KI - Künstliche Intelligenz (2021) 35:413–423
https://doi.org/10.1007/s13218-021-00713-x

SYSTEMS DESCRIPTION

LIMES: A Framework for Link Discovery on the Semantic Web

Axel-Cyrille Ngonga Ngomo[1] · **Mohamed Ahmed Sherif**[1] · **Kleanthi Georgala**[2] · **Mofeed Mohamed Hassan**[2] · **Kevin Dreßler**[2] · **Klaus Lyko**[2] · **Daniel Obraczka**[2] · **Tommaso Soru**[2]

Received: 16 April 2019 / Accepted: 11 February 2021 / Published online: 17 March 2021
© The Author(s) 2021

Abstract

The Linked Data paradigm builds upon the backbone of distributed knowledge bases connected by typed links. The mere volume of current knowledge bases as well as their sheer number pose two major challenges when aiming to support the computation of links across and within them. The first is that tools for link discovery have to be time-efficient when they compute links. Secondly, these tools have to produce links of high quality to serve the applications built upon Linked Data well. Solutions to the second problem build upon efficient computational approaches developed to solve the first and combine these with dedicated machine learning techniques. The current version of the LIMES framework is the product of seven years of research on these two challenges. A series of machine learning techniques and efficient computation approaches were developed and integrated into this framework to address the link discovery problem. The framework combines these diverse algorithms within a generic and extensible architecture. In this article, we give an overview of version 1.7.4 of the open-source release of the framework. In particular, we focus on an overview of the architecture of the framework, an intuition of its inner workings and a brief overview of the approaches it contains. Some descriptions of the applications within which the framework was used complete the paper. Our framework is open-source and available under a GNU license at https://github.com/dice-group/LIMES together with a user manual and a developer manual.

1 Introduction

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 [12]. In this paper, 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, see Sect. 2) to address two main challenges:

1. *Time-efficiency* The mere size of existing knowledge bases (e.g., 30 + billion triples in LinkedGeoData [39], 20 + billion triples in LinkedTCGA[2] [28]) 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 [15, 19], orthodromic spaces [17] and on filter-based paradigms [37].

✉ Axel-Cyrille Ngonga Ngomo
axel.ngomo@upb.de

Mohamed Ahmed Sherif
mohamed.sherif@upb.de

Kleanthi Georgala
georgala@informatik.uni-leipzig.de

Mofeed Mohamed Hassan
hassan@informatik.uni-leipzig.de

Kevin Dreßler
kevin.dressler@upb.de

Klaus Lyko
lyko@informatik.uni-leipzig.de

Daniel Obraczka
obraczka@informatik.uni-leipzig.de

Tommaso Soru
soru@informatik.uni-leipzig.de

[1] Paderborn University, Data Science Group, Technologiepark 6, 33100 Paderborn, Germany

[2] Institute of Computer Science, AKSW Group, University of Leipzig, Augustusplatz 10, 04009 Leipzig, Germany

---

[1] http://www.w3.org/DesignIssues/LinkedData.

[2] An RDF representation of The Cancer Genome Atlas.

2. *Accuracy* Central to this paper are the solutions to accuracy provided in the framework. Efficient solutions are of little help if the results they generate are inaccurate. Hence, Limes also accommodates dedicated machine-learning 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 [21–23], unsupervised learning [23] and even positive-only learning [29].

The main goal of this paper is to give an overview of the Limes framework and some of the applications within which it was used. We begin by presenting the link discovery problem and how we address this problem within a declarative setting (see Sect. 2). Then, we present our solution to the link discovery problem in the form of Limes and its architecture. The subsequent sections present the different families of algorithms implemented within the framework. We begin by a short overview of the algorithms that ensure the efficiency of the framework (see Sect. 4). Thereafter, we give an overview of algorithms that address the accuracy problem (see Sect. 5). We round up the core of the paper with some of the applications within which Limes was used, including benchmarking and the publication of 5-star linked datasets (Sect. 6). An overview of the evaluation results of algorithms included in Limes (Sect. 7) and a conclusion (Sect. 8) complete the paper.

2 The Link Discovery Problem

The formal specification of *Link Discovery* (LD) adopted herein is akin to that proposed in [16]. Given two (not necessarily distinct) sets S resp. T of source resp. target resources as well as a relation R, the goal of LD is 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 [7], Limes [16] and KnoFuss [26]) 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 that specifies M' a *link specification* (short LS; also called linkage rule in the literature, see e.g., [7]). Note that the LD problem can be expressed equivalently using distances instead of similarities in the following manner: Given two sets S and T of instances, a (complex) distance measure δ and a distance threshold $\theta \in [0, \infty[$, determine $M' = \{(s, t) \in S \times T : \delta(s, t) \leq \tau\}$.[3] Consequently, distance and similarities are used within link specifications in this paper.

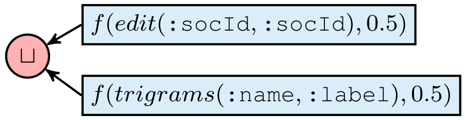

$f(edit(\texttt{:socId}, \texttt{:socId}), 0.5)$
⊔
$f(trigrams(\texttt{:name}, \texttt{:label}), 0.5)$

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

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 knowledge bases, time-efficient approaches able to reduce this runtime are hence a central component of Limes as they are necessary for link specifications to be computed in acceptable times. This efficient computation is in turn the proxy necessary for machine learning techniques to be used to optimize the choice of appropriate σ and θ and thus ensure that M' approximates M well even when M is large [12]).

Several approaches can be chosen when aiming to define the syntax and semantics of LSs in detail [7, 22, 26]. 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: (i) *similarity measures m*, which allow the comparison of property values or portions of the concise bound description of 2 resources and (ii) *operators op*, which can be used to combine these similarities into more complex specifications. 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 dubbed `edit`[4]. Every atomic measure is a measure. 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 $\mathcal{M}$.

We define a *filter* as a function $f(m, \theta)$. We call a specification *atomic* when it consists of exactly one filtering function. 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 $\sqcap$, $\sqcup$ and $\backslash$ as they are complete w.r.t. the Boolean algebra and frequently used to define LS. An example of a

---

[3] 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 $\theta = (1 + \tau)^{-1}$.

[4] We define the edit similarity of two strings s and t as $(1 + lev(s, t))^{-1}$, where lev stands for the Levenshtein distance.

Table 1 Link Specification Syntax and Semantics

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

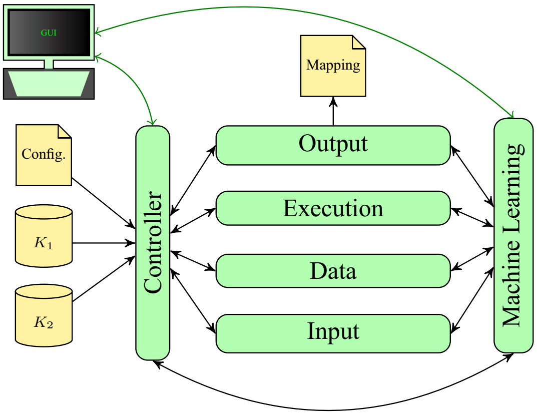


Fig. 2 General architecture of Limes

complex LS is given in Fig. 1. We denote the set of all LS as $\mathcal{L}$.

We define the semantics $[[L]]_M$ of a LS 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 LS L with respect to $S \times T$ contains the links that will be generated by L.

3 Architecture of the Framework

As shown in Fig. 2, the Limes framework consists of 6 main layers, each of which can be extended to accommodate new or improved functionality. The input to the framework is a *configuration file*, which describes how the sets S and T are to be retrieved from two knowledge bases K_1 and K_2 (e.g., remote SPARQL endpoints). Moreover, the configuration file describes how links are to be computed. To this end, the user can chose to either provide a LS explicitly or a configuration for a particular machine learning algorithm. The result of the framework is a (set of) mapping(s). In the following, we give an overview of the inner workings of each of the layers.

Controller layer The processing of the configuration file by the different layers is orchestrated by the *controller layer*. The controller instantiates the different implementations of input modules (e.g., reading data from files or from SPARQL endpoints), the data modules (e.g., file cache, in-memory), the execution modules (e.g., planners, number of processors for parallel processing) and the output modules (e.g., the serialization format and the output location) according to the configuration file or using default values. Once the layers have been instantiated, the configuration is forwarded to the input layer.

Input layer The input layer reads the configuration and extracts all the information necessary to execute the specification or the machine learning approach chosen by the user. This information includes (1) the location of the input knowledge bases K_1 and K_2 (e.g., SPARQL endpoints or files), (2) the specification of the sets S and T, (3) the measures and thresholds or the machine learning approach to use. The current version of Limes supports RDF configuration files based on the Limes Configuration Ontology (LCO)[5] (see Fig. 3) and XML configuration files based on the Limes Specification Language (LSL) [15]. If the configuration file is valid (w.r.t. the LCO or LSL), the input layer then calls its *query module*. This module uses the configuration for S and T to retrieve instances and properties from the input knowledge bases that adhere to the restrictions specified in the configuration file. All data retrieved is then forwarded to the data layer via the controller.

Data layer The *data layer* stores the input data gathered by the input layer using memory, file or hybrid storage techniques. Currently, Limes relies on a *hybrid cache* as default storage implementation. This implementation generates a hash for any set of resources specified in a configuration file and serializes this set into a file. Whenever faced with a new specification, the cache first determines whether the data required for the computation is already available locally by using the hash aforementioned. If the data is available, it is retrieved from the file into memory. In other cases, the hybrid cache retrieves the data as specified, generates a hash and caches it on the hard drive. By these means, the data layer addresses the practical problem of data availability, especially from remote data sources. Once all data is available, the controller layer chooses (based on the configuration file) between execution of complex measures specified by the user or running a machine learning algorithm.

Execution layer If the user specifies the LS to execute manually, the Limes controller layer calls the *execution layer*. The execution layer then re-writes, plans and executes the LS specified by the user. The *rewriter* aims to simplify complex LS by removing potential redundancy so as to eventually speed up the overall execution. To this end, it exploits the subset relations and the Boolean algebra which underpin the syntax and semantics of complex specifications. The *planner module* maps a LS to an equivalent execution plan, which is a sequence of atomic execution operations from

---

[5] https://w3id.org/dice-research/limes/lco.owl.

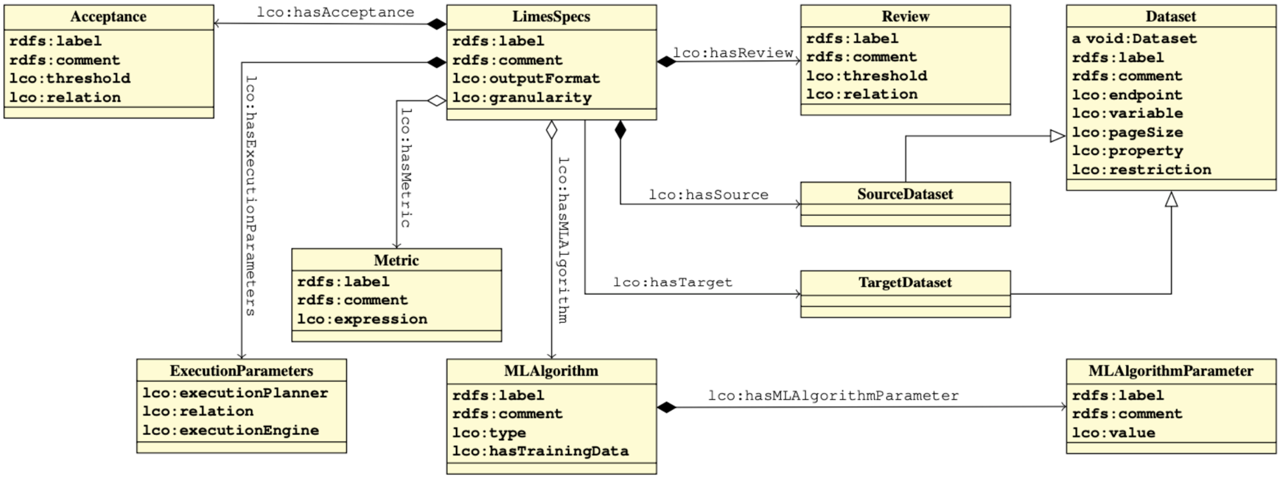
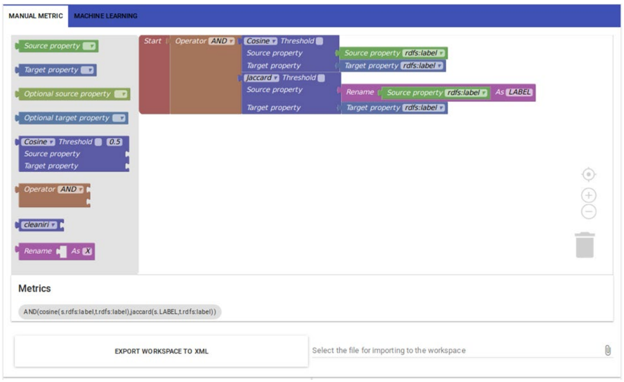


Fig. 3 LIMES Configuration ontology (LCO) ontology

which a mapping results. These operations include `EXECUTE` (running a particular atomic similarity measure, e.g., the trigram similarity[6]) and operations on mappings such as `UNION` (union of two mappings) and `DIFF` (difference of two mappings). For an atomic LS, i.e., a LS such that σ is an atomic similarity measure (e.g., trigram similarity, qgrams similarity, etc.), the planner module generates a *simple plan* with one `EXECUTE` instruction. For a complex LS, the planner module determines the order in which atomic LS should be executed as well as the sequence in which intermediary results should be processed. For example, the planner module may decide to first run some atomic LS L_1 and then filter the results using another atomic LS L_2 instead of running both L_1 and L_2 independently and merge the results. The *execution engine* is then responsible for executing the plan of an input LS and returns the set of potential links as a mapping.

Machine learning layer In case the user opts for using a machine learning approach, the LIMES controller calls the *machine learning layer*. This layer instantiates the machine learning algorithm selected by the user and executes it in tandem with the execution layer. To this end, the machine learning approaches begin by retrieving the training data provided by the user if required. The approaches then generate LSs, which are run in the execution layer. The resulting mappings are evaluated using quality measures (e.g., F-measure, pseudo-F-measure [23]) and returned to the user once a termination condition has been fulfilled. Currently, LIMES implements the EAGLE [22], COALA [24], EUCLID [23] and WOMBAT [29] algorithms. A brief overview of these approaches is given in Sect. 4.



Metrics

AND(cosine(s.rdfs:label,t.rdfs:label),jaccard(s.LABEL,t.rdfs:label))

Fig. 4 Manually created LS via LIMES web UI

Output layer The *output layer* serializes the output of LIMES. Currently, it supports the serialization into files in any RDF serialization also chosen by the user. In addition, the framework support CSV and XML as output formats.

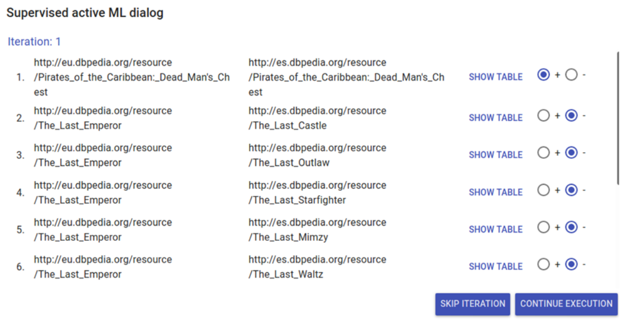
Web Graphical User Interface.

In addition to supporting configurations as input files, LIMES provides a Web-based graphical user interface (GUI)[7] to assist the end user during the LD process [34]. The framework supports the manual creation of LS as shown in Fig. 4 for users who already know which LS they would like to execute to achieve a certain linking task. However, most users do not know which LS suits their linking task best and therefore need help throughout this process. Hence, the GUI provides wizards which ease the LS creation process and allow configuration of the machine learning algorithms. After the selection of an algorithm, the user can modify the default configurations if necessary and run it. If a batch or

---

[6] We measure the similarity of two input strings by counting the number of *trigrams* (the group of three consecutive characters) they share. Formally, the trigram similarity is the normalized sum of absolute differences between tri-gram vectors of both the input strings.

[7] `https://limes.demos.dice-research.org`.


Supervised active ML dialog

Iteration: 1

1.	http://eu.dbpedia.org/resource/Pirates_of_the_Caribbean:_Dead_Man's_Chest	http://es.dbpedia.org/resource/Pirates_of_the_Caribbean:_Dead_Man's_Chest	SHOW TABLE	● + ○ -
2.	http://eu.dbpedia.org/resource/The_Last_Emperor	http://es.dbpedia.org/resource/The_Last_Castle	SHOW TABLE	○ + ● -
3.	http://eu.dbpedia.org/resource/The_Last_Emperor	http://es.dbpedia.org/resource/The_Last_Outlaw	SHOW TABLE	○ + ● -
4.	http://eu.dbpedia.org/resource/The_Last_Emperor	http://es.dbpedia.org/resource/The_Last_Starfighter	SHOW TABLE	○ + ● -
5.	http://eu.dbpedia.org/resource/The_Last_Emperor	http://es.dbpedia.org/resource/The_Last_Mimzy	SHOW TABLE	○ + ● -
6.	http://eu.dbpedia.org/resource/The_Last_Emperor	http://es.dbpedia.org/resource/The_Last_Waltz	SHOW TABLE	○ + ● -

Fig. 5 User feedback interface for active learning

unsupervised learning approach is selected, the execution is carried out once and the results are presented to the user. In the case of active learning, the LIMES web GUI presents the most highly informative link candidates to the user, who label these candidates as either a match or non-match (see Fig. 5). This procedure is repeated until the user is satisfied with the result. In all cases, the GUI allows the exportation of the configurations generated by machine learning approaches for future use.

4 Scalability Algorithms

As mentioned in the introduction, the scalability of LIMES is based on a number of time-efficient approaches for atomic LS that the framework supports. In the following, we summarize the approaches supported by LIMES version 1.5.0. By no means do we aim to give all technical details necessary to understand how these approaches work in detail. Interested readers are referred to the publications mentioned in each subsection.

The LIMES algorithm [19] is the first scalable approach developed explicitly for LD. The basic intuition behind the approach is based on regarding the LD problem as a bounded distance problem $\delta(s,t) \leq \tau$. If δ abides by the triangle inequality, the $\delta(s,t) \geq \delta(s,e) - \delta(e,t)$ for any e. Based on this insight, LIMES implements a space tiling approach that assigns each t to an exemplar e such that $\delta(e,t)$ is known for many t. Hence, for all s, LIMES can first compute a lower bound for the distance between s and t. If this lower bound is larger that τ, then the exact computation of $\delta(s,t)$ is not carried out, as (s,t) is guaranteed not to belong in the output of this atomic measure.

The $\mathcal{HR}^3$ algorithm [15] builds upon some of the insights of the LIMES algorithms within spaces with *Minkowski distances*. The basic insight behind the approach is that the equation $\delta(s,t) \leq \tau$ describes a hypersphere of radius τ around s. While computing spheres in multidimensional spaces can be time-consuming, hypercubes can be computed rapidly. Hence, the approach relies on the approximation hyperspheres with hypercubes. In contrast to previous approaches such as HYPPO [14], the approach combines hypercubes with an indexing approach and can hence discard a larger number of comparisons. One of the most important theoretical insights behind this approach is that it is the first approach guaranteed to being able to achieve the smallest possible number of comparisons when given sufficient memory. It is hence the first *reduction-ratio-optimal* approach for link discovery.

ORCHID [17] is a link discovery algorithm for geo-spatial data, which belong to the largest sources of Linked Data. Like $\mathcal{HR}^3$, ORCHID is reductio-ratio-optimal. In contrast to $\mathcal{HR}^3$, it operates in orthodromic spaces, in which Minkowski distances do not hold. To achieve the goal of reduction-ratio optimality, ORCHID uses a space discretization approach which relies on tiling the surface of the planet based on latitude and longitude information. The approach then only compares polygons $t \in T$ which lie within a certain range of $s \in S$. Like in $\mathcal{HR}^3$, the range is described via a set of hypercubes (i.e, squares in 2 dimensions). However, the shape that is to be approximated is a disk projected onto the surface of the sphere. The projection is accounted for by compensating for the curvature of the surface of the planet using an increased number of squares. See [31] for a survey of point-set distance measures implemented by LIMES and optimized using ORCHID.

AEGLE [6] is a time-efficient approach for computing temporal relations. Our approach supports all possible temporal relations between event data that can be modelled as intervals according to Allen's Interval Algebra [1]. The key idea behind the algorithm is to reduce the 13 Allen relations to 8 simpler relations, which compare either the beginning or the end of an event with the beginning or the end of another event. Given that time is ordered, AEGLE reduces the problem of computing these simple relations to the problem of matching entities across sorted lists, which can be carried out within a time complexity of $O(n \log n)$. The approach is guaranteed to compute complete results for any temporal relation.

RADON [30] addresses the efficient computation of topological relations on geo-spatial datasets. The main innovation of the approach is a novel sparse index for geo-spatial resources based on *minimum bounding boxes* (MBB)[8]. Based on this index, it is able to discard

---

[8] The smallest area box, within which all the points of a geo-spatial representation of a resource lie.

unnecessary computations for DE-9IM relations[9]. RADON applies a *swapping strategy* as its first step, where it swaps source and target datasets and computes the reverse[10] relation r' instead of r in case it finds that the *estimated total hypervolume* of the target is less that the one of the source dataset. Then in its second step, RADON utilizes a space tiling approach to insert all source and target geometries into an index I, which maps resources to sets of hypercubes. Finally, RADON implements the last speedup strategy using a MBB-based filtering technique, Like AEGLE, RADON is guaranteed to achieve a result completeness of 100% as it able to compute all topological relations of the DE-9IM model.

Keys for graphs are the projection of the concept of primary keys from relational databases. A key is thus a set of properties such that all instances of a given class are distinguishable from each other by their property values. Therefore, LD machine-learning-based algorithms optimize their performance using such keys. LIMES features ROCKER [35], a refinement operator-based approach for key discovery. The ROCKER algorithm first streams the input RDF graph to build a hash index including instance URIs and a concatenation of object values for each property. The indexes are then stored in a local database to enable an efficient computation of the discriminability score. Starting from the empty set, a refinement graph is visited, adding a property to the set at each refinement step. As keys abide by several monotonicities (e.g., every superset of a key is a key), additional optimizations allow scalability to large datasets. ROCKER has shown state-of-the-art results in terms of discovered keys, efficiency, and memory consumption [35].

LD frameworks rely on similarity measures such as the *Levenshtein* (or *edit*) distance to discover similar instances. The edit distance calculates the number of operations (i.e., addition, deletion or replacement of a character) needed to transform a string to another. However, the assumption that all operations must have the same weight does not always apply. For instance, the replacement of a `z` with an `s` often leads to the same word written in American and British English, respectively. This operation should have a lower weight than, e.g., an addition of the character `I` to the end of `King George I`. While manifold algorithms have been developed for the efficient discovery of similar strings using edit distance [9, 40], REEDED was developed to support also the rapid execution of weighted edit distances. Like the approaches aforementioned, REEDED joins two sets of strings by applying three filters to avoid computing the similarity values for all the pairs of strings. The (1) length-aware filter discards the pairs of strings having a too different length, the (2) character-aware filter selects only the pairs which do not differ by more than a given number of characters and the (3) verification filter finally verifies the weighted edit distance among the pairs against a threshold [37].

The *Jaro-Winkler* string similarity measure, which was originally designed for the deduplication of person names, is a common choice to generate links between knowledge bases based on labels. We therefore developed a runtime-optimized Jaro-Winkler algorithm for LD [3]. The approach consists of two steps: indexing and tree-pruning. The indexing step itself also consists of two phases: (1) strings of both source and target datasets get sorted into buckets based on upper and lower bounds pertaining to their lengths and (2) in each bucket, strings of the target dataset get indexed by adding them to a tree akin to a *trie* [4]. Tree pruning is then applied to cut off subtrees for which the similarity threshold cannot be achieved due to character mismatches. This significantly reduces the number of necessary Jaro-Winkler similarity computations.

In contrast to the algorithms above, HELIOS [18] addresses the scalability of link discovery by improving the planning of LSs. To this end, HELIOS implements a rewriter and a planner. The rewriter consists of a set of algebraic rules, which can transform portions of LS into formally equivalent expressions, which can be potentially executed faster. For example, expressions such as $AND(\sigma(s,t) \geq \theta_1, \sigma(s,t) \geq \theta_2)$ are transformed into $\sigma(s,t) \geq \max(\theta_1, \theta_2)$. In this particular simple example, the number of `EXECUTE` calls can hence be reduced from 2 to 1. The planner transforms a rewritten LS into a plan by aiming to find a sequence of execution steps that minimizes the total runtime of the LS. To this end, the planner relies on runtime approximations derived from linear regressions on large static dictionaries. CONDOR [5] builds upon HELIOS by extending the static planner with dynamic planning to achieve even better runtime.

5 Accuracy Algorithms

Pushing for efficiency of LD is a mere means to an end, which is the computation of accurate links between knowledge bases. The LIMES framework includes a family of machine learning approaches designed specifically for the purpose of link discovery. The core algorithms can be adapted for unsupervised, active and batch learning. In the following, we present the main ideas behind these approaches while refraining from providing complex technical details, which can be retrieved from the corresponding publications.

RAVEN [21] is the first approach designed to address the active learning of LS. The basic intuition behind the

---

[9] A standard used to describe the topological relations (e.g., covers and overlaps) between two geometries in two-dimensional space .

[10] Formally, the reverse relation r' of a relation r is defined as $r'(y,x) \Leftrightarrow r(x,y)$.

approach is that LS can be regarded as classifiers. Links that abide by the LS then belong to the class +1 while all other links belong to −1. Based on this intuition, the approach tackles the problem of learning LSs by first using the solution of the hospital-resident problem to detect potentially matching classes and properties in K_1 and K_2. The algorithm then assumes that it knows the type of LS that is to be learned (e.g., conjunctive specifications, in which all specifications operators are conjunctions). Based on this assumption and the available property matching, it learns the thresholds associated with each property pair iteratively by using an extension of the perceptron learning paradigm. RAVEN first applies perceptron learning to all training data that is made available by the user. Then, most informative unlabeled pairs (by virtue of their proximity to the decision boundary of the perceptron classifier) are sent as queries to the user. The labeled answers are finally added to the training data, which closes the loop. The iteration is carried on until the user terminates the process or a termination condition such as a maximal number of questions is reached.

EAGLE [22] implements a machine learning approach for LS of arbitrary complexity based on genetic programming. To this end, the approach models LS as trees, each of which stands for the genome of an individual in a population. EAGLE begins by generating a population of random LS, i.e., of individuals with random genomes. Based on training data or on an objective function, the algorithm determines the fitness of the individuals in the population. Operators such as mutation and crossover are then used to generate new members of the population. The fittest individuals are finally selected as members of the next population. When used as an active learning approach, EAGLE relies on a committee-based algorithm to determine most informative positive and negative examples. In addition to the classical entropy-based selection, the COALA [24] extension of EAGLE also allows the correlation between resources to be taken into consideration during the selection process for most information examples. All active learning versions of EAGLE ensure that each iteration of the approach only demands a small number of labels from the user. EAGLE is also designed to support the unsupervised learning approach of LS. In this case, the approach aims to find individuals that maximize so-called *pseudo F-Measures* [23, 25].

The ACIDS approach [36] targets instance matching by joining active learning with classification using linear Support Vector Machines (SVM). Given two instances s and t, the similarities among their datatype values are collected in a feature vector of size N. A pair (s, t) is represented as a point in the similarity space $[0, 1]^N$. At each iteration, pairs are (1) selected based on their proximity to the SVM classifier, (2) labeled by the user, and (3) added to the SVM training set. Mini-batches of pairs are labeled as positive if the instances are to be linked with an R, or negative otherwise. The SVM model is then built after each step, where the classifier is a hyperplane dividing the similarity space into two subspaces. String similarities are computed using a weighted Levenshtein distance. Using an update rule inspired by perceptron learning, edit operation weights are learned by maximizing the training F-score.

EUCLID [23] is a deterministic learning algorithm loosely based on RAVEN. EUCLID reuses the property matching techniques implemented in RAVEN and the idea of having known classifier shapes. In contrast to previous algorithms, EUCLID learns combinations of atomic LS by using a hierarchical search approach. While EUCLID was designed for unsupervised learning, it can also be used for supervised learning based on labeled training data [23].

One of the most crucial tasks when dealing with evolving datasets lies in updating the links from these data sets to other data sets. While supervised approaches have been devised to achieve this goal, they assume the provision of both positive and negative examples for links [20]. However, the links available on the Data Web only provide positive examples for relations and no negative ones, as the *open-world assumption* underlying the Web of Data suggests that the non-existence of a link between two resources cannot be understood as stating that these two resources are not related. In LIMES, we addressed this drawback by proposing WOMBAT [29], the first approach for learning links based on positive examples only. WOMBAT is inspired by the concept of generalisation in quasi-ordered spaces. Given a set of positive examples, it aims to find a classifier that covers a large number of positive examples (i.e., achieves a high recall on the positive examples) while still achieving a high precision. The simple version of WOMBAT relies on a two-step approach, which learns atomic LS and subsequently finds ways to combine them to achieve high F-measures. The complete version of the algorithm relies on a full-fledged upward refinement operator, which is guaranteed to find the best specification possible but scales less well than the simple version.

6 LIMES Use Cases

LIMES was already used in a large number of use cases. In the following, we present some of the datasets and other applications, in which techniques developed for LIMES or the LIMES framework itself, were used successfully.

Datasets LIMES is actively used by the linked data community to link new generated datasets into the already existing datasets in the LOD. For example, The multiligual dataset of SEMANTICQURAN [32] is linked using LIMES into 3 versions of the RDF representation of *Wiktionary* and to *DBpedia*. A second dataset also linked with LIMES is the LINKEDTCGA [28], an RDF representation of The Cancer

Genome Atlas. LIMES was used to compute the more than 16 million links which connect LINKEDTCGA with chromosomes from OMIM and HGNC.

Knowledge base repair The COLIBRI [25] approach attempts to repair instance knowledge in n knowledge bases. COLIBRI discovers links for transitive relations (e.g.,`owl:sameAs`) between instances in knowledge bases while correcting errors in the same knowledge bases. In contrast to most of the existing approaches, COLIBRI takes an n-set[11] of resources $K_1, \dots, K_n$ with $n \geq 2$ as input. Thereafter, COLIBRI relies on the LIMES 's unsupervised deterministic approach EUCLID to link each pair (K_i, K_j) of sets of resources (with $i \neq j$). The resource mappings resulting from the link discovery are then forwarded to a voting approach, which is able to detect poor mappings. This information is subsequently used to find sources of errors in the mappings, such as erroneous or missing information in the instances. The errors in the mappings are finally corrected and the approach starts the next linking iteration.

*Question answering*LIMES was also used in the creation of the knowledge base behind the question answering engine DEQA [8]. This engine was designed to be a comprehensive framework for deep web question answering. To this end, the engine provides functionality for the extraction of structured knowledge from (1) unstructured data sources (e.g., using FOX [38] and MAG [11]) and (2) the deep Web using the OXPath language.[12] The results of the extraction are linked with other knowledge sources on the Web using LIMES. In the example presented in [8], geo-spatial links such as `nearBy` were generated using LIMES ' approach $\mathcal{HR}^3$. The approach was deployed on a dataset combining data on flats in Oxford and geo-spatial information from LinkedGeoData and was able to support the answering of complex questions such as “`Give me a flat with 2 bedrooms near to a school`”.

Benchmarking While the creation of datasets for the Linked Data Web is an obvious use of the framework, LIMES and its algorithms can be used for several other purposes. A non-obvious application is the generation of benchmarks based on query logs [10, 27]. The DBpedia SPARQL benchmark [10] relies on DBpedia query logs to detect clusters of similar queries, which are used as templates for generating queries during the execution of the benchmark. Computing the similarity between the millions of queries in the DBpedia query log proved to be an impractical endeavor when implemented in a naïve fashion. The SPARQL queries were represented as binary features vectors. Using The $\mathcal{HR}^3$ algorithm [15]., the runtime of the similarity computations could be reduced drastically by expressing the problem of finding similar queries as a near-duplicate detection problem. The application of LIMES in the automatic benchmarks generation approach FEASIBLE [27] was also in the clustering of queries. The approach was designed to enable users to select the number of queries that their benchmark should generate. The examplar-based approach from LIMES [19] was used as a foundation for detecting the exact number of clusters required by the user while optimizing the homogeneity of the said clusters. The resulting benchmark was shown to outperform the state of the art in how well it encompasses the idiosyncrasies of the input query log.

Dataset enrichment Over the recent years, a few frameworks for RDF data enrichment such as LDIF[13] and DEER[14] have been developed. The frameworks provide enrichment methods such as entity recognition [38], link discovery and schema enrichment [2]. However, devising appropriate configurations for these tools can prove a difficult endeavour, as the tools require the right sequence of enrichment functions to be chosen and these functions to be configured adequately. In DEER [33], we address this problem by presenting a supervised machine learning approach for the automatic detection of enrichment pipelines based on a refinement operator and self-configuration algorithms for enrichment functions. The output of DEER is an enrichment pipeline that can be used on whole datasets to generate enriched versions.

Using link predicates such as `owl:sameAs`, the DEER 's *linking* enrichment operator is used to connect further datasets With the proper configurations, the *linking* enrichment operator can be used to generate arbitrary predicate types. For instance, using the `gn:nearby` for linking geographical resources. The idea of the self configuration of linking enrichment function is to (1) first perform an automatic source-target pairwise predicates matching. e.g., matching the `rdfs:label` property from source dataset to the property `skos:prefLabel` from target dataset., then (2) perform link discovery based on LIMES ' WOMBAT algorithm [23]

7 Evaluation

The approaches included in LIMES are the result of more than ten years of research, during which the state of the art evolved significantly. In Table 2, we give an overview of the performance improvement (w.r.t. runtime and/or accuracy) of a selection of algorithms currently implemented in LIMES. The improvements mentioned refer to improvements w.r.t. the state of art at the time at which the papers were written. We refer the readers to the corresponding research paper for

---

[11] An n-set is a set of magnitude n.

[12] http://www.oxpath.org/.

[13] http://ldif.wbsg.de/.

[14] https://github.com/dice-group/deer/.

Table 2 Evaluation of various algorithms used in Limes. The numbers provided were retrieved from the corresponding papers. These numbers are not upper bounds but merely reflect improvements observed empirically on particular datasets. O.o.m stands for orders of magnitude

Algorithm	Improvement		Ref.
	Runtime	F-Measure	
Limes	Up to 60-fold	–	[19]
$\mathcal{HR}^3$	Up to 7%	–	[15]
Orchid	Up to 2 o.o.m.	–	[17]
Aegle	Up to 4 o.o.m.	Correct (100%)	[6]
Radon	Up to 3 o.o.m.	Correct (100%)	[30]
Rocker	Up to 1 o.o.m.	Correct (100%)	[35]
Reeded	Up to 15-fold	Correct (100%)	[37]
Jaro-Winkler	Up to 5.5-fold	–	[3]
Helios	Up to 300%	–	[18]
Raven	Up to 33%	90% – 100%	[21]
Eagle	Up to 14-fold	> 90%	[22]
Acids	–	+ 7%	[36]
Coala	–	+ 25%	[24]
Euclid	Up to 33%	+ 11%	[23]
Wombat	–	+ 23%	[29]
Raven	Up to 33%	90% – 100%	[21]
Eagle	Up to 14-fold	Superior to 90%	[22]
Acids	–	Up to 7%	[36]
Coala	–	Up to 25%	[24]
Euclid	Up to 33%	Up to 11%	[23]
Wombat	–	Up to 23%	[29]

the evaluation settings and the experimental results for each of the algorithms mentioned in the table.

8 Conclusion and Future Work

With Limes, we offer an extensible framework for the discovery of links between knowledge bases. The framework has already been used in several applications and shown to be a reliable, near industry-grade framework for link discovery. The graphical user interface[15] and the manuals for users and developers[16], which accompany the tool, make it a viable framework for novices and expert users. While a number of challenges have been addressed in the framework, the increasing amount of Linked Data available on the Web and the large number of applications that rely on it demand that we address ever more challenges over the years to come. In particular, the intelligent use of memory (disk, RAM, etc.) becomes a problem of central importance when faced with amounts of RAM that cannot fit the sets S and T [13] or when faced with streaming or complex data (e.g., 5D geospatial data). Big Data frameworks such as SPARK[17] and FLINK[18] promise to alleviate this problem when used correctly. Also worth of investigation are approximative approaches able to cope with the noise in Linked Open Data sources as these data sources become increasingly important for real-life applications.

Acknowledgements This work was supported by the H2020 projects SLIPO (GA no. 731581) and HOBBIT (GA no. 688227), the Eurostars Project SAGE (GA no. E!10882) as well as the DFG project LinkingLOD (project no. NG 105/3-2) and the BMVI projects LIMBO (19F2029C) and OPAL (19F2028A).

Funding Open Access funding enabled and organized by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Allen JF (1983) Maintaining knowledge about temporal intervals. Commun ACM 26(11):832–843. https://doi.org/10.1145/182.358434
2. Bühmann L, Lehmann J (2013) Pattern based knowledge base enrichment. In: The semantic web—ISWC 2013—12th international semantic web conference, Sydney, NSW, Australia, October 21–25, 2013, Proceedings, Part I, pp 33–48. https://doi.org/10.1007/978-3-642-41335-3_3
3. Dreßler K, Ngomo AN (2017) On the efficient execution of bounded Jaro–Winkler distances. Semant Web 8(2):185–196. https://doi.org/10.3233/SW-150209
4. Fredkin E (1960) Trie memory. Commun ACM 3(9):490–499. https://doi.org/10.1145/367390.367400
5. Georgala K, Obraczka D, Ngomo ACN (2018) Dynamic planning for link discovery. In: European semantic web conference. Springer, pp 240–255
6. Georgala K, Sherif MA, Ngomo AN (2016) 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), pp 948–956. https://doi.org/10.3233/978-1-61499-672-9-948

---

[15] https://limes.demos.dice-research.org/.

[16] https://dice-group.github.io/LIMES/#/user_manual/index.

[17] http://spark.apache.org/.

[18] https://flink.apache.org/.

7. Isele R, Jentzsch A, Bizer C (2011) Efficient multidimensional blocking for link discovery without losing recall. In: Proceedings of the 14th international workshop on the web and databases 2011, WebDB 2011, Athens, Greece, June 12, 2011 . http://webdb2011.rutgers.edu/papers/Paper%2039/silk.pdf
8. Lehmann J, Furche T, Grasso G, Ngomo AN, Schallhart C, Sellers AJ, Unger C, Bühmann L, Gerber D, Höffner K, Liu D, Auer S (2012) 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, pp 131–147. https://doi.org/10.1007/978-3-642-35173-0_9
9. Li G, Deng D, Wang J, Feng J (2011) Pass-join: a partition-based method for similarity joins. Proc VLDB Endow 5(3):253–264
10. Morsey M, Lehmann J, Auer S, Ngonga Ngomo AC (2011) DBpedia SPARQL benchmark—performance assessment with real queries on real data. In: ISWC 2011. http://jens-lehmann.org/files/2011/dbpsb.pdf
11. Moussallem D, Usbeck R, Röeder M, Ngomo ACN (2017) Mag: a multilingual, knowledge-base agnostic and deterministic entity linking approach. In: Proceedings of the knowledge capture conference. ACM, p 9
12. Nentwig M, Hartung M, Ngomo AN, Rahm E (2017) A survey of current link discovery frameworks. Semant Web 8(3):419–436. https://doi.org/10.3233/SW-150210
13. Ngomo ACN, Hassan MM (2016) The lazy traveling salesman—memory management for large-scale link discovery. In: Sack H, Blomqvist E, d'Aquin M, Ghidini C, Ponzetto SP, Lange C (eds) ESWC, lecture notes in computer science, vol 9678, pp 423–438. Springer. http://dblp.uni-trier.de/db/conf/esws/eswc2016.html#NgomoH16
14. Ngomo AN (2011) A time-efficient hybrid approach to link discovery. In: Proceedings of the 6th international workshop on ontology matching, Bonn, Germany, October 24, 2011. http://ceur-ws.org/Vol-814/om2011_Tpaper1.pdf
15. Ngomo AN (2012) Link discovery with guaranteed reduction ratio in affine spaces with Minkowski measures. In: The semantic web—ISWC 2012—11th international semantic web conference, Boston, MA, USA, November 11–15, 2012, Proceedings, Part I, pp 378–393. https://doi.org/10.1007/978-3-642-35176-1_24
16. Ngomo AN (2012) On link discovery using a hybrid approach. J Data Semant 1(4):203–217. https://doi.org/10.1007/s13740-012-0012-y
17. Ngomo AN (2013) ORCHID—reduction-ratio-optimal computation of geo-spatial distances for link discovery. In: The semantic web—ISWC 2013—12th international semantic web conference, Sydney, NSW, Australia, October 21–25, 2013, Proceedings, Part I, pp 395–410 . https://doi.org/10.1007/978-3-642-41335-3_25
18. Ngomo AN (2014) HELIOS—execution optimization for link discovery. In: The semantic web—ISWC 2014—13th international semantic web conference, Riva del Garda, Italy, October 19–23, 2014. Proceedings, Part I, pp 17–32 . https://doi.org/10.1007/978-3-319-11964-9_2
19. Ngomo AN, Auer S (2011) LIMES—a time-efficient approach for large-scale link discovery on the web of data. In: IJCAI 2011, Proceedings of the 22nd international joint conference on artificial intelligence, Barcelona, Catalonia, Spain, July 16–22, 2011, pp 2312–2317. https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-385
20. Ngomo AN, Auer S, Lehmann J, Zaveri A (2014) Introduction to linked data and its lifecycle on the web. In: Reasoning web. Reasoning on the web in the big data era—10th international summer school 2014, Athens, Greece, September 8–13, 2014. Proceedings, pp 1–99. https://doi.org/10.1007/978-3-319-10587-1_1
21. Ngomo AN, Lehmann J, Auer S, Höffner K (2011) RAVEN—active learning of link specifications. In: Proceedings of the 6th international workshop on ontology matching, Bonn, Germany, October 24, 2011, pp 25–36. http://ceur-ws.org/Vol-814/om2011_Tpaper3.pdf
22. Ngomo AN, Lyko K (2012) 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, pp 149–163. https://doi.org/10.1007/978-3-642-30284-8_17
23. Ngomo AN, Lyko K (2013) Unsupervised learning of link specifications: deterministic vs. non-deterministic. In: Proceedings of the 8th international workshop on ontology matching co-located with the 12th international semantic web conference (ISWC 2013), Sydney, Australia, October 21, 2013, pp 25–36
24. Ngomo AN, Lyko K, Christen V (2013) 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, pp 442–456. https://doi.org/10.1007/978-3-642-38288-8_30
25. Ngomo AN, Sherif MA, Lyko K (2014) Unsupervised link discovery through knowledge base repair. In: The semantic web: trends and challenges—11th international conference, ESWC 2014, Anissaras, Crete, Greece, May 25–29, 2014. Proceedings, pp 380–394. https://doi.org/10.1007/978-3-319-07443-6_26
26. Nikolov A, Uren VS, Motta E (2007) 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, pp 185–186 . https://doi.org/10.1145/1298406.1298446
27. Saleem M, Mehmood Q, Ngonga Ngomo AC (2015) Feasible: A feature-based sparql benchmark generation framework. In: International semantic web conference (ISWC). http://svn.aksw.org/papers/2015/ISWC_FEASIBLE/public.pdf
28. Saleem M, Padmanabhuni SS, Ngomo AN, Almeida JS, Decker, S, Deus HF (2013) Linked cancer genome atlas database. In: I-SEMANTICS 2013—9th international conference on semantic systems, ISEM '13, Graz, Austria, September 4–6, 2013, pp 129–134. https://doi.org/10.1145/2506182.2506200
29. Sherif M, Ngonga Ngomo AC, Lehmann J (2017) WOMBAT—a generalization approach for automatic link discovery. In: 14th extended semantic web conference, Portorož, Slovenia, 28th May—1st June 2017. Springer. http://svn.aksw.org/papers/2017/ESWC_WOMBAT/public.pdf
30. Sherif MA, Dreßler K, Smeros P, Ngonga Ngomo AC (2017) Radon—rapid discovery of topological relations. In: Proceedings of the thirty-first AAAI conference on artificial intelligence (AAAI-17). https://svn.aksw.org/papers/2017/AAAI_RADON/public.pdf
31. Sherif MA, Ngomo ACN (2017) A systematic survey of point set distance measures for link discovery. Semant Web J. https://content.iospress.com/articles/semantic-web/sw285
32. Sherif MA, Ngomo AN (2015) Semantic quran. Semant Web 6(4):339–345. https://doi.org/10.3233/SW-140137
33. Sherif MA, Ngomo AN, Lehmann J (2015) Automating RDF dataset transformation and enrichment. In: The semantic web. Latest advances and new domains—12th European semantic web conference, ESWC 2015, Portoroz, Slovenia, May 31–June 4, 2015. Proceedings, pp 371–387. https://doi.org/10.1007/978-3-319-18818-8_23
34. Sherif MA, Pestryakova S, Dreßler K, Ngomo ACN (2019) Limeswebui—link discovery made simple. In: 18th international semantic web conference (ISWC 2019). CEUR-WS.org. http://svn.aksw.org/papers/2019/ISWC_limesWebUI/public.pdf
35. Soru T, Marx E, Ngomo AN (2015) ROCKER: a refinement operator for key discovery. In: Proceedings of the 24th international

conference on world wide web, WWW 2015, Florence, Italy, May 18–22, 2015, pp 1025–1033. https://doi.org/10.1145/2736277.2741642

36. Soru T, Ngomo AN (2012) 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, pp 97–112. https://doi.org/10.1007/978-3-642-37996-3_7
37. Soru T, Ngomo AN (2013) Rapid execution of weighted edit distances. In: Proceedings of the 8th international workshop on ontology matching co-located with the 12th international semantic web conference (ISWC 2013), Sydney, Australia, October 21, 2013, pp 1–12. http://ceur-ws.org/Vol-1111/om2013_Tpaper1.pdf
38. Speck R, Ngomo AN (2014) Ensemble learning for named entity recognition. In: The semantic web—ISWC 2014—13th international semantic web conference, Riva del Garda, Italy, October 19–23, 2014. Proceedings, Part I, pp 519–534. https://doi.org/10.1007/978-3-319-11964-9_33
39. Stadler C, Lehmann J, Höffner K, Auer S (2012) Linkedgeodata: a core for a web of spatial open data. Semant Web 3(4):333–354. https://doi.org/10.3233/SW-2011-0052
40. Xiao C, Wang W, Lin X (2008) Ed-join: an efficient algorithm for similarity joins with edit distance constraints. Proc VLDB Endow 1(1):933–944