gui/target/jfx/app/lib` folder needs to be in the same folder as the `.jar` for the `.jar` to work.

3.4.3. Running LIMES from Java

For running LIMES from java, please consult the developer manual at:

```
https://dice-group.github.io/LIMES/developer_manual
```

3.5. LIMES GUI

The main purpose of the LIMES GUI is to provide the users of LIMES with an easy interface for configuring LIMES, without the need to write the configuration file in the XML or RDF serialization. The LIMES GUI (see

Figure 4) consists of two main components the *menu bar* and the *tool box*. We will discuss each of the two components in the following subsections.

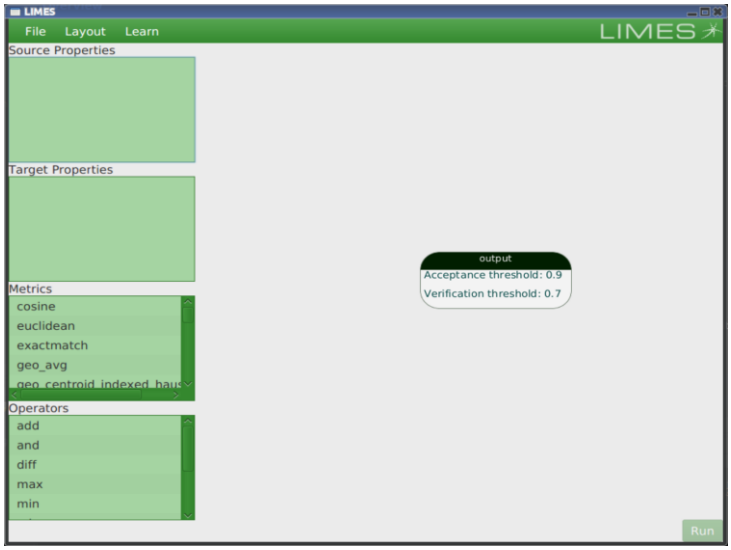


Figure 4: LIMES GUI

3.5.1. Menu Bar

The menu bar (as shown in top of Figure 4) contains three drop-down menus:

- **File:** The file drop-down menu gives the possibility to:
  - New: Create a new configuration
  - Load Config: Load a configuration file.
  - Save Config: Save a configuration to a file (only possible, after loading a configuration or creating a new configuration)
  - Exit
- **Layout:** Handles the layout of the current metric
  - Refresh Layout: Rearranges the nodes of the metric in a tree-like structure
  - Delete Graph: Delete the current metric leaving only an output node
- **Learn:** All the machine learning functionality of the GUI can be accessed through this drop-down menu (These features are only available when a configuration is loaded):
  - *Active Learning*
  - *Batch Learning*
  - *Unsupervised Learning*

3.5.2. Toolbox

On the panel of the GUI, one can find the toolbox (see Figure 5) containing all options of LIMES. The user needs to build his own metric after (s)he loaded/made a configuration, the components of the toolbox include:

- Source/Target Properties: The properties you want to link (if you did not load or create a configuration these are empty)
- Measures: All the measures you can use to link properties
- Operators: All the operators you can use to combine measures

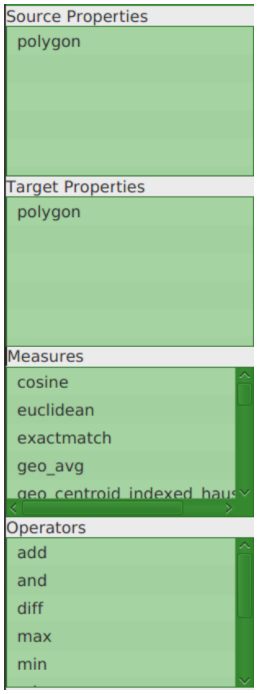


Figure 5: Toolbox

3.5.3. Metric Builder

The metric builder (see Figure 6) eases the process of complex link specification creation, especially for end users with limited programming experience. In particular, the user can visually link the various atomic link specification nodes to create the complex ones (s)he needs.

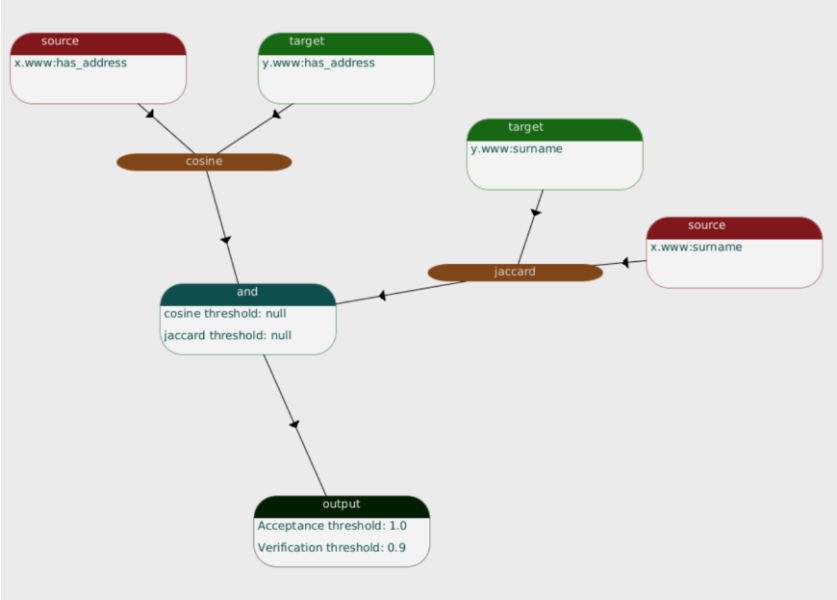


Figure 6: Metric builder

3.5.4. Creating New Configuration via GUI

The GUI user can configure LIMES to **read data from an external SPARQL endpoint** by clicking on File -> New. A window will pop up (as the one shown in Figure 7), in which the source and target endpoints of the new configuration can be configured.

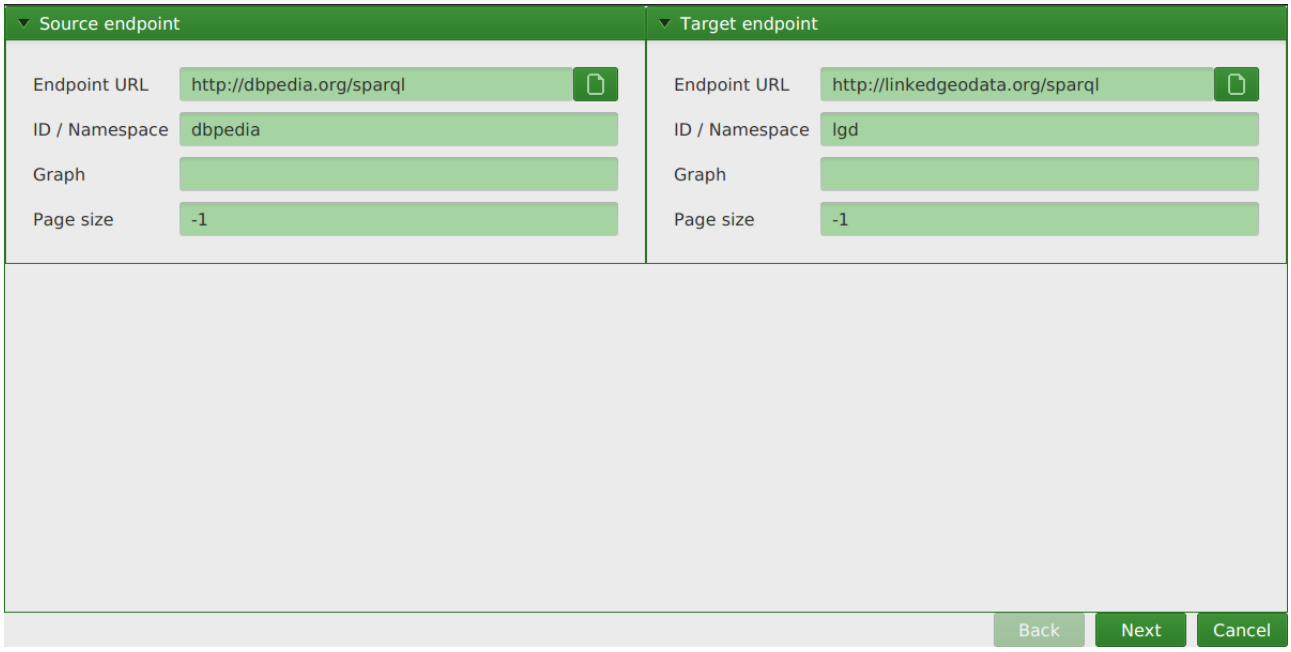


Figure 7: SPARQL configuration window

The user can use the following fields for configuring his/her LIMES run:

- *EndpointURL:* Either a URL of a SPARQL Endpoint is entered here or the *filepath* to a local endpoint. Files can also be entered more easily by pressing the little green button with the file symbol which opens a file chooser dialog.

- *ID/Namespace*: Source/Target Endpoint can be given a name (optional).
- *Graph*: Specify the graph. If this is left empty, the default graph will be used.
- *Page size*: How many pages of the endpoint should be fetched? (-1 = all)

Let's use `http://dbpedia.org/sparql` as source endpoint and `http://linkedgeodata.org/sparql` as target endpoint URL. We enter `DBpedia` as *source ID* and `LGD` as target ID.

Pressing the *Next* button gets you to the next step of **class matching** presented in Figure 8. A source and target class must be selected by clicking on it to continue. Some classes have subclasses which can be accessed by clicking on the arrow besides them. We click on `HistoricPlace` from DBpedia and `HistoricThing` from *LGD*.

The Next step is **property matching** is shown in Figure 9. Clicking on the available properties moves them to the bottom container, where the already added properties can be seen. If you change your mind, clicking on added properties moves them back up. Alternatively, all available properties can be added with the button Add All. At least one source and one target property have to be added.

Let's take `rdfs:label` for both. Since the properties are alphabetically sorted you can find those towards the bottom of each list.

Press Finish and you are now ready to build a metric!

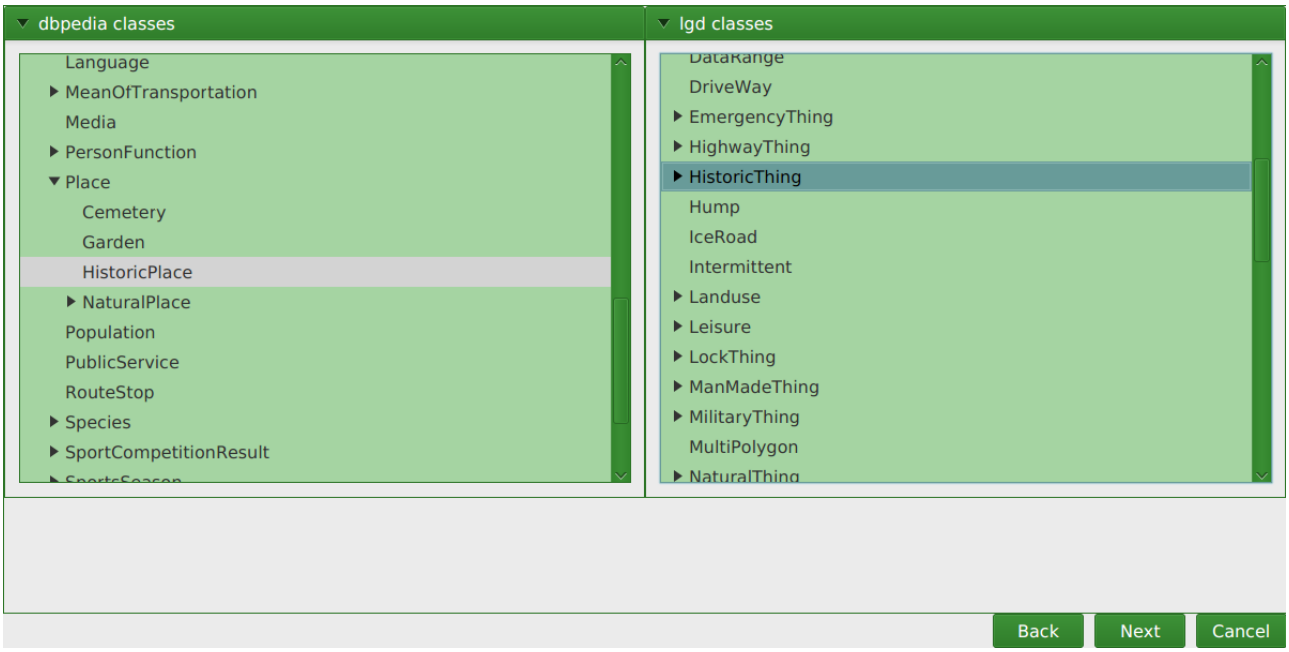


Figure 8: Class matcher

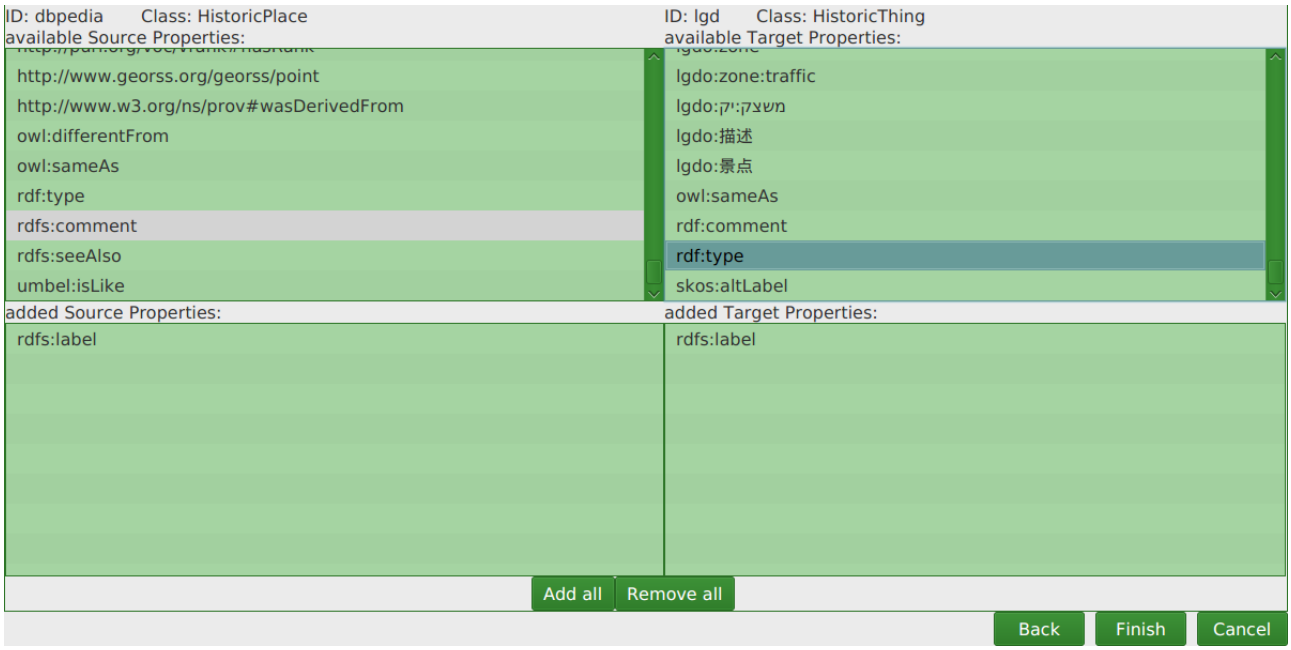


Figure 9: Property matcher

3.5.5. Creating a new Link Specification via GUI

By clicking on elements from the toolbox, the user can make the nodes he/she needs to appear in the *metric builder* (Like the ones in Figure 6). Let us assume that the user clicked on both `rdfs:label` properties he/she has. Now we need a measure to check the similarity between those properties. Let's choose `cosine` for example.

Right clicking the nodes will create a small context menu for the current node. If the user clicks *Link To*, the user can link the node with an appropriate other node. The following links are permitted:

- property -> measure
- measure -> output
- measure -> operator
- operator -> output

Also, operator and measure need two nodes that link to them. The context menu also gives the user the possibility to *Close* it or *Delete* the node. If the user wants to delete a link, he/she just needs to right-click the arrow. Let's link our properties with `cosine` and the measure with `output`. If the user wants, he/she can define a Acceptance Threshold and Verification Threshold.

3.5.6. Link Specification Running

If the user followed the steps, his/her *link specification* should look something like the one presented in Figure 10. For running the current *link specification*, the user just click on the button in the bottom right corner.

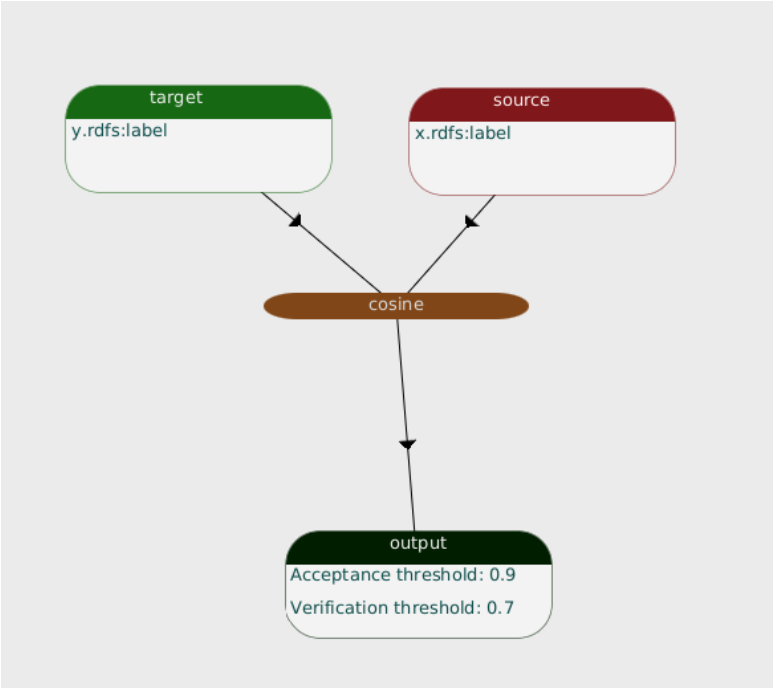
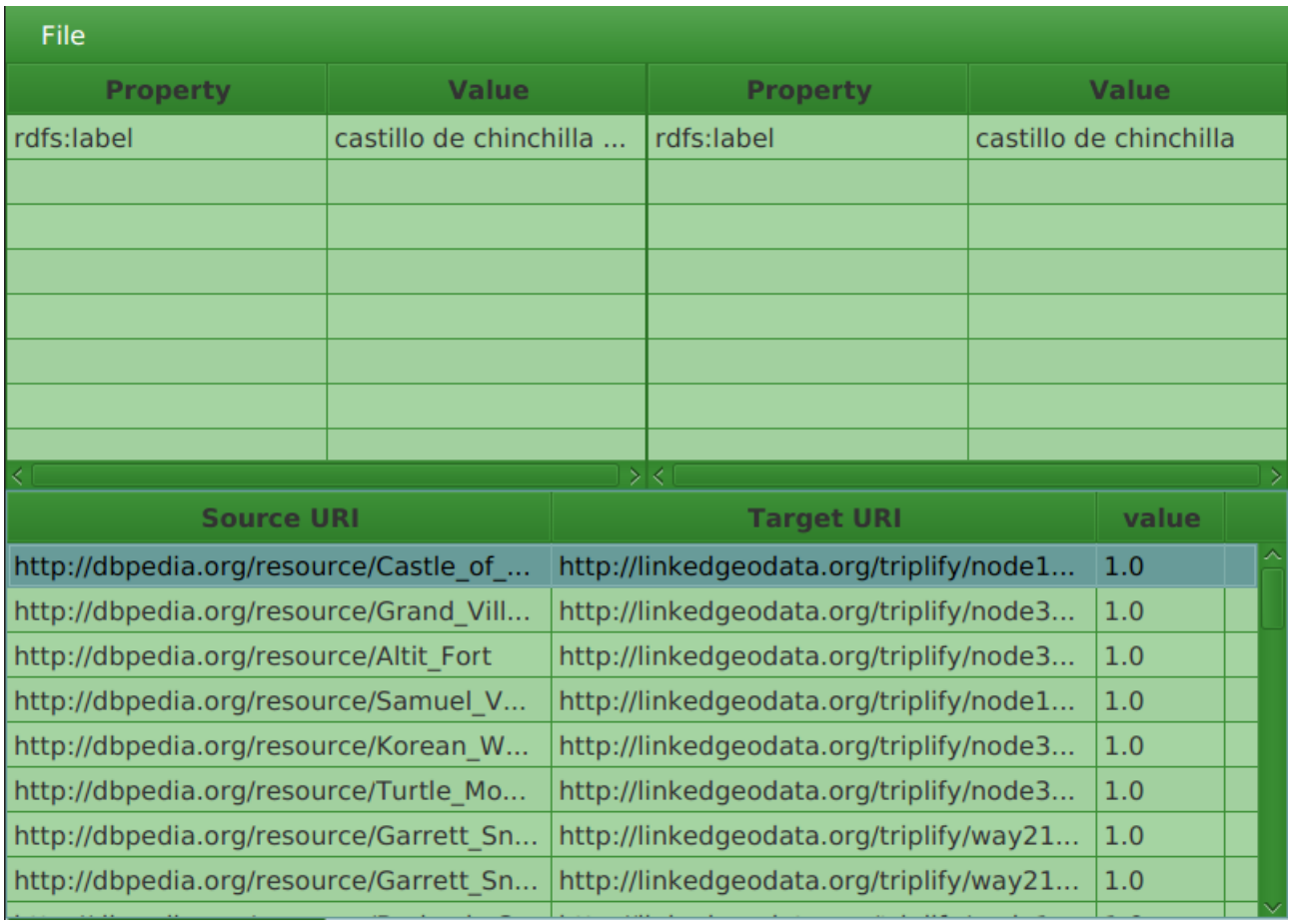


Figure 10: LS example in the metric builder

After the progress popup vanished the user should see his/her results in a new window such as the one presented in Figure 11. In the top left the user has the possibility to save the resulted set of links into a file.



File

Property	Value	Property	Value
rdfs:label	castillo de chinchilla ...	rdfs:label	castillo de chinchilla

Source URI	Target URI	value
http://dbpedia.org/resource/Castle_of_...	http://linkedgeodata.org/triplify/node1...	1.0
http://dbpedia.org/resource/Grand_Vill...	http://linkedgeodata.org/triplify/node3...	1.0
http://dbpedia.org/resource/Altit_Fort	http://linkedgeodata.org/triplify/node3...	1.0
http://dbpedia.org/resource/Samuel_V...	http://linkedgeodata.org/triplify/node1...	1.0
http://dbpedia.org/resource/Korean_W...	http://linkedgeodata.org/triplify/node3...	1.0
http://dbpedia.org/resource/Turtle_Mo...	http://linkedgeodata.org/triplify/node3...	1.0
http://dbpedia.org/resource/Garrett_Sn...	http://linkedgeodata.org/triplify/way21...	1.0
http://dbpedia.org/resource/Garrett_Sn...	http://linkedgeodata.org/triplify/way21...	1.0

Figure 11: Result window

3.5.7. Machine Learning via GUI

Since finding a good metric expression can be hard, we also have implemented machine learning algorithms in the GUI. There are three different types of algorithms you can use:

- Active Learning
- Batch Learning
- Unsupervised Learning

To use any of these algorithms, the user has to either create a new LIMES configuration or load one from file. In the menu bar, the user should click on *Learn* and choose the type he/she wants to use. A new window will pop up. In the top left corner. The user will find a drop-down menu, showing him/her which algorithms implement the chosen learning type. After he/she clicks on his/her desired algorithm, the window will be filled with the elements he/she can use to set the configuration parameters. Figure 12 shows an example of setting the parameters of the WOMBAT simple ML algorithm from the GUI.

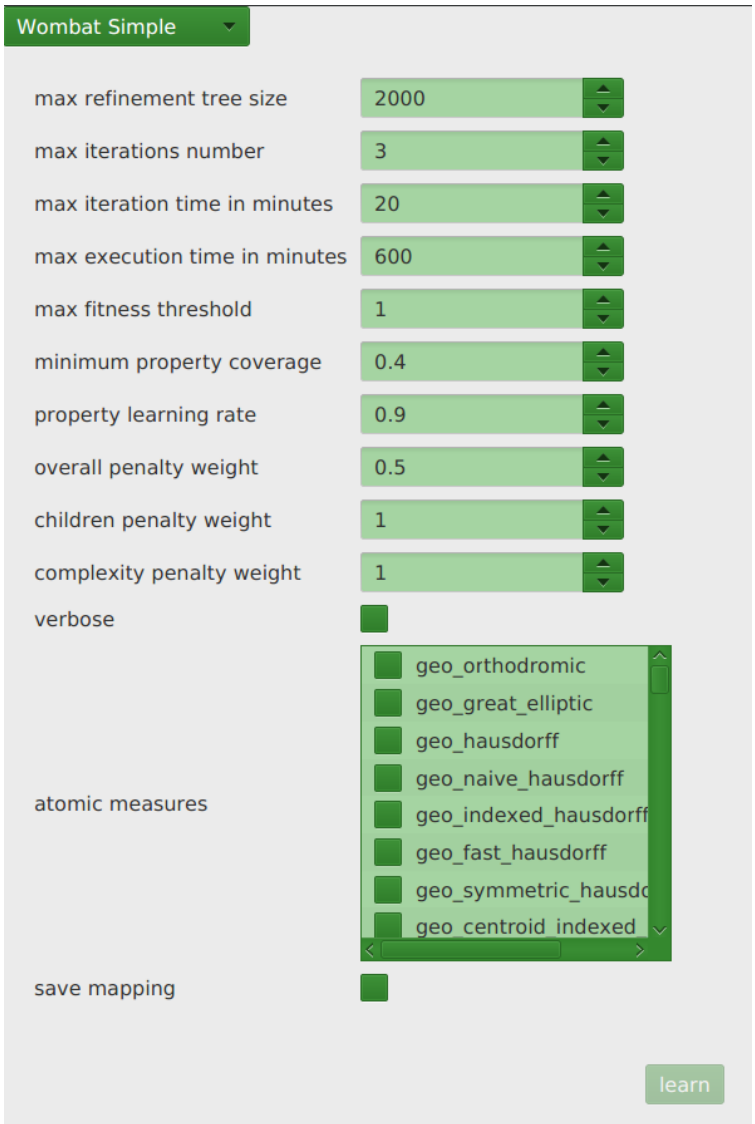


Figure 12: WOMBAT simple parameters in the GUI

3.5.7.1. Active Learning via GUI

In case the user is happy with the parameters he/she set, the user must click on *Learn* in the bottom right corner. After the progress popup vanishes he/she will see a new window, where the algorithm wants you to label link candidates as matches or non-matches as shown in Figure 13.

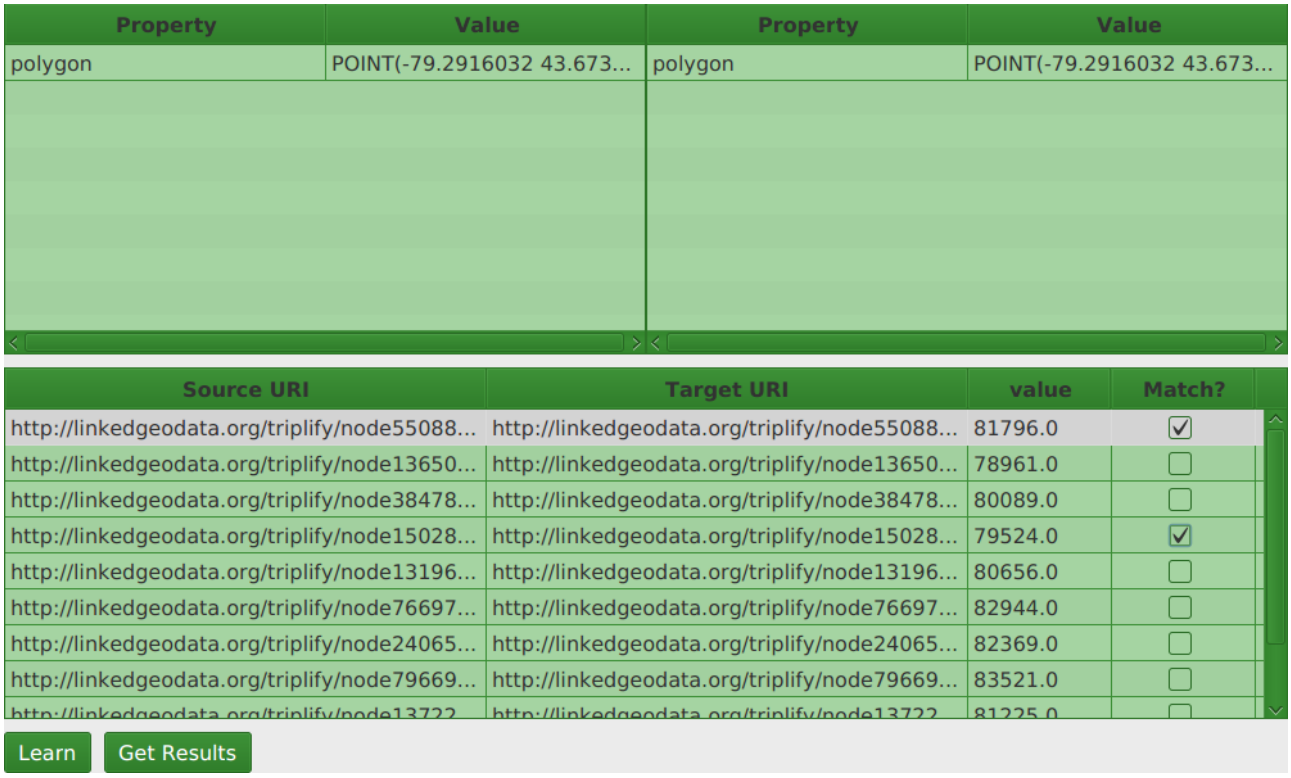


Figure 13: active learning via GUI

The user can click on *Learn* again and another iteration starts. If she doesn't want another iteration, the user can click on *Get Results* and a new view with results will pop up. This time he/she also has the possibility to *Save the resulted link specification* in the bottom left corner.

3.5.8. Batch Learning via GUI

This learning type only takes one iteration and the user has to provide a file containing the training mapping. The file can be either CSV or RDF in any serialization. For CSV, the first line contains the properties on which the user wants to match, and the following lines the respective properties' values of the instance. For example:

```
id1,id2
```

```
http://www.okkam.org/oaie/person1-Person2190,http://www.okkam.org/oaie/person2-Person2191
```

For example, if the user uses an RDF file in n-triple serialization, a mapping for `owl:sameAs` predicate will look like:

```
<http://linkedgeodata.org/triplify/node3635863841>
<http://www.w3.org/2002/07/owl#sameAs>
<http://linkedgeodata.org/triplify/node3635863841>
```

Of course, the more training data the user provides the better the algorithm can learn. After the user click on *Save* the learning will start.

3.5.9. Unsupervised Learning via GUI

This is the type of machine learning algorithms that needs the least effort from the user. The user has just to click on *Learn* and after the algorithm is finished, the results will be presented. Note that, LIMES only support the unsupervised machine learning for `owl:sameAs` relations.

4. Evaluation

In this Section, we present the experiments that assess the efficiency and scalability of LIMES. In our experiments, we evaluated the running time of LIMES, as it is the primary concern in the context of the SLIPO project. First, we briefly present the POI datasets we used in our evaluation. Next, we present two interlinking configurations which we implemented exemplarily for the use case DINUC A.1, as specified in Deliverable D5.1 (Pilot Specifications). The experiments assess the performance of the software for these interlinking configurations involving more than 310K and 350K POIs, respectively, and show that LIMES is able to discover diverse relationships between in very large POI datasets in an acceptable time.

4.1. Datasets and measures

4.1.1. Datasets

In our experiments, we use the TomTom and Herold company POI datasets for Austria. The key properties of these datasets are depictured in Table 3.

Table 3: Datasets Used in the Run Time Evaluation

	Source Dataset	Target Dataset
Dataset Name	TomTom_mnpoi_Austria.nt	public.herold.nt
Provenance (Owner)	TomTom	Herold
Size in MB	1986.56	3614.72
Number of POIs	311,078	350,053
Number of triples	13,587,632	26,214,691
Properties used in linking	3	3

4.1.2. Measures

In the performed experiments, we measure run times (*in seconds*) for several parts of the interlinking process. Specifically, we report run times for (a) the loading of the data from disk, (b) the interlinking phase, and (c) the saving of the result files. We also report the overall run time of the LIMES software. Moreover, we measure the sizes of the result files, i.e., the number of relationships discovered between POIs by LIMES.

4.2. Interlinking Configurations

We developed two different interlinking configurations to showcase LIMES ability to discover arbitrary relationships between POIs.

The first configuration (see 6.1.1) will generate so called `owl:sameAs` links, i.e., it aims to discover POIs that describe the same real-world object in both datasets. Therefore, it searches for POIs that share the same

address but uses string similarity measures to allow for small differences in the address representation, e.g., "*Akademiestraße*" and "*Akademiestr.*" should be treated as the same.

The second configuration (see 6.1.2) will generate `dbp:nearby` links, i.e. it aims to discover POIs that are within a given distance to each other. Therefore, it applies the HR3 algorithm in order to find POIs that are within 50m or 100m of each other.

4.3. Results

The logs of LIMES feature precise time stamps of its execution phases. Therefore, we compiled the experimental results from the LIMES logs after running the two interlinking configurations presented in the last section for the two input datasets presented in Section 4.1.1. We present the results of our evaluation in Table 4.

Table 4: Results for the Experimental Evaluation

Type	Measure	Value
`owl:sameAs`	Loading time	645s
	Linking time	674s
	Saving time	4s
	Result size	226,275 + 0
	Total time	1326s
`slipo:nearby`	Loading time	632s
	Linking time	33s
	Saving time	10s
	Result size	1,254,462 (50m) + 2,325,111 (100m)
	Total time	675s

As we can see in the results, LIMES is indeed able to find links in an acceptable time frame. Moreover, most of the time in the second experiment (`slipo:nearby`) is actually spent in the loading phase, with the discovery of more than 3M `slipo:nearby` relationships only requiring 32s.

5. References

[1] M. Nentwig, M. Hartung, A. N. Ngomo, E. Rahm, A survey of current link discovery frameworks, Semantic Web 8 (3) (2017) 419–436.
doi:10.3233/SW-150210.
URL http://dx.doi.org/10.3233/SW-150210

[2] C. Stadler, J. Lehmann, K. Höffner, S. Auer, Linkedgeodata: A core for a web of spatial open data, Semantic Web 3 (4) (2012) 333–354.
doi:10.3233/SW-2011-0052.
URL http://dx.doi.org/10.3233/SW-2011-0052

[3] M. Saleem, S. S. Padmanabhuni, A. N. Ngomo, J. S. Almeida, S. Decker, H. F. Deus, Linked cancer genome atlas database, in: M. Sabou, E. Blomqvist, T. D. Noia, H. Sack, T. Pellegrini (Eds.), I-SEMANTICS 2013 -9th International Conference on Semantic Systems, ISEM '13, Graz, Austria, September 4-6, 2013, ACM, 2013, pp. 129–134.
doi:10.1145/2506182.2506200.
URL http://doi.acm.org/10.1145/2506182.2506200

[4] A. N. Ngomo, S. Auer, LIMES - A time-efficient approach for large scale link discovery on the web of data, in: T. Walsh (Ed.), IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011, IJCAI/AAAI, 2011,pp. 2312–2317.
doi:10.5591/978-1-57735-516-8/IJCAI11-385.
URL http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-385

[5] A. N. Ngomo, Link discovery with guaranteed reduction ratio in affine spaces with Minkowski measures, in: The Semantic Web - ISWC2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I, 2012, pp. 378–393.
doi:10.1007/978-3-642-35176-1_24.
URL http://dx.doi.org/10.1007/978-3-642-35176-1_24

[6] A. N. Ngomo, ORCHID - reduction-ratio-optimal computation of geospatial distances for link discovery, in: Alani et al.
doi:10.1007/978- 3- 642- 41335- 3_25
URL http://dx.doi.org/10.1007/978-3-642-41335-3_ 25

[7] T. Soru, A. N. Ngomo, Rapid execution of weighted edit distances, in:Shvaiko et al. [52], pp. 25–36.
URL http://ceur-ws.org/Vol-1111/om2013_Tpaper3.pdf

[8] A. N. Ngomo, J. Lehmann, S. Auer, K. Höffner, RAVEN - active learning of link specifications, in: Shvaiko et al. [53], pp. 25–36.
URL http://ceur-ws.org/Vol-814/om2011_Tpaper3.pdf

[9] A. N. Ngomo, K. Lyko, EAGLE: efficient active learning of link specifications using genetic programming, in: The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings, 2012,

[10] A. N. Ngomo, K. Lyko, Unsupervised learning of link specifications: deterministic vs. non-deterministic, in: Shvaiko et al. [52], pp. 1–12.
URL http://ceur-ws.org/Vol-1111/om2013_Tpaper1.pdf

[11] M. Sherif, A.-C. Ngonga Ngomo, J. Lehmann, WOMBAT - A Generalization Approach for Automatic Link Discovery, in: 14th Extended Semantic Web Conference, Portorož, Slovenia, 28th May - 1st June 2017, Springer,2017.
URL http://svn.aksw.org/papers/2017/ESWC_WOMBAT/public.pdf

[12] A. N. Ngomo, On link discovery using a hybrid approach, J. Data Semantics 1 (4) (2012) 203–217.
doi:10.1007/s13740-012-0012-y.
URL http://dx.doi.org/10.1007/s13740-012-0012-y

[13] R. Isele, A. Jentzsch, C. Bizer, Efficient multidimensional blocking for link discovery without losing recall, in: Proceedings of the 14thInternational Workshop on the Web and Databases 2011, WebDB 2011, Athens, Greece, June 12, 2011, 2011.
URL http://webdb2011.rutgers.edu/papers/Paper%2039/silk.pdf

[14] A. Nikolov, V. S. Uren, E. Motta, Knofuss: a comprehensive architecture for knowledge fusion, in: Proceedings of the 4th International Conference on Knowledge Capture (K-CAP 2007), October 28-31, 2007, Whistler, BC, Canada, 2007, pp. 185–186.
doi:10.1145/1298406.1298446.
URL http://doi.acm.org/10.1145/1298406.1298446

[15] A. N. Ngomo, K. Lyko, V. Christen, COALA - correlation-aware active learning of link specifications, in: The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings, 2013, pp. 442–456.
doi:10.1007/978-3-642-38288-8_30.
URL http://dx.doi.org/10.1007/978-3-642-38288-8_30

[16] A.-C. Ngonga Ngomo, D. Obraczka, K. Georgala, Using machine learning for link discovery on the web of data, in: Demos at the European Conference on Artificial Intelligence, 2016.

[17] A. N. Ngomo, A time-efficient hybrid approach to link discovery, in: Shvaiko et al. [53].
URL http://ceur-ws.org/Vol-814/om2011_Tpaper1.pdf

[18] M. A. Sherif, A.-C. N. Ngomo, A Systematic Survey of Point Set Distance Measures for Link Discovery, Semantic Web Journal.
URL http://www.semantic-web-journal.net/content/systematic-survey-point-set-distance-measureslink-discovery-1

[19] K. Georgala, M. A. Sherif, A. N. Ngomo, An efficient approach for the generation of allen relations, in: ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016), 2016, pp. 948–956. doi:10.3233/978-1-61499-672-9-948.
URL http://dx.doi.org/10.3233/978-1-61499-672-9-948

[20] J. F. Allen, Maintaining knowledge about temporal intervals, Commun. ACM 26 (11) (1983) 832–843. doi:10.1145/182.358434.URL http://doi.acm.org/10.1145/182.358434

[21] M. A. Sherif, K. Dreßler, P. Smeros, A. N. Ngomo, Radon - rapid discovery of topological relations, in: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA., 2017, pp. 175–181.
URL http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14199

[22] T. Soru, E. Marx, A. N. Ngomo, ROCKER: A refinement operator for key discovery, in: Proceedings of the 24th International Conference onWorld Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015, 2015, pp. 1025–1033. doi:10.1145/2736277.2741642.
URL http://doi.acm.org/10.1145/2736277.2741642

[23] C. Xiao, W. Wang, X. Lin, Ed-join: an efficient algorithm for similarity joins with edit distance constraints, Proceedings of the VLDB Endowment1 (1) (2008) 933–944.

[24] G. Li, D. Deng, J. Wang, J. Feng, Pass-join: A partition-based method for similarity joins, Proceedings of the VLDB Endowment 5 (3) (2011)253–264.

[25] K. Dreßler, A. N. Ngomo, On the efficient execution of bounded jaro winkler distances, Semantic Web 8 (2) (2017) 185–196.
doi:10.3233/SW-150209.
URL http://dx.doi.org/10.3233/SW-150209

[26] E. Fredkin, Trie memory, Commun. ACM 3 (9) (1960) 490–499.
doi:10.1145/367390.367400.
URL http://doi.acm.org/10.1145/367390.367400

[27] A. N. Ngomo, HELIOS - execution optimization for link discovery, in:Mika et al. [54], pp. 17–32. doi:10.1007/978-3-319-11964-9_2.
URL http://dx.doi.org/10.1007/978-3-319-11964-9_2

[28] A. N. Ngomo, M. A. Sherif, K. Lyko, Unsupervised link discovery through knowledge base repair, in: The Semantic Web: Trends and Challenges -11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May25-29, 2014. Proceedings, 2014, pp. 380–394. doi:10.1007/978-3-319-07443-6_26.
URL http://dx.doi.org/10.1007/978-3-319-07443-6_26

[29] T. Soru, A. N. Ngomo, Active learning of domain-specific distances for link discovery, in: Semantic Technology, Second Joint International Conference, JIST 2012, Nara, Japan, December 2-4, 2012. Proceedings, 2012, pp. 97–112.
doi:10.1007/978-3-642-37996-3_7.
URL http://dx.doi.org/10.1007/978-3-642-37996-3_7

[30] A. N. Ngomo, S. Auer, J. Lehmann, A. Zaveri, Introduction to linked data and its lifecycle on the web, in: Reasoning Web. Reasoning onthe Web in the Big Data Era - 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings, 2014, pp. 1–99.doi:10.1007/978-3-319-10587-1_1.
URL http://dx.doi.org/10.1007/978-3-319-10587-1_1

[31] M. A. Sherif, A. N. Ngomo, Semantic quran, Semantic Web 6 (4) (2015)339–345. doi:10.3233/SW-140137.
URL http://dx.doi.org/10.3233/SW-140137

[32] J. Lehmann, T. Furche, G. Grasso, A. N. Ngomo, C. Schallhart, A. J.Sellers, C. Unger, L. Bühmann, D. Gerber, K. Höffner, D. Liu, S. Auer,deqa: Deep web extraction for question answering, in: The Semantic Web- ISWC 2012 - 11th International Semantic Web Conference, Boston, MA,USA, November 11-15, 2012, Proceedings, Part II, 2012, pp. 131–147. doi:10.1007/978-3-642-35173-0_9.
URL http://dx.doi.org/10.1007/978-3-642-35173-0_9

[33] R. Speck, A. N. Ngomo, Ensemble learning for named entity recognition, in: Mika et al. [54], pp. 519–534. doi:10.1007/978-3-319-11964-9_33.
URL http://dx.doi.org/10.1007/978-3-319-11964-9_33

[34] R. Usbeck, A. N. Ngomo, M. Röder, D. Gerber, S. A. Coelho, S. Auer, A. Both, AGDISTIS - graph-based disambiguation of named entities using linked data, in: Mika et al. [54], pp. 457–471. doi:10.1007/978-3-319-11964-9_29.
URL http://dx.doi.org/10.1007/978-3-319-11964-9_29

[35] M. Morsey, J. Lehmann, S. Auer, A.-C. Ngonga Ngomo, DBpedia SPARQL Benchmark—Performance Assessment with Real Queries onReal Data, in: ISWC 2011, 2011.
URL http://jens-lehmann.org/files/2011/dbpsb.pdf

[36] M. Saleem, Q. Mehmood, A.-C. Ngonga Ngomo, Feasible: A feature based SPARQL benchmark generation framework, in: International Semantic Web Conference (ISWC), 2015.
URL http://svn.aksw.org/papers/2015/ISWC_FEASIBLE/public.pdf

[37] L. Bühmann, J. Lehmann, Pattern based knowledge base enrichment, in:Alani et al. [51], pp. 33–48. doi:10.1007/978-3-642-41335-3_3.
URL http://dx.doi.org/10.1007/978-3-642-41335-3_3

[38] M. A. Sherif, A. N. Ngomo, J. Lehmann, Automating RDF dataset transformation and enrichment, in: The Semantic Web. Latest Advances andNew Domains - 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31 - June 4, 2015. Proceedings, 2015, pp. 371–387. doi:10.1007/978-3-319-18818-8_23.
URL http://dx.doi.org/10.1007/978-3-319-18818-8_23

[39] J. Volz, C. Bizer, M. Gaedke, G. Kobilarov, Discovering and maintaining links on the web of data, in: The Semantic Web - ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings, 2009, pp. 650–665.

doi:10.1007/978-3-642-04930-9_41.
URL http://dx.doi.org/10.1007/978-3-642-04930-9_41

[40] X. Niu, S. Rong, Y. Zhang, H. Wang, Zhishi. links results for OAEI 2011, in: Shvaiko et al. [53].
URL http://ceur-ws.org/Vol-814/oaei11_paper16.pdf

[41] S. Araújo, J. Hidders, D. Schwabe, A. P. de Vries, SERIMI – resource description similarity, RDF instance matching and interlinking, in: Proceedings of the 6th International Workshop on Ontology Matching, Bonn, Germany, October 24, 2011, 2011.
URL http://ceur-ws.org/Vol-814/om2011_poster6.pdf

[42] K. Nguyen, R. Ichise, B. Le, SLINT: a schema-independent linked data interlinking system, in: Proceedings of the 7th International Workshop on Ontology Matching, Boston, MA, USA, November 11, 2012, 2012.
URL http://ceur-ws.org/Vol-946/om2012_Tpaper1.pdf

[43] J. Tang, B. Liang, J. Li, K. Wang, Risk minimization based ontology mapping, in: Content Computing, Advanced Workshop on Content Computing, AWCC 2004, ZhenJiang, JiangSu, China, November 15-17, 2004, Proceedings, 2004, pp. 469–480.
doi:10.1007/978-3-540-30483-8_58.
URL http://dx.doi.org/10.1007/978-3-540-30483-8_58

[44] I. F. Cruz, F. P. Antonelli, C. Stroe, Agreement maker: Efficient matching for large real-world schemas and ontologies, PVLDB 2 (2) (2009) 1586–1589.
URL http://www.vldb.org/pvldb/2/vldb09-1003.pdf

[45] E. Jiménez-Ruiz, B. C. Grau, LogMap: Logic-based and scalable ontology matching, in: The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I, 2011, pp. 273–288.
doi:10.1007/978-3-642-25073-6_18.
URL http://dx.doi.org/10.1007/978-3-642-25073-6_18

[46] M. Niepert, C. Meilicke, H. Stuckenschmidt, A probabilistic-logical framework for ontology matching, in: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia,USA, July 11-15, 2010, 2010.
URL http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1744

[47] L. Zhu, M. Ghasemi-Gol, P. A. Szekely, A. Galstyan, C. A. Knoblock, Unsupervised entity resolution on multi-type graphs, in: The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I, 2016, pp. 649–667.
doi:10.1007/978-3-319-46523-4_39.
URL http://dx.doi.org/10.1007/978-3-319-46523-4_39

[48] A.-C. N. Ngomo, M. M. Hassan, The lazy traveling salesman – memory management for large-scale link discovery., in: H. Sack, E. Blomqvist, M. d'Aquin, C. Ghidini, S. P. Ponzetto, C. Lange (Eds.), ESWC, Vol. 9678of Lecture Notes in Computer Science, Springer, 2016, pp. 423–438.
URL http://dblp.uni-trier.de/db/conf/esws/eswc2016.html#NgomoH16

[49] A.-C. Ngonga Ngomo, L. Kolb, N. Heino, M. Hartung, S. Auer, E. Rahm, When to reach for the cloud: Using parallel hardware for link discovery.,in: Proceedings of ESCW, 2013.

[50] M. A. Sherif, A. N. Ngomo, An optimization approach for load balancing in parallel link discovery, in: Proceedings of the 11th International Conference on Semantic Systems, SEMANTICS 2015, Vienna, Austria, September 15-17, 2015, 2015, pp. 161–168.
doi:10.1145/2814864.2814872.
URL http://doi.acm.org/10.1145/2814864.2814872

[51] H. Alani, L. Kagal, A. Fokoue, P. T. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. F. Noy, C. Welty, K. Janowicz (Eds.), The Semantic Web -ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I, Vol. 8218 of Lecture Notes in Computer Science, Springer, 2013.
doi:10.1007/978-3-642-41335-3.
URL http://dx.doi.org/10.1007/978-3-642-41335-3

[52] P. Shvaiko, J. Euzenat, K. Srinivas, M. Mao, E. Jiménez-Ruiz (Eds.),Proceedings of the 8th International Workshop on Ontology Matching co-located with the 12th International Semantic Web Conference (ISWC2013), Sydney, Australia, October 21, 2013, Vol. 1111 of CEUR Workshop Proceedings, CEUR-WS.org, 2013.
URL http://ceur-ws.org/Vol-1111

[53] P. Shvaiko, J. Euzenat, T. Heath, C. Quix, M. Mao, I. F. Cruz (Eds.), Proceedings of the 6th International Workshop on Ontology Matching, Bonn, Germany, October 24, 2011, Vol. 814 of CEUR Workshop Proceedings, CEUR-WS.org, 2011.
URL http://ceur-ws.org/Vol-814

[54] P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. A. Knoblock, D. Vrandecic, P. T. Groth, N. F. Noy, K. Janowicz, C. A. Goble (Eds.), The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I, Vol. 8796 of Lecture Notes in Computer Science, Springer, 2014.
doi:10.1007/978-3-319-11964-9.
URL http://dx.doi.org/10.1007/978-3-319-11964-9

6. Annex

6.1. Interlinking Configurations

6.1.1. Configuration 1: `owl:SameAs`

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE LIMES SYSTEM "limes.dtd">
<LIMES>
   <PREFIX>
       <NAMESPACE>http://slipo.eu/def#</NAMESPACE>
       <LABEL>slipo</LABEL>
   </PREFIX>

   <PREFIX>
       <NAMESPACE>http://www.opengis.net/ont/geosparql#</NAMESPACE>
       <LABEL>geos</LABEL>
   </PREFIX>

   <SOURCE>
       <ID>TomTom</ID>
       <ENDPOINT>TomTom_mnpoi_Austria.nt</ENDPOINT>
       <VAR>?x</VAR>
       <PAGESIZE>-1</PAGESIZE>
       <RESTRICTION>?x a geos:Feature</RESTRICTION>
       <PROPERTY>geos:hasGeometry/geos:asWKT RENAME polygon</PROPERTY>
       <PROPERTY>slipo:address/slipo:number AS nolang->lowercase
                 RENAME number</PROPERTY>
       <PROPERTY>slipo:address/slipo:postcode
                    AS nolang->lowercase RENAME postcode</PROPERTY>
       <PROPERTY>slipo:address/slipo:street
                    AS nolang->regularAlphabet->lowercase
                    RENAME street</PROPERTY>
       <PROPERTY>slipo:address/slipo:locality
                 AS nolang->lowercase RENAME locality</PROPERTY>
       <PROPERTY>slipo:phone/slipo:contactValue
                 AS nolang->removebraces->lowercase->replace(" ","")->number
                 RENAME phone</PROPERTY>
       <TYPE>NT</TYPE>
   </SOURCE>

   <TARGET>
       <ID>Herold</ID>
       <ENDPOINT>public.herold.nt</ENDPOINT>
       <VAR>?y</VAR>
       <PAGESIZE>-1</PAGESIZE>
       <RESTRICTION>?y a geos:Feature</RESTRICTION>
       <PROPERTY>geos:hasGeometry/geos:asWKT
                    RENAME polygon</PROPERTY>
       <PROPERTY>slipo:address/slipo:number
                 AS nolang->lowercase RENAME number</PROPERTY>
       <PROPERTY>slipo:address/slipo:postcode
                 AS nolang->lowercase RENAME postcode</PROPERTY>
```

```
        <PROPERTY>slipo:address/slipo:street
                  AS nolang->regularAlphabet->lowercase RENAME street</PROPERTY>
        <PROPERTY>slipo:address/slipo:locality AS nolang->lowercase
                  RENAME locality</PROPERTY>
        <PROPERTY>slipo:phone/slipo:contactValue
                  AS nolang->removebraces->lowercase->replace(" ","")->number
                  RENAME phone</PROPERTY>
        <TYPE>NT</TYPE>
    </TARGET>

    <METRIC>AND(AND(JaroWinkler(x.street,y.street)|0.93,
                   ExactMatch(x.number,y.number)|1.0)|1.0,
                   ExactMatch(x.postcode, y.postcode)|1.0)</METRIC>

    <ACCEPTANCE>
        <THRESHOLD>0.95</THRESHOLD>
        <FILE>tomtom-herold-links.nt</FILE>
        <RELATION>owl:sameAs</RELATION>
    </ACCEPTANCE>

    <REVIEW>
        <THRESHOLD>0.89</THRESHOLD>
        <FILE>tomtom-herold-links-verify.nt</FILE>
        <RELATION>owl:sameAs</RELATION>
    </REVIEW>

    <EXECUTION>
        <REWRITER>default</REWRITER>
        <PLANNER>default</PLANNER>
        <ENGINE>default</ENGINE>
    </EXECUTION>

    <OUTPUT>TAB</OUTPUT>

</LIMES>
```

6.1.2. Configuration 2: Nearby

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE LIMES SYSTEM "limes.dtd">
<LIMES>
    <PREFIX>
        <NAMESPACE>http://slipo.eu/def#</NAMESPACE>
        <LABEL>slipo</LABEL>
    </PREFIX>

    <PREFIX>
        <NAMESPACE>http://www.opengis.net/ont/geosparql#</NAMESPACE>
        <LABEL>geos</LABEL>
    </PREFIX>

    <SOURCE>
        <ID>TomTom</ID>
        <ENDPOINT>TomTom_mnpoi_Austria.nt</ENDPOINT>
        <VAR>?x</VAR>
        <PAGESIZE>-1</PAGESIZE>
        <RESTRICTION>?x a geos:Feature</RESTRICTION>
```

```
        <PROPERTY>geos:hasGeometry/geos:asWKT RENAME polygon</PROPERTY>
        <TYPE>NT</TYPE>
    </SOURCE>

    <TARGET>
        <ID>Herold</ID>
        <ENDPOINT>public.herold.nt</ENDPOINT>
        <VAR>?y</VAR>
        <PAGESIZE>-1</PAGESIZE>
        <RESTRICTION>?y a geos:Feature</RESTRICTION>
        <PROPERTY>geos:hasGeometry/geos:asWKT RENAME polygon</PROPERTY>
        <TYPE>NT</TYPE>
       </TARGET>

    <METRIC>geo_hausdorff(x.polygon, y.polygon)</METRIC>

    <ACCEPTANCE>
        <THRESHOLD>0.952380952</THRESHOLD>
        <FILE>verynear.nt</FILE>
        <RELATION>slipo:near</RELATION>
    </ACCEPTANCE>

    <REVIEW>
        <THRESHOLD>0.9090909090</THRESHOLD>
        <FILE>near.nt</FILE>
        <RELATION>slipo:near</RELATION>
    </REVIEW>

    <EXECUTION>
        <REWRITER>default</REWRITER>
        <PLANNER>default</PLANNER>
        <ENGINE>default</ENGINE>
    </EXECUTION>

    <OUTPUT>TAB</OUTPUT>

</LIMES>
```