J Data Semant (2012) 1:203–217
DOI 10.1007/s13740-012-0012-y

ORIGINAL ARTICLE

On Link Discovery using a Hybrid Approach

Axel-Cyrille Ngonga Ngomo

Received: 26 December 2011 / Revised: 18 September 2012 / Accepted: 20 September 2012 / Published online: 20 October 2012

Abstract With the growth of the Linked Data Web, time-efficient link discovery frameworks have become indispensable for implementing the fourth Linked Data principle, i.e., the provision of links between data sources. Due to the sheer size of the Data Web, detecting links even when using trivial link specifications based on a single property can be time-demanding. Moreover, non-trivial link discovery tasks require complex link specifications and are consequently even more challenging to optimize with respect to runtime. In this paper, we present a hybrid approach to link discovery that allows combining time-efficient algorithms specialized on specific data types. Especially, we present the HYPPO algorithm, which can process numeric data efficiently. These algorithms are combined by using original insights on the translation of complex link specifications to combinations of atomic specifications via a series of operations on sets and filters. We show in nine experiments that our approach outperforms SILK 2.5.1 with respect to runtime by up to four orders of magnitude.

1 Introduction

The Linked Data Web has evolved from 12 knowledge bases in May 2007 to 295 knowledge bases in September 2011[1] [1]. While the number of RDF triples available on the Linked Data Web has now surpassed 31 billion, the number of links still stagnates around 500 million. Consequently, less than 2 % of these triples are links between knowledge bases [17,19]. In addition, most knowledge bases are linked to only one knowledge base.[2] Yet, links between knowledge bases play a key role in important tasks such as cross-ontology question answering [14], large-scale inferences [26] and data integration [3]. Given the enormous amount of information available on the Linked Data Web, time-efficient link discovery (LD) frameworks have become indispensable for implementing the fourth Linked Data principle, i.e., the provision of links between data sources [17,27]. These frameworks rely on *link specifications*, which explicate conditions for computing new links between entities across knowledge bases. Due to the mere size of the Web of Data, detecting links even when using trivial specifications can be time-demanding. Moreover, non-trivial LD tasks require complex link specifications for discovering accurate links between instances and are consequently even more challenging to optimize with respect to runtime. Our approach is based on original insights on the distribution of property domains and ranges on the Web of Data. Based on these insights, we deduce requirements to efficient LD frameworks. We then use these requirements to specify the time-efficient approaches that underlie our framework, LIMES version 0.5.[3] We show that our framework outperforms the state of the art by orders of magnitude with respect to runtime while abiding to the restriction of not losing recall.[4]

[1] http://www4.wiwiss.fu-berlin.de/lodcloud/state/.

A.-C. Ngonga Ngomo (✉)
Department of Computer Science, University of Leipzig,
Johannisgasse 26, 04103 Leipzig, Germany
e-mail: ngonga@informatik.uni-leipzig.de
URL: http://bis.uni-leipzig.de/AxelNgonga

[2] While it is clear that most knowledge bases should be linked to several other knowledge bases, determining the desirable proportion of links on the Linked Data Cloud remains work in progress.

[3] An online demo of the framework can be found at http://limes.sf.net.

[4] Not losing recall is used in the same sense as [11] and means in this context that given a link specification, our approach is guaranteed to find all pairs of source and target instances that abide by the said specification.

The contributions of this paper are as follows:

1. We present a formal grammar for link specifications that encompass the functionality of state-of-the-art frameworks for LD.
2. Based on this grammar, we present a time-efficient approach for LD that is based on translating complex link specifications into a combination of atomic specifications via a concatenation of operations on sets and filter operations.
3. We use this method to enable the PPJoin+ [31] and EDJoin [30] algorithms to be used for processing complex link specifications.
4. We specify and evaluate HYPPO, a novel LD approach designed to operate on numeric values in metric spaces.
5. We evaluate our approach against SILK (version 2.5.1) within nine experiments and show that we outperform it by up to four orders of magnitude with respect to runtime while abiding to the constraint of not losing recall. Note that we chose SILK because it is the only freely available framework which supports specifications with a similarity measure whose complexity is similar to those supported by our approach.

The rest of this paper is structured as follows. In Sect. 2, we give an overview of related work on LD and related research fields. Section 3 presents the preliminaries to our work. This section encompasses a formal definition of the problem at hand and a short study of the distribution of property ranges on the Data Web. We use the results of this study to infer requirements to time-efficient LD frameworks. These requirements are the basis for Sect. 4, in which we specify a formal grammar for link specification and an approach to convert complex link specifications into an aggregation of atomic link specifications via set operations and filters. We subsequently present the core algorithms underlying our approach in Sect. 5. Section 6 gives a brief overview of the architecture of LIMES, the framework within which we implemented our approach. In Sect. 7, we evaluate our approaches in nine different experiments. After a discussion of our findings, we present our future work and conclude. Note that this paper extends upon the work presented in [16]. We carried out manifold extensions of our previous work, including the following:

– We present a broader motivation for our approach, including a study of property types across several knowledge bases.
– We extended the specification of the grammar underlying our framework and include a corresponding example.
– Moreover, we explicate the inner workings of LIMES by presenting its current architecture and Graphical User Interface.
– The experiments and results sections are completely new. All experiments presented in [16] were repeated with the (at the time of writing) newest release of SILK (version 2.5.1). In addition, six new experiments were designed to compare the runtime of HYPPO with that of SILK's numeric processing algorithm.

2 Related Work

Current frameworks for LD on the Web of Data can be subdivided into two categories: *domain-specific* and *universal* frameworks [17]. Domain-specific LD frameworks were developed with the aim of discovering links between knowledge bases from a particular domain. For example, RKB's Consistent Reference Service (RKB-CRS) [9] is a service that aims to compute URI equivalence within the domain of academia. It applies string similarity functions to properties such as publications' titles to detect initial equivalences. It then uses this knowledge to conclude on the equivalence of other entities such as authors, places of work and conferences. Another domain-specific tool is GNAT [23], which was designed especially for the music domain. To compute the similarity of resources, GNAT relies on audio fingerprinting. This approach can also combine the similarity of resources with that of their neighbors to compute `owl:sameAs` links. Further simple or domain-specific approaches can be found in [6,10,21,22,25].

Universal LD frameworks are designed to carry out mapping tasks independently from the domain of the source and target knowledge bases. For example, RDF-AI [24] implements a five-step approach that comprises the preprocessing, matching, fusion, interlink and post-processing of data sets. These modules can be configured by means of XML-files. The SILK framework [11] implements a LD approach dubbed Multiblock. In contrast to several other blocking approaches, MultiBlock is guaranteed to be lossless, which means that given a link specification, it is guaranteed to generate all triples that abide by the specification. To achieve this goal, Multiblock maps the different similarities included in complex link specifications to a multidimensional space. The coordinates of the resources that are to be linked are then computed by the means of an elaborate indexing scheme. The computation of links is finally achieved by computing overlapping blocks and carrying out similarity computations within these blocks only. Like LIMES, SILK can be configured using an XML-based language. The original LIMES approach [17] is a time-efficient and lossless approach for LD, which presupposes that the datasets to link are in a metric space. By using the characteristics of metric spaces, it begins by computing exemplars, which are prototypical points for portions of space. The approach then uses the triangle inequality to compute pessimistic approximations of distances. Based on these approximations, it can discard a large number of computations without losing links.

LD is closely related with record linkage [7,29] and deduplication [4], topics upon which a wealth of literature has been written (see e.g., [5] for a survey). Link discovery builds upon this research but goes beyond these two tasks by aiming to provide the means to link entities via any of the relations available on the Linked Data Web. For example, the LIMES framework has been used to link drugs with their active moiety and inactive ingredients as well as to link houses in Oxford with nearby geo-spatial entities [13].[5] Different blocking techniques such as standard blocking, sorted-neighborhood, bi-gram indexing, canopy clustering and adaptive blocking have been developed by the database community to address the problem of the quadratic time complexity of brute force comparison [12]. In addition, time-efficient approaches have been proposed to compute string similarities for record linkage, including AllPairs [2], PPJoin and PPJoin+ [31], EDJoin [30] and Trie-Join [28]. The most time-efficient string matching algorithms can only deal with simple link specifications (i.e., they can only compare entities by the means of one pair of property values), which is mostly insufficient when computing links between large knowledge bases. In this paper, we show how we can harness time-efficient approaches by combining them in a framework that enables them to be used when dealing with complex configurations. We integrate PPJoin+ and EDJoin in our framework. We also present the novel Hypersphere Approximation Algorithm (HYPPO), which ensures that our framework can deal efficiently with numeric values and consequently with the whole diversity of data types found on the Web of Data.

3 Preliminaries

3.1 Problem Definition

The goal of LD is to discover the set of pair of instances $(s, t) \in S \times T$ that are related by a relation R, where 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. We are aware that several categories of approaches can be envisaged for discovering links between instances, for example using formal inferences or semantic similarity functions. Throughout this paper, we will consider LD via properties. This is the most common definition of instance-based LD [17,27], which translates into the following formal definition:

Definition 1 (Link discovery). Given two sets S (source) and T (target) of instances, a (complex) similarity measure σ over the properties of $s \in S$ and $t \in T$ and a similarity threshold $\theta \in [0, 1]$, the goal of LD is to compute the set of pairs of instances $(s, t) \in S \times T$ such that $\sigma(s, t) \geq \theta$.

This problem can be expressed equivalently as follows:

Definition 2 (Link discovery on distances). Given two sets S and T of instances, a (complex) distance measure δ over the properties of $s \in S$ and $t \in T$ and a distance threshold $\theta \in [0, \infty[$, the goal of LD is to compute the set of pairs of instances $(s, t) \in S \times T$ such that $\delta(s, t) \leq \tau$.

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, distance and similarities are used interchangeably within our framework.

Although it is sometimes sufficient to define atomic similarity functions (i.e., similarity functions that operate on exactly one property pair) for LD, many LD problems demand the specification of complex similarity functions to return accurate links. For example, while the name of bands can be used for detecting duplicate bands across different knowledge bases, linking cities from different knowledge bases requires taking more properties into consideration (e.g., the different names of the cities as well as their latitude and longitude) to compute links accurately. The same holds for movies, where similarity functions based on properties such as the label and length of the movie as well as the name of its director are necessary to achieve high-accuracy link discovery. Consequently, linking on the Data Web demands frameworks that support complex link specifications.

3.2 Categorization of Approaches to Link Discovery

Three main categories of approaches can be envisaged when dealing with complex link specifications. The first type of approaches, which we dub *multidimensional*, address linking by mapping each instance to one or several points in a multi-dimensional (usually but not necessarily metric) space. They then use runtime reduction techniques, most commonly blocking [12], to discard comparisons that would not lead to a similarity above the user-given threshold. An example of such an approach is SILK's Multiblock [11]. The main advantage of such approaches is that they can exclude a large number of comparisons and that they are able to detect blocks that do not overlap significantly.

The second category of approaches, which we call *mono-dimensional*, relies on generating necessary constraints across single dimensions of the similarity space and using these for extracting linking candidates based on these constraints. These candidates are then merged and validated,

[5] The corresponding link specifications are available as download at http://aksw.org/Projects/LIMES.

Table 1 Distribution of datatype property ranges on the Web of Data

Knowledge bases	#Datatype properties*	#Data types	String	Numeric	Others
LGD	1,001	3	0	1,001	0
DBpedia	1,048	60	282	765	1
DailyMed*	17	1	17	0	0
Jamendo*	15	4	8	5	2
DBLP*	5	2	4	1	0

The data types for these knowledge bases marked with * were retrieved manually. LGD stands for LinkedGeoData

i.e., checked for whether they satisfy the linking condition specified by the user. The main advantage of such approaches is that they deal with only one dimension at once, thus making runtime reduction approaches computationally cheaper. On the other hand, discarding on only one dimension at once is usually less accurate since converting the user-given constraints to one dimension usually leads to necessary but not sufficient conditions. Consequently, mono-dimensional approaches generate more candidates that must be validated.

Hybrid approaches aim to make the best of both worlds by using runtime reduction techniques on the fragments of the link specifications where the blocks do not overlap significantly (like multidimensional approaches) to generate candidates. These candidates are then merged to generate the final list of links (mono-dimensional approaches). By these means, hybrid approaches thus aim to ensure that only the cheapest computations which discard a large percentage of non-matches are carried out. In this paper, we describe such an approach. The basic intuition behind our approach is that time-efficient linking frameworks designed for the Web of Data should provide dedicated algorithms for processing the most commonly used data types found on the Web of Data. These approaches should make use of the intrinsic characteristics of the data type that they process to operate as efficiently as possible. Our approach can efficiently process all property values that can be mapped efficiently to a metric space by applying the HYPPO algorithm in that space. In addition, it provides dedicated algorithms for processing data types that cannot (yet) be efficiently mapped to metric spaces. To determine the most common data types on the Web of Data, we carried out a short study of the distribution of property ranges across different knowledge bases and used it to specify our approach to LD.

3.3 Requirements to Link Discovery Frameworks

We studied the distribution of attribute values for datatype properties in several popular knowledge bases of different sizes. We divided the data types into three categories: strings (e.g., `rdfs:label`, `abstract`, etc.), numeric values[6] (e.g., populations, elevations, molecular weights, etc.) and others (mostly, URIs and URLs). The results of our study are shown in Table 1. Note that most of these values were retrieved manually as only a few knowledge bases provide `rdfs:range` information. While the distribution across the categories differs widely depending on the knowledge bases, one can easily see that the first two categories of data types (i.e., strings and numeric values) are the most commonly used. Consequently, LD frameworks for the Web of Data should be able to handle these two categories efficiently to enable time-efficient instance linking.

Our framework thus implements a *hybrid approach* that takes the distribution of data types into account by implementing dedicated functionality for processing simple linking tasks on strings (e.g., comparing the names of two cities) and combinations of numeric values (e.g., comparing the population and elevation of two cities) as well for merging their results to carry out complex linking tasks (e.g., linking cities using their labels, population and elevation) efficiently. To achieve this goal, our framework implements a grammar for transforming complex linking tasks into a combination of simple linking tasks and set operations. In the following, we present this grammar and then present efficacious approaches to carrying out simple linking tasks that lead to the time-efficient completion of linking tasks.

4 Link Specifications as Operations on Sets

In state-of-the-art LD frameworks, the condition for establishing links is usually expressed by using combinations of operations such as MAX (maximum), MIN (minimum) and linear combinations on binary similarity measures that compare property values of two instances $(s, t) \in S \times T$. Note that transformation operations may be applied to the property values (for example a lower-case transformation for strings) but do not affect our formal model. We present a formal grammar that encompasses complex link specifications as found in current LD frameworks (e.g., LIMES [19], SILK [11], KnoFuss [20]) and show how complex configurations resulting from this grammar can be translated into a sequence of set and filter operations on simple configurations. We use $\rightsquigarrow$ to denote generation rules for metrics and specifications. The symbol $\equiv$ denotes the equivalence of two specifications.

[6] Note that we consider numerical data to be data with a datatype such that there is a bijective mapping between the set of all elements of these datatypes and the real numbers.

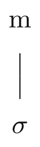


Fig. 1 Atomic measure

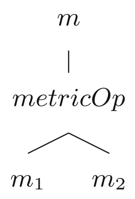


Fig. 2 Complex measure

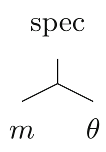


Fig. 3 Atomic specification

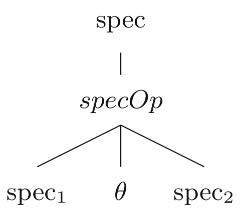


Fig. 4 Complex specification

Our definition of a link specification relies on the definition of *atomic similarity measures* and *similarity measures*. Generally, a similarity measure m is a function such that $m : S \times T \rightarrow [0, 1]$. We call a measure atomic (dubbed $atomicMeasure$) when it relies on exactly one similarity measure σ (e.g., trigrams similarity for strings) to compute the similarity of two instances s and t by the comparing certain property values (e.g., the labels or the elevations) of s and t. A similarity measure m is either an atomic similarity measure $atomicMeasure$ or the combination of two similarity measures via metric operators $metricOp$ such as *MAX,MIN* or linear combinations as implemented in LIMES. Thus, the following rule set for constructing metrics holds:

1. $m \rightsquigarrow atomicMeasure$
2. $m \rightsquigarrow metricOp(m_1, m_2)$

Note that frameworks differ in the type of operators they implement.

We call a link specification atomic ($atomicSpec$) if it compares the value of a measure m with a threshold τ, thus returning the pairs (s, t) that satisfy the condition $\sigma(s, t) \geq \theta$. A link specification $spec(m, \theta)$ is either an atomic link specification or the combination of two link specifications via

1. specification operators $specOp$ such as *AND* (the conditions of both specifications must be satisfied, equivalent to set intersection), *OR* (set union), *XOR* (symmetric set difference), or *DIFF* (set difference) and
2. a filtering threshold.

Thus, the following grammar for specifications holds:

1. $spec(m, \theta) \rightsquigarrow atomicSpec(m, \theta)$
2. $spec(m, \theta) \rightsquigarrow specOp(spec(m_1, \theta_1), spec(m_2, \theta_2), \theta_3)$

In the second rule, the threshold θ_3 is the filtering threshold, while θ_1 and θ_2 stand for the thresholds that are assigned to the measures m_1 and m_2. Each link specification that abides by the grammar specified above can be consistently transformed into a tree that contains the central constructs depicted in Figs. 1, 2, 3 and 4. Note that the set of operators on metrics and on link specification can be easily extended without altering the constructs allowed by our grammar.

Note that several time-efficient algorithms such as PPJoin+ operate solely on atomic measures and would not be usable if specifications could not be reduced to run only on atomic measures. For the operators MIN, MAX and linear combinations, we can reduce configurations that rely on complex measures to operations on configurations that rely on atomic measures via the following rules:

1. $spec(MAX(m_1, m_2), \theta) \equiv AND(spec(m_1, \theta), spec(m_2, \theta), 0)$
2. $spec(MIN(m_1, m_2), \theta) \equiv OR(spec(m_1, \theta), spec(m_2, \theta), 0)$
3. $spec(\alpha m_1 + \beta m_2, \theta) \equiv AND(spec(m_1, (\theta - \beta)/\alpha), spec(m_2, (\theta - \alpha)/\beta), \theta)$

Note that we can derive equivalent conditions on a smaller number of dimensions for the first two operations. Thus, we do not need to filter the results of the operators and can set the filtering threshold to 0. However, the simpler linking specifications that can be extracted for linear combinations are necessary to fulfill their premise, but not equivalent to the premise. Thus, in the case of linear combinations, it is important to validate the final set of candidates coming from the intersection of the two sets specified on a smaller number of dimensions against the original linear combination using the filtering threshold θ.

An example of such a transformation is shown in Figs. 5 and 6. Given these transformations, we can reduce all complex specifications that abide by our grammar to a sequence of set and filter operations on atomic specifications which rely on atomic measures. We can now apply time-efficient approaches designed for atomic measures on each category of data types to process even highly complex link specifications on the Web of Data.

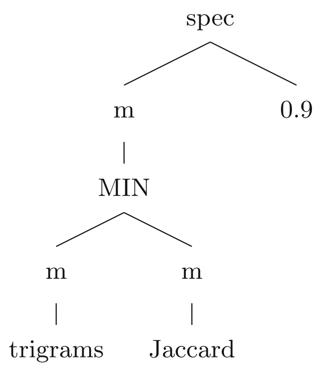


Fig. 5 Complex link specification based on metric operators

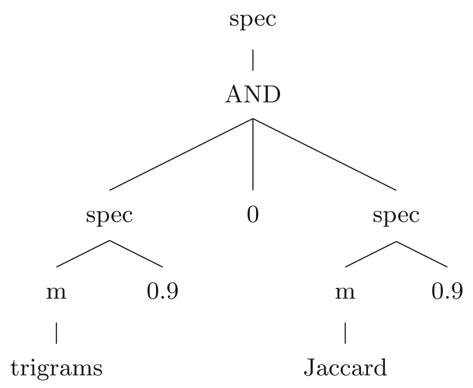


Fig. 6 Equivalent complex link specification based on specification operators

5 Processing Simple Configurations

Our framework implements a hybrid approach to LD and implements two main types of matchers for processing simple configurations: string matchers and numeric matchers. In the following, we present the idea behind the string matching algorithms we employ as well as present the novel HYPPO algorithm.

5.1 Processing Strings

The first category of matchers implemented in our framework deals exclusively with strings by harnessing the near-duplicate detection algorithms PPJoin+ [31] and EDJoin [30]. Instead of mapping strings to a vector space, PPJoin+ and EDJoin use a combination of three main insights to implement a time-efficient string comparison approach. First, they use the idea that strings with a given similarity must share a certain number of characters in their prefix to be able to have a similarity beyond the user-specified threshold. A similar intuition governs the suffix filtering implemented by these algorithms. Finally, the algorithms make use of the position of each word w in the index to retrieve a lower and upper bound of the index of the terms with which w might be similar. By combining these three approaches, PPJoin+ and EDJoin can discard a large number of non-matches. The integration of these two algorithms into our framework ensures that we can mitigate the pitfall of the time-demanding transformation of strings to vector spaces as implemented by multidimensional approaches. The main drawback of PPJoin+ and EDJoin is that they can only operate on one dimension [12]. However, by applying the transformations of configurations specified above, we make the algorithms at hand applicable to link discovery tasks with complex configurations. While mapping strings to a vector space demands some transformation steps and can be thus computationally demanding, all numeric values explicitly describe a vector space. The second approach implemented in our framework deals exclusively with numeric values and implements a novel approach dubbed *HYPPO*.

5.2 Processing Numeric Values

Current approaches to LD mostly focus on processing strings efficiently. Yet, as shown in Table 1, values that can be mapped to real numbers (e.g., elevations, temperatures, populations, etc.) play an important role on the Link Data Web. We developed the HYPPO algorithm to address the efficient processing of such property values. HYPPO stands for HYpersphere aPPrOximation algorithm. It addresses the problem of efficiently mapping instance pairs $(s, t) \in S \times T$ described using exclusively numeric values in a n-dimensional metric space. The approach assumes a distance metric δ for measuring the distance between objects and returns all pairs such that $\delta(s, t) \leq \theta$, where θ is a distance threshold. Let $\omega = (\omega_1, \ldots, \omega_n)$ and $x = (x_1, \ldots, x_n)$ be points in the n-dimensional space $\Omega = S \cup T$. The observation behind HYPPO is that in spaces (Ω, δ) with orthogonal, i.e., uncorrelated dimensions, the most common distance metrics can be decomposed into the combination of functions $\phi_{i,i\in\{1\ldots n\}}$ which operate on exactly one dimension of $\Omega : \delta = f(\phi_1, \ldots, \phi_n)$. For example, for Minkowski distances of order $p > 1$, $\phi_i(x, \omega) = |x_i - \omega_i|$ for all values of i and $\delta(x, \omega) = \sqrt[p]{\sum \phi_i(x, \omega)^p}$. Note that the Euclidean distance is the Minkowsky distance of order 2. The Minkowski distance can be extended further by weighting the different axes of Ω. In this case, $\delta(x, \omega) = \sqrt[p]{\sum \gamma_{ii}^p \phi_i(x, \omega)^p}$ and $\phi_i(x, \omega) = \gamma_{ii}|x_i - \omega_i|$, where γ_{ii} are the entries of a positive diagonal matrix.

Some distances do exist, which do not assume an orthogonal basis for the metric space. Mahalanobis distances for example are characterized by the equation $\delta(x, \omega) = \sqrt{(x - \omega)\Gamma(x - \omega)^{\mathrm{T}}}$, where Γ is a $n \times n$ covariance matrix. However, given that each space with correlated dimensions can always be transformed into an affine space with an orthonormal basis, we will assume in the remainder of this paper that the dimensions of Ω are independent. Given this assumption, it is important to notice that the following inequality holds:

$$\phi_i(x, \omega) \leq \delta(x, \omega), \quad (1)$$

ergo, $\delta(x, \omega)$ is the upper bound of $\phi_i(x, \omega)$. Note that this is the sole condition that we pose upon δ for HYPPO to be applicable. Also note that this condition can always be brought about in a metric space by transforming its basis into an orthogonal basis.

The basic intuition behind HYPPO is that the hypersphere $H(\omega, \theta) = \{x \in \Omega : \delta(x, \omega) \leq \theta\}$ is a subset of the hypercube V defined as $V(\omega, \theta) = \{x \in \Omega : \forall i \in \{1 \dots n\}, \phi_i(x_i, \omega_i) \leq \theta\}$ due to inequality (1). Consequently, one can reduce the number of comparisons necessary to detect all elements of $H(\omega, \theta)$ by discarding all elements which are not in $V(\omega, \theta)$ as non-matches. HYPPO uses this intuition by implementing a two-step approach to LD. First, it divides Ω into hypercubes of the same volume. Second, it compares each $s \in S$ with those $t \in T$ that lie in cubes at a distance below θ. Note that these two steps differ from the steps followed by similar algorithms (such as blocking) in two ways. First, we do not use only one but several hypercubes to approximate $H(\omega, \theta)$. Most blocking approach rely on finding *one block* that contains the elements that are to be compared with ω [12]. Note that in contrast to most blocking techniques, HYPPO is guaranteed to be lossless as H is completely enclosed in V.

Formally, let $\Delta = \theta/\alpha$. We call $\alpha \in \mathbb{N}$ the granularity parameter. HYPPO first tiles Ω into the adjacent hypercubes (short: cubes) C that contain all the points ω such that $\forall i \in \{1 \dots n\}, c_i\Delta \leq \omega_i < (c_i + 1)\Delta, (c_1, \dots, c_n) \in \mathbb{N}^n$. We call the vector $(c_1, \dots, c_n)$ as the coordinates of the cube C. Each point $\omega \in \Omega$ lies in the cube $C(\omega)$ with coordinates $(\lfloor \omega_i/\Delta \rfloor)_{i=1\dots n}$. Given such a space tiling and inequality (1), it is obvious that all elements of $H(\omega, \theta)$ lie in the set $\mathfrak{C}(\omega, \alpha)$ of cubes such that $\forall i \in \{1 \dots n\} : |c_i - c(\omega)_i| \leq \alpha$. Figure 7 shows examples of space tilings for different values of α.

The accuracy of the approximation performed by HYPPO can be computed easily: the number of cubes that approximate $H(\omega, \theta)$ is $(2\alpha + 1)^n$, leading to a total volume $V_{\mathfrak{C}}(\alpha, \theta) = ((2\alpha + 1)\Delta)^n = (\frac{2\alpha+1}{\alpha}\theta)^n$ that approximates $H(\omega, \theta)$. The volume $V_H(\theta)$ of $H(\omega, \theta)$ is given by $S_n\theta^n$, where S_n is the volume of a unit sphere in n dimensions, i.e., 2 for $n = 1, \pi$ for $n = 2$, $\frac{4\pi}{3}$ for $n = 3$ and so on. The approximation ratio

$$\frac{V_{\mathfrak{C}}(\alpha, \theta)}{V_H(\theta)} = \frac{(2\alpha + 1)^n}{S(n)\alpha^n}, \quad (2)$$

permits to determine the accuracy of HYPPO's approximation as shown in Fig. 8 for dimensions between 1 and 3 and values of α up to 10 . Note that $V_{\mathfrak{C}}$ and V_H do not depend on ω and that $\frac{V_{\mathfrak{C}}(\alpha,\theta)}{V_H(\theta)}$ does not depend on θ. Furthermore, note that the higher the value of α, the better the accuracy of HYPPO. Yet, higher values of α also lead to an exponentially growing number of hypercubes $|\mathfrak{C}(\omega, \alpha)|$ and thus to longer runtimes when constructing $\mathfrak{C}(\omega, \alpha)$ to approximate $H(\omega, \theta)$. Once the space tiling has been completed, all that remains to do is to compare each $s \in S$ with all the $t \in T \cap (\bigcup C \in \mathfrak{C}(\omega, \alpha))$ and to return those pairs of entities such that $\delta(s, t) \leq \theta$. Algorithm 1 shows HYPPO's pseudocode.

Algorithm 1 Current implementation of HYPPO

```
Require: Source data $S$
Require: Target data $T$
Require: Distance threshold $\theta$
Require: Distance function $\delta$
Require: Granularity factor $\alpha$
  Mapping $M := \emptyset$
  $\Delta = \theta/\alpha$
  for $\omega \in S \cup T$ do
    $C(\lfloor\omega_1/\Delta\rfloor, ..., \lfloor\omega_n/\Delta\rfloor) := C(\lfloor\omega_1/\Delta\rfloor, ..., \lfloor\omega_n/\Delta\rfloor) \cup \{\omega\}$
  end for
  for $s \in S$ do
    for $C \in \mathfrak{C}(s, \alpha)$ do
      for $t \in C \cap T$ do
        if $\delta(s, t) \leq \theta$ then
          $M := M \cup \{(s, t)\}$
        end if
      end for
    end for
  end for
  return $M$
```

6 Implementation

The approach presented in this paper was implemented in the LIMES Framework[7] version 0.5. The input of LIMES is a XML file that allows configuring the processing pipeline of LIMES. An example of such as file is shown in Fig. 11. The core of the LIMES' implementation is shown in Fig. 9 and consists of five main layers. The *input layer* implements a series of modules that allow the retrieval of data not only from SPARQL endpoints but also from RDF serializations (such as N3 and Turtle) and Character-Separated Value (CSV) files. In our example, LIMES retrieves data from two SPARQL endpoints. The data items retrieved by the input layer are transmitted to the *preprocessing layer*, which

[7] See http://limes.sf.net. The user manual available at the same page describes the architecture presented herein in more detail.

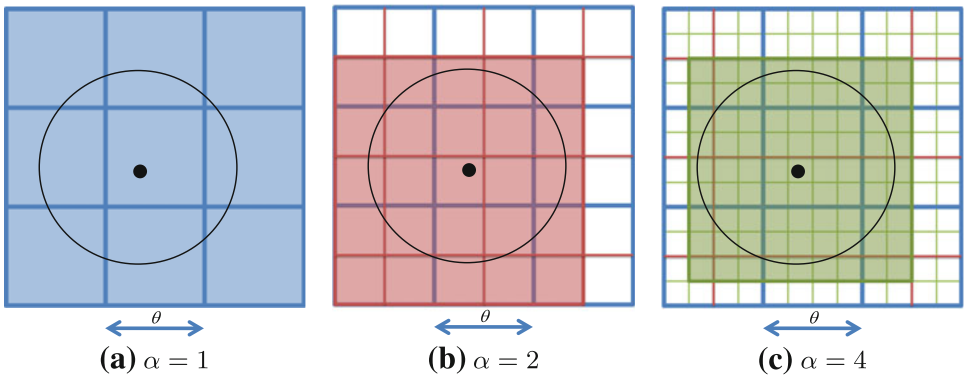


Fig. 7 Space tiling for different values of α. The *colored squares* show the set of elements that must be compared with the instance located at the *black dot*. *The points within the circle* lie within the distance θ of the *black dot*

implements several functions for transforming the input data into less noisy data. This layer implements methods for regular expression replacement, lower and upper case transformation, string cleaning and stripping and many more. The preprocessing methods can be combined by the user to a preprocessing pipeline at will. In our example, the property values that are strings are first converted to the lowercase and all language tags are then removed. The preprocessed data are then stored in the *data layer*, which implements several strategies for storing data including memory and file caching methods. The mapping strategy specified in the configuration file is then carried out by the *processing layer* based on the data in the data layer. Several mappers for complex link specifications and atomic mappers for atomic configurations are implemented and allow for a speedy execution of LD tasks. In our examples, two places are considered to stand for the same real-world entity if their transformed labels have a trigrams similarity of at least 0.9 and the difference of their populations in the two knowledge bases is maximally 999 (i.e, the numeric similarity of their population is at least 10^{-3}). The resulting links are serialized by the *output layer*, which also allows giving out the results of the computation in several formats including Turtle, N3 and CSV files. Note that LIMES can be extended easily to accommodate user-specific functionality thanks to its highly modular architecture. In addition, LIMES provides a graphical user interface dubbed COLANUT (COmplex Linking in A NUTshell, see Fig. 10), which guides the user through the link specification process by providing statistical ontology matching function based on stable marriage algorithms [8,15] as described in [18].

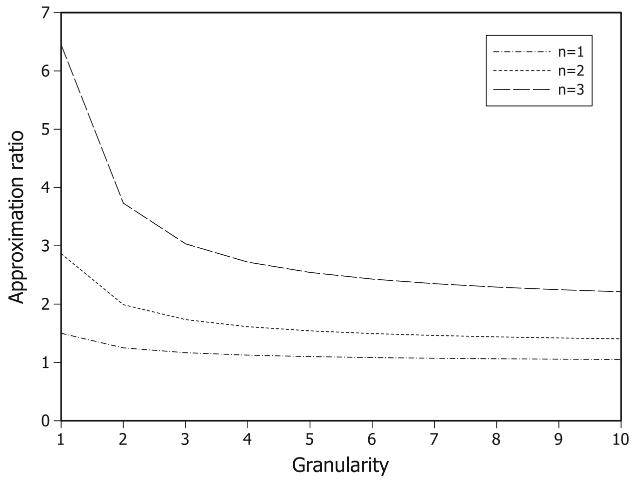


Fig. 8 Approximation ratio for $n \in \{1, 2, 3\}$. The *x*-axis shows values of α while the *y*-axis shows the approximation ratios

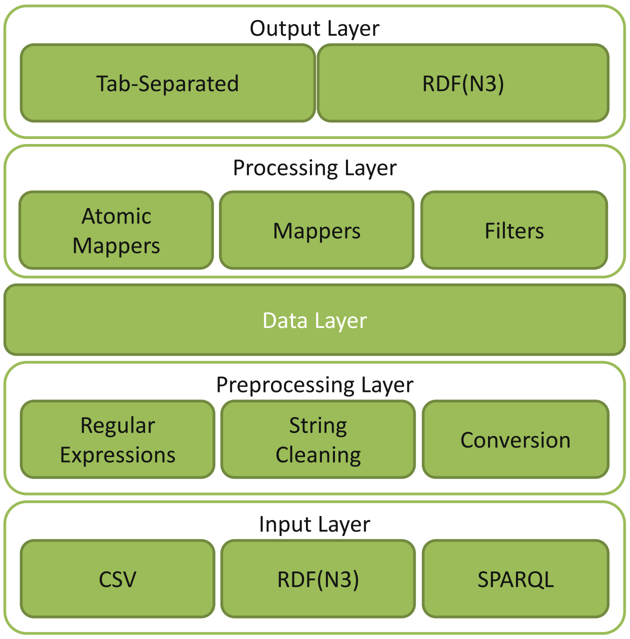


Fig. 9 Architecture of LIMES

7 Evaluation

We compared our approach with that implemented in SILK version 2.5.1. We chose SILK because (to the best of our knowledge) it is the only other LD framework that allows the specification of such complex linking experiments.

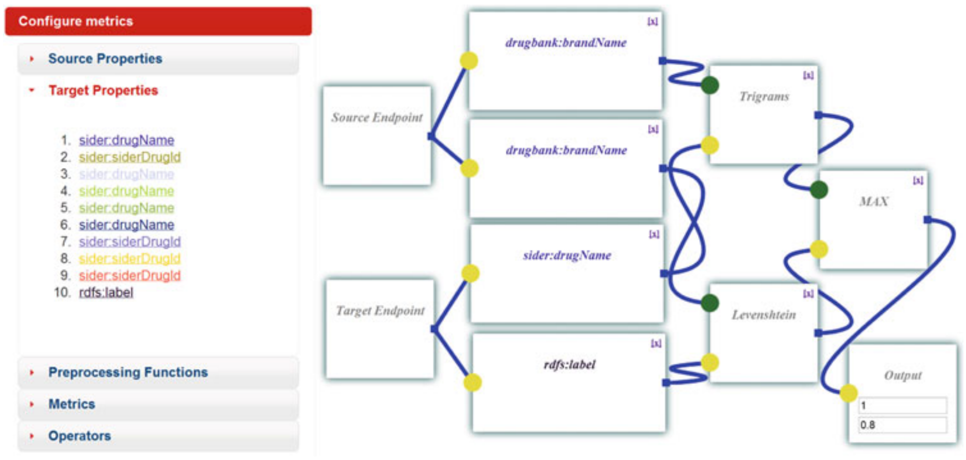


Fig. 10 Colanut's main window. The user can combine properties, metrics and transformation functions on the *left*. Note that the properties are *color-coded*. Properties across two knowledge bases that share the *same color* are suggested to be equivalent by Colanut's property matcher

We could not separate the fetching and the indexing of the data in SILK as these two processes are intertwined. To ensure that our evaluation was not biased towards LIMES, we only measured the time needed by SILK to compare links. This was realized by allowing both tools to download all data necessary for the linking experiments unto the hard drive of our machine. Note that SILK indexes the data. It downloads and stores the data and the index locally on the hard drive. LIMES on the other hand simply downloads the data and serializes them into a file. We ran the experiments by allowing both systems to retrieve the data necessary for linking from the hard drive of the local machine. As all computations are done on the fly in LIMES (i.e., no pre-indexing or data segmentation is carried out during the download of the data), our measurements of LIMES' overall runtime display the sum of the tiling and link computation time, while the measurements of SILK reflect exclusively the time necessary for the computation of links.

We ran all experiments on the same computer running a Windows 7 Enterprise 64-bit installation on a 2.8 GHz i7 processor with 8 GB RAM. The JVM was allocated 4 GB RAM in the first series of experiments and 7.4 GB RAM in the second series of experiments. All experiments were carried out five times except when stated otherwise. In all cases, we report best runtimes. The a-priori complexity of the experiments was computed as $n|S||T|$ where n is the number of property pairs used during the experiments, $|S|$ is the size of the source knowledge base and $|T|$ is the size of the target knowledge base.

7.1 Experiments with HYPPO

In our first series of experiments, we aimed at determining the behavior of HYPPO on problems with a varying number of dimensions. Thus, we evaluated HYPPO within six use cases of 1, 2 or 3 dimensions and compared it with SILK. To ensure that we compared solely HYPPO and SILK, we designed experiments that aimed at deduplicating instances in DBpedia and executed solely the fragment of the specification that dealt with numeric values. We chose DBpedia because it contains a large amount of general knowledge. Note that all data sets were retrieved from a local copy of DBpedia 3.6. We ran all experiments with distance thresholds (θ) and granularity (α) values between 1 and 16. In all experiments, we used the normed similarity based on the Euclidean Distance.

The goal of the first experiment, dubbed *Towns*, was to deduplicate towns based on their population. The second experiment, dubbed *Books*, was also one-dimensional and aimed at finding duplicate records of books based on the number of pages of each book. The third and fourth experiment were two-dimensional. *Vacations*, the third experiment, aimed to detect similar towns by comparing their population and elevation. The fourth experiment, dubbed *Actors*, deduplicated people based on their height and weight. The fifth and sixth experiments were carried out on three dimensions. The fifth experiment was designed to detect duplicate television *Series* using their number of episodes, number of seasons and runtime for a movie portal. Finally, the sixth experiment generated links between cities by means of their elevation, land area and water area for studies on *Hydrology*. An overview of the experiments is given in Table 2.

The results of this series of experiments are shown in Figs. 12, 13 and 14. With respect to time complexity, our evaluation hints towards the runtime of HYPPO growing linearly with the threshold θ. We outperform SILK by up to four orders of magnitude, e.g., in the Series experiment (see Fig. 14a). The results of most of these experiments suggest that the runtimes of HYPPO depend significantly on the value of α. We expected the runtimes to decrease significantly up to a certain value of α and then to increase or remain constant. Our experiments confirm this behavior. Overall, setting α to 4 leads to a good trade-off between the exponentially growing

Table 2 Summary of experimental setups for experiments on HYPPO

Experiment	# Instances	A-priori complexity	# Dimensions
Town	27,525	0.76×10^9	1
Books	14,714	0.22×10^9	1
Vacations	21,925	0.96×10^9	2
Actors	15,909	0.51×10^9	2
Series	4,841	0.07×10^9	3
Hydrology	19,095	1.09×10^9	3

Fig. 11 Example of a LIMES configuration file. The corresponding link discovery task maps places from LinkedGeoData and Geonames by measuring the similarity of their labels and of their populations

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE LIMES SYSTEM "limes.dtd">
<LIMES>
<PREFIX>
  <NAMESPACE>http://dbpedia.org/ontology/</NAMESPACE>
  <LABEL>dbpedia-o</LABEL></PREFIX>
<PREFIX>
  <NAMESPACE>http://www.w3.org/1999/02/22-rdf-syntax-ns#</NAMESPACE>
  <LABEL>rdf</LABEL></PREFIX>
<PREFIX>
  <NAMESPACE>http://www.w3.org/2000/01/rdf-schema#</NAMESPACE>
  <LABEL>rdfs</LABEL></PREFIX>
<PREFIX>
  <NAMESPACE>http://www.w3.org/2002/07/owl#</NAMESPACE>
  <LABEL>owl</LABEL></PREFIX>
<PREFIX>
  <NAMESPACE>http://www.geonames.org/ontology#</NAMESPACE>
  <LABEL>geonames</LABEL></PREFIX>
<PREFIX>
  <NAMESPACE>http://linkedgeodata.org/property/</NAMESPACE>
  <LABEL>lgdp</LABEL></PREFIX>
<PREFIX>
  <NAMESPACE>http://linkedgeodata.org/ontology/</NAMESPACE>
  <LABEL>lgdo</LABEL></PREFIX>
<SOURCE>
  <ID>geonames</ID>
  <ENDPOINT>http://lgd.aksw.org:8900/sparql</ENDPOINT>
  <GRAPH>http://geonames.org</GRAPH>
  <VAR>?x</VAR>
  <PAGESIZE>-1</PAGESIZE>
  <RESTRICTION> </RESTRICTION>
  <PROPERTY>geonames:population</PROPERTY>
  <PROPERTY>geonames:alternateName AS lowercase->nolang</PROPERTY>
  <PROPERTY>geonames:name AS lowercase->nolang</PROPERTY>
</SOURCE>
<TARGET>
  <ID>lgd</ID>
  <ENDPOINT>http://linkedgeodata.org/sparql/</ENDPOINT>
  <VAR>?y</VAR>
  <PAGESIZE>1000</PAGESIZE>
  <RESTRICTION>?y rdf:type lgdo:Place</RESTRICTION>
  <PROPERTY>lgdo:population</PROPERTY>
  <PROPERTY>rdfs:label AS lowercase->nolang</PROPERTY>
</TARGET>
<METRIC>
  AND(euclidean(x.dbpedia-o:populationTotal,y.lgdo:population)|0.001,
  OR(trigrams(x.geonames:alternateName, y.rdfs:label)|0.9,
  trigrams(x.geonames:name, y.rdfs:label)|0.9)|0.9)
</METRIC>
<ACCEPTANCE>
  <THRESHOLD>0.001</THRESHOLD>
  <FILE>places.ttl</FILE>
  <RELATION>owl:sameAs</RELATION>
</ACCEPTANCE>
<REVIEW>
  <THRESHOLD>0</THRESHOLD>
  <FILE>places_review.ttl</FILE>
  <RELATION>owl:sameAs</RELATION>
</REVIEW>
<EXECUTION>Simple</EXECUTION>
<GRANULARITY>4</GRANULARITY>
<OUTPUT>TURTLE</OUTPUT>
</LIMES>
```

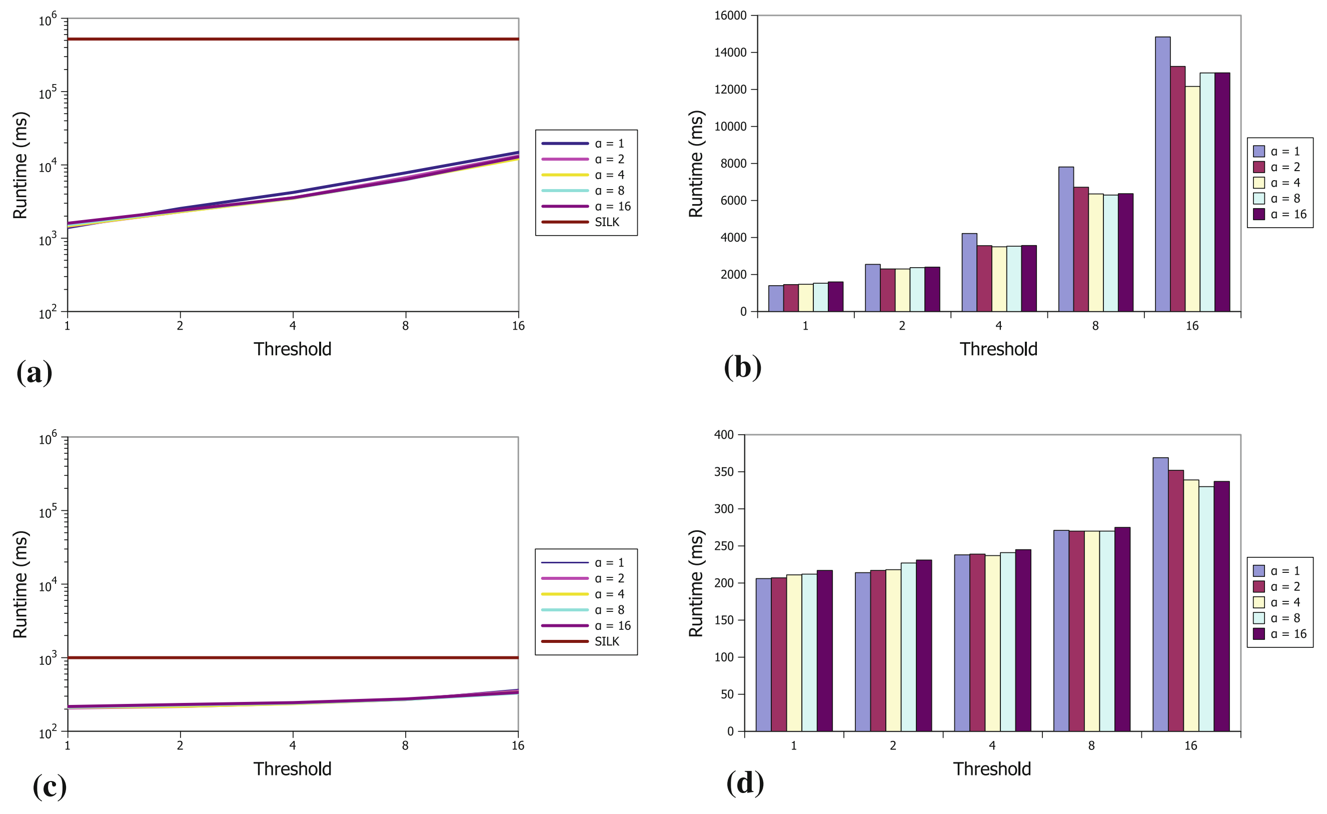


Fig. 12 Results of the one-dimensional experiments (Books and Towns). The right figures are in log-log-scale, whilst the left ones show the runtimes of LIMES on a linear scale. Note that given the size of the experiment, the different values of α are superimposed. **a** Comparison with SILK in Towns experiment. **b** Comparison for different α-values in Towns experiment. **c** Comparison with SILK in Books experiment. **d** Comparison for different α-values in Books experiment

number of cubes and the better approximation reached for large values of α. Another noteworthy observation is that the runtime of LIMES in the Hydrology experiment decreased when θ was changed from 1 to 2. This was simply due to the results of the experiments being similar for these different values of θ. The coarser approximation for larger θ led to less hypercubes being generated for in the experiments, which consequently led to a smaller total runtime.

7.2 Experiments with LIMES

We compared our whole framework with SILK2.5.1 in three experiments of different complexity based on geographic data. We chose to use geographic datasets because they are large and require the use of several attributes for linking. Given the complexity of the data, having a time-efficient indexing scheme plays a central role in these experiments. Thus, we first measured the runtime without indexing (like in the previous experiments). In addition, we approximated the runtime necessary for the ARQ[8] library (which is used in both tools) to fetch the data to compute from the endpoints. By these means, we could approximate the total runtime of both approaches including indexing. In the first experiment, we computed links between *villages* in DBpedia and LinkedGeoData based on the `rdfs:label` and the `population` of instances. The link condition was twofold: (1) the difference in population had to be lower or equal to θ and (2) the labels had to have a trigram similarity larger or equal to τ. In the second experiment, we aimed to link towns and *cities* from DBpedia with populated places in Geonames. We used the names (`gn:name`), alternate names (`gn:alternateName`) and `population` of cities as criteria for the comparison. Finally, we computed links between *Geo-locations* in LinkedGeoData and GeoNames using four combinations of criteria for comparing entities: their longitude (`wgs84:long`), latitude (`wgs84:lat`), preferred names and names.

The setup of the experiments is summarized in Table 3. We used two threshold setups. In the *strict setup*, the similarity threshold τ_s on strings was set to 0.9, the maximal difference in population θ_p was set to 9 and the maximal difference in latitude and longitude θ_l was set to 1. In the *lenient setup*, τ_s was set to 0.7 and θ_p to 19. The lenient setup was not used in the Geo-Locations experiments because it led to too many links, which filled up the 7.4 G of RAM allocated to both tools and led to swapping, thus falsifying the evaluation

8 http://jena.sourceforge.net/ARQ/.

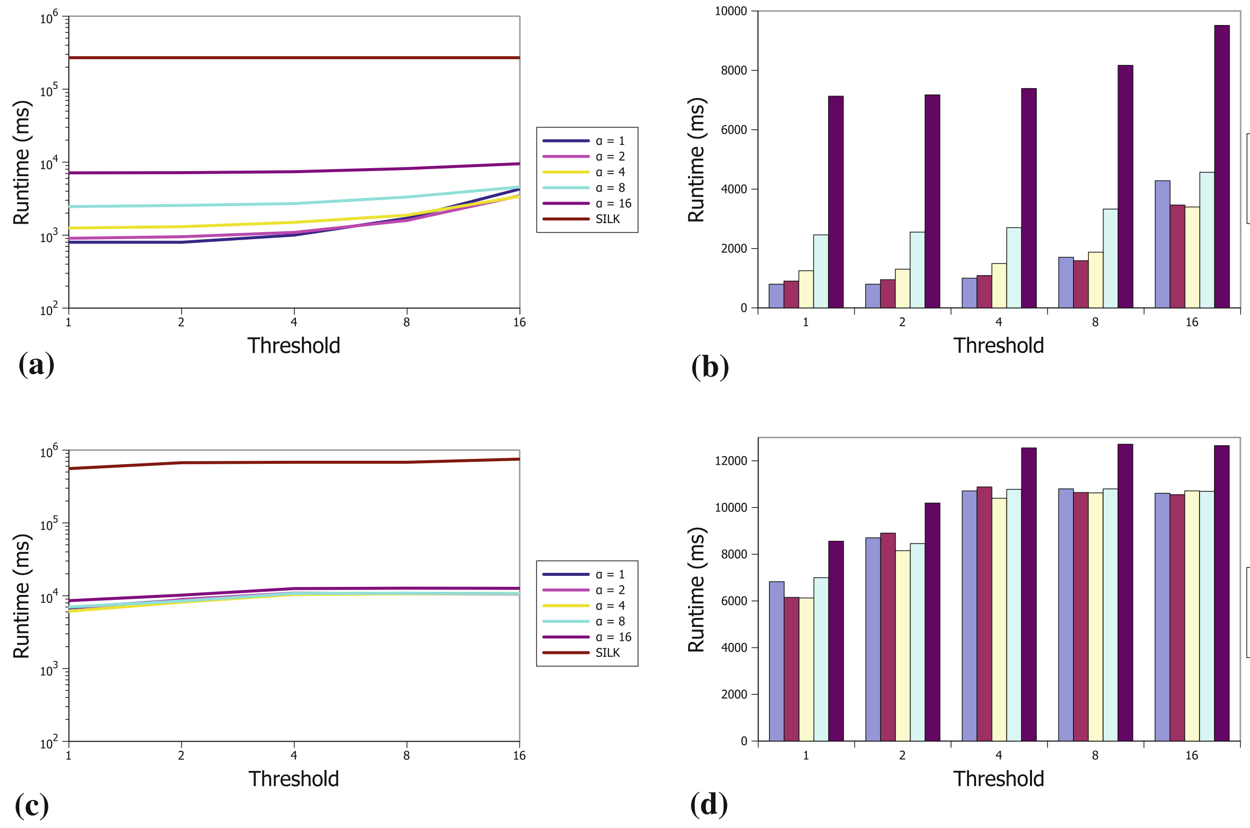


Fig. 13 Results of the two-dimensional experiments (Vacations and Actors). **a** Comparison with SILK in Vacations experiment. **b** Comparison for different α-values in Vacations experiment. **c** Comparison with SILK in the Actors experiment. **d** Comparison for different α-values in Actors experiment

Table 3 Summary of experimental setups for LIMES and SILK

Experiment	$\|S\|$	$\|T\|$	Dims	Complexity	Thresholds
Villages	26,717	103,175	2	5.5×10^9	τ_s, θ_p
Cities	36,877	39,800	3	4.4×10^9	τ_s, θ_p
Geo-Locations	50,031	74,458	4	14.9×10^9	$\tau_s, \theta_p, \theta_l$

Dims stands for dimensions

of the runtimes. In all setups, we use the trigrams similarity metric for strings and the Euclidean distance for numeric values.

Our results (see Fig. 15) confirm that we outperform SILK by orders of magnitude in all setups. Note that the runtimes reported for SILK are the lower bound of the actual runtime of the system as the indexing and data storage phase are merged. For example, gathering the data from the SPARQL endpoints required ca. 319 s for the Villages experiment. Once these values are subtracted from SILK's total runtime (e.g., 3,715 s for the Villages(strict) experiment), it becomes clear that the indexing phase required more than 70 % of SILK's total runtime for this large link discovery task, making LIMES more that 20 times faster than SILK in the worst case. In the best case, LIMES outperforms SILK by more than 4.5 orders of magnitude even when the indexing time of SILK was not taken into consideration. Especially in this experiment, our results suggest that the accurate approximation of HYPPO enabled LIMES to discard a significant number of unnecessary comparisons that could be discarded by an approach that uses a rougher approximation. The generation of overlapping blocks in combination with the number of dimensions of the experiments as well as the data distribution seem to account for the high runtimes requires by SILK. In contrast, the data distribution in the Geolocations experiment led to less blocks being generated and thus to better runtimes for SILK.

We compared the runtimes of LIMES for different values of α as shown in Fig. 16. Our results show that our assumption

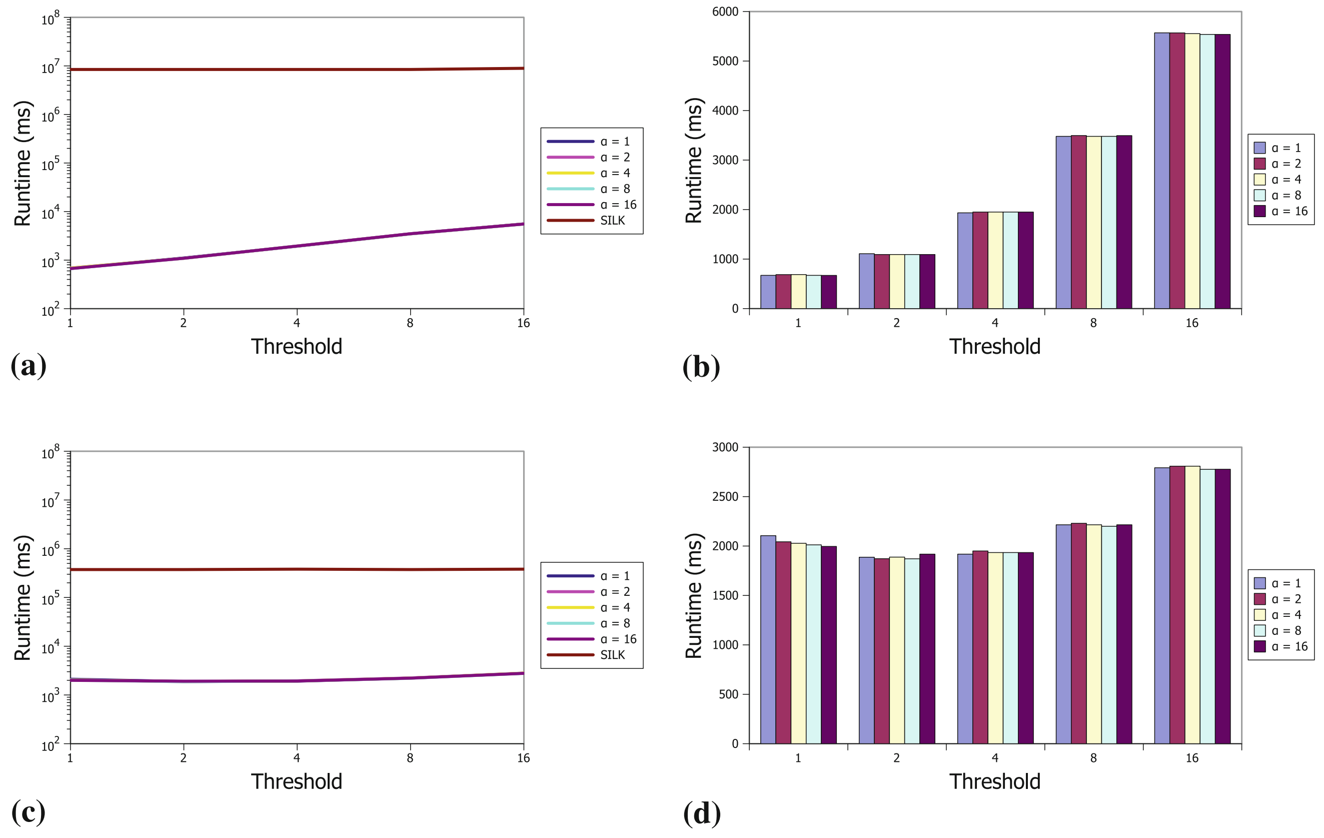
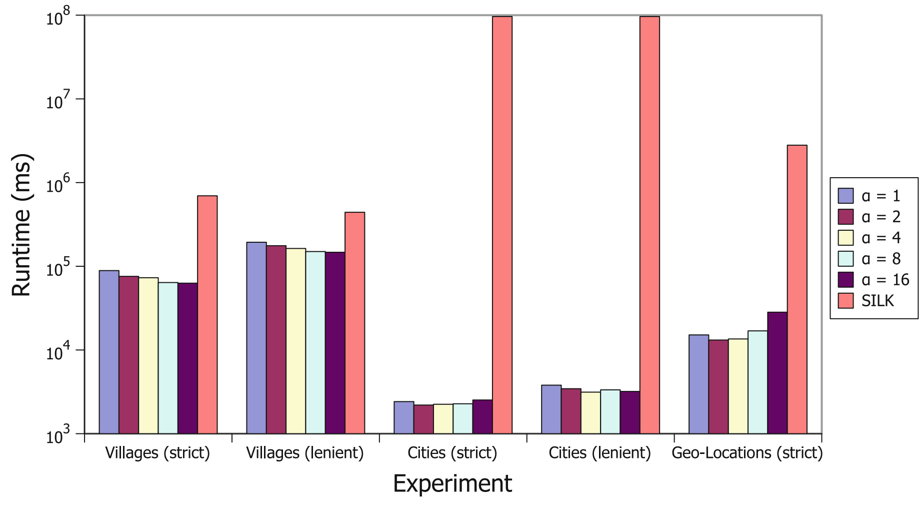


Fig. 14 Results of the three-dimensional experiments (Series and Hydrology). **a** Comparison with SILK in the Series experiment. **b** Comparison for different α-values in the Series experiment. **c** Comparison with SILK in the Hydrology experiment. **d** Comparison for different α-values in the Hydrology experiment

Fig. 15 Comparison of the runtime of LIMES and SILK on large-scale link discovery tasks



on the relation between α and runtimes is accurate as finding the right value for α can reduce the total runtime of the algorithm by approximately 40 % (Geo-Locations, $\alpha = 4$). In these experiments, setting α to 4 also led to an improved performance.

8 Discussion and Future Work

In this paper, we presented and evaluated a novel hybrid approach to LD. We first presented a series of requirements to LD frameworks. Based on these requirements,

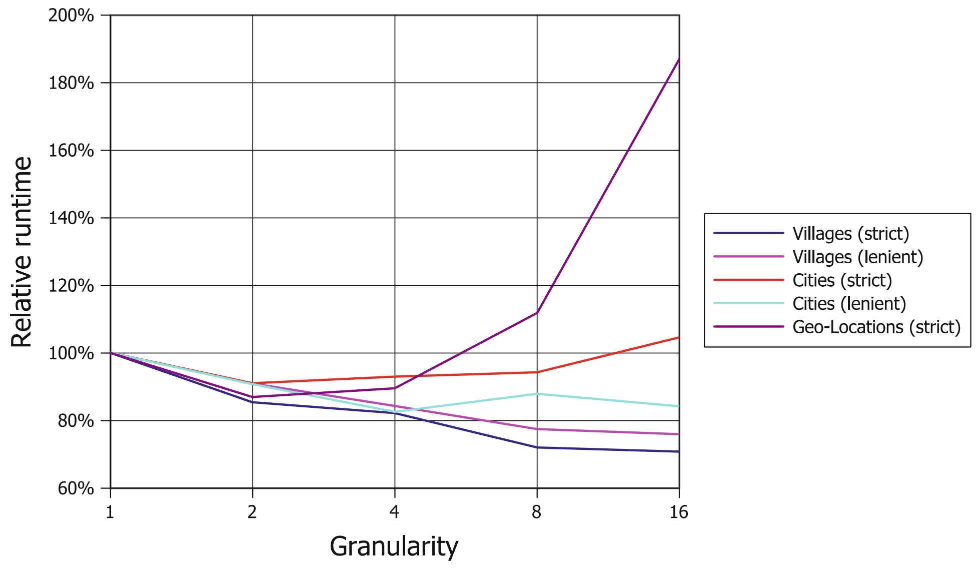


Fig. 16 Runtimes of LIMES relatively to the runtime for $\alpha = 1$

we specified the characteristics of such frameworks. We then presented original insights for converting complex link specifications into simple link specifications. Based on these conversions, we inferred that efficient means for processing simple link specifications are the key for time-efficient linking. We then presented the key time-efficient approaches implemented in LIMES and showed how these approaches can be combined for time-efficient linking. A thorough evaluation of our framework in nine experiments showed that we outperform SILK by up to 4.5 orders of magnitude while not losing a single link.

One of the central innovations of this paper is the HYpersphere aPPrOximation algorithm, HYPPO. Although it was defined for numeric values, HYPPO can be easily generalized to the efficient computation of pairs of entities that are totally ordered, i.e., to all sets of entities $e = (e_1, \dots, e_n) \in E$ such that a real function f_i exists, which preserves the order $\succ$ on the ith dimension of E, ergo $\forall e, e' \in E : e_i \succ e'_i \rightarrow f(e_i) > f(e'_i)$. Yet, it is important to notice that such a function can be complex and thus lead to overheads that may nullify the time gain of HYPPO. In future work, we will aim to find such functions for different data types. In addition, we will aim to formulate an approach for determining the best value of α for any given link specification. The new version of LIMES promises to be a stepping stone for the creation of a multitude of novel semantic applications, as it is time-efficient enough to make complex interactive scenarios for link discovery possible even at large scale [18].

Acknowledgments This work was supported by the Eurostars Project E!4604 SCMS and a research fellowship grant of the Research Unit Media Convergence of the University of Mainz.

References

1. Auer S, Lehmann J, Ngonga Ngomo A-C (2011) Introduction to linked data and its lifecycle on the web. In: Reasoning web, pp 1–75
2. Bayardo RJ, Ma Y, Srikant R (2007) Scaling up all pairs similarity search. In: WWW, pp 131–140
3. Ben-David D, Domany T, Tarem A (2010) Enterprise data classification using semantic web technologies. In: ISWC
4. Bleiholder J, Naumann F (2008) Data fusion. ACM Comput Surv 41(1):1–41
5. Christen P (2012) A survey of indexing techniques for scalable record linkage and deduplication. IEEE Trans Knowl Data Eng 24(9):1537–1555
6. Cudré-Mauroux P, Haghani P, Jost M, Aberer K, de Meer H (2009) idmesh: graph-based disambiguation of linked data. In: WWW, pp 591–600
7. Elmagarmid AK, Ipeirotis PG, Verykios VS (2007) Duplicate record detection: a survey. IEEE Trans Knowl Data Eng 19:1–16
8. Gale D, Shapley LS (1962) College admissions and the stability of marriage. Am Math Mon 69(1):9–15
9. Glaser H, Millard IC, Sung W-K, Lee S, Kim P, You B-J (2009) Research on linked data and co-reference resolution. University of Southampton, Technical Report
10. Hogan A, Polleres A, Umbrich J, Zimmermann A (2010) Some entities are more equal than others: statistical methods to consolidate linked data. In: Workshop on new forms of reasoning for the semantic web: scalable and dynamic (NeFoRS2010)
11. Isele R, Jentzsch A, Bizer C (2011) Efficient multidimensional blocking for link discovery without losing recall. In: WebDB
12. Köpcke H, Thor A, Rahm E (2009) Comparative evaluation of entity resolution approaches with fever. Proc VLDB Endow 2(2):1574–1577
13. Lehmann J, Furche T, Grasso G, Ngonga Ngomo A-C, Schallhart C, Sellers A, Unger C, Bühmann L, Gerber D, Höffner K, Liu D, Auer S (2012) Deqa: deep web extraction for question answering. In: Proceedings of ISWC, (to appear)
14. Lopez V, Uren V, Sabou MR, Motta E (2009) Cross ontology query answering on the semantic web: an initial evaluation. In: K-CAP '09: proceedings of the fifth international conference on knowledge capture, New York, NY, USA. ACM, pp 17–24

15. Manlove D, Irving R, Iwama K, Miyazaki S, Morita Y (2002) Hard variants of stable marriage. Theor Comput Sci 276(1–2):261–279
16. Ngonga Ngomo A-C (2011) A time-efficient hybrid approach to link discovery. In: Sixth international workshop on ontology matching at ISWC
17. Ngonga Ngomo A-C, Auer S (2011) Limes: a time-efficient approach for large-scale link discovery on the web of data. In: Proceedings of the international joint conference on artificial intelligence
18. Ngonga Ngomo A-C, Lehmann J, Auer S, Höffner K (2011) RAVEN: active learning of link specifications. In: Proceedings of the sixth international ontology matching workshop
19. Ngonga Ngomo A-C, Lyko K (2012) Eagle: efficient active learning of link specifications using genetic programming. In: Proceedings of ESWC
20. Nikolov A, D'Aquin M, Motta E (2012) Unsupervised learning of data linking configuration. In: Proceedings of ESWC
21. Nikolov A, Uren VS, Motta E, De Roeck AN (2009) Overcoming schema heterogeneity between linked semantic repositories to improve coreference resolution. In: ASWC, pp 332–346
22. Papadakis G, Ioannou E, Niedere C, Palpanasz T, Nejdl W (2011) Eliminating the redundancy in blocking-based entity resolution methods. In: JCDL
23. Raimond Y, Sutton C, Sandler M (2008) Automatic interlinking of music datasets on the semantic web. In: Proceedings of the 1st workshop about linked data on the web
24. Scharffe F, Liu Y, Zhou C (2009) Rdf-ai: an architecture for rdf datasets matching, fusion and interlink. In: Proceedings of IJCAI 2009 workshop on identity, reference, and knowledge representation (IR-KR), Pasadena (CA US)
25. Sleeman J, Finin T (2010) Computing foaf co-reference relations with rules and machine learning. In: Proceedings of the third international workshop on social data on the web
26. Urbani J, Kotoulas S, Maassen J, van Harmelen F, Bal H (2010) Owl reasoning with webpie: calculating the closure of 100 billion triples. In: Proceedings of the ESWC 2010
27. Volz J, Bizer C, Gaedke M, Kobilarov G (2009) Discovering and maintaining links on the web of data. In: ISWC, pp 650–665
28. Wang J, Li G, Feng J (2010) Trie-join: efficient trie-based string similarity joins with edit-distance constraints. PVLDB 3(1):1219–1230
29. Winkler W (2006) Overview of record linkage and current research directions. Technical Report, Bureau of the Census, Research Report Series
30. 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
31. Xiao C, Wang W, Lin X, Yu JX (2008) Efficient similarity joins for near duplicate detection. In: WWW, pp 131–140