A Time-Efficient Hybrid Approach to Link Discovery

Axel-Cyrille Ngonga Ngomo[1]

Department of Computer Science
University of Leipzig
Johannisgasse 26, 04103 Leipzig
`ngonga@informatik.uni-leipzig.de`,
WWW home page: `http://bis.uni-leipzig.de/AxelNgonga`

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 specifications based on a single property can be very 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 novel hybrid approach to link discovery that combines two very fast algorithms. Both 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 three experiments that our approach outperforms SILK by more than six orders of magnitude while abiding to the restriction of not losing any link.

Keywords: Linked Data, Link Discovery, Algorithms, Constraints

1 Introduction

The Linked Data Web has evolved from 12 knowledge bases in May 2007 to 203 knowledge bases in September 2010, i.e., in less than four years [6]. While the number of RDF triples available in the Linked Data Web has now surpassed 27 billion, less than 3% of these triples are links between knowledge bases [10]. Yet, links between knowledge bases play a key role in important tasks such as cross-ontology question answering [9], large-scale inferences [15] and data integration [2]. 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 [16, 10]. These frameworks rely on *link specifications*, which explicate conditions for computing new links between entities in knowledge bases. Due to the mere size of the Web of Data, detecting links even when using trivial specifications can be very 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. In this paper, we present a novel lossless hybrid approach to LD. Our approach is based on original insights on the distribution of property domain and ranges on the Web of Data. Based on these insights, we infer the requirements to efficient LD frameworks. We then use these requirements to specify the time-efficient approaches that underlie our framework, LIMES version 0.5[1]. We show that our framework outperforms state-of-the-art frameworks by several orders of magnitude with respect to runtime without losing links.

The contributions of this paper are as follows:

1. We present a formal grammar for link specifications that encompasses the functionality of state-of-the-art frameworks for LD.
2. Based on this grammar, we present a very 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+ [18] algorithm to be used for processing complex link specifications.
4. We specify and evaluate the HYpersphere aPPrOximation algorithm HYPPO, a fully novel LD approach designed to operate on numeric values.
5. We evaluate our approach against SILK [7] within three experiments and show that we outperform it by up to six orders of magnitude with respect to runtime while abiding to the constraint of not losing links.

The rest of this paper is structured as follows: In Section 2, we give a brief overview of related work on LD and related research fields. Section 3 presents the preliminaries to our work. These preliminaries are the basis for Section 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 Section 5. In section 6, we evaluate our approaches in three different large-scale experiments and show that we outperform the state-of-the-art approach SILK. After a discussion of our findings, we present our future work and conclude.

2 Related Work

Current frameworks for LD on the Web of Data can be subdivided into two categories: *domain-specific* and *universal* frameworks [10]. Domain-specific LD frameworks aim to discover links between knowledge bases from a particular domain. For example, the RKB knowledge base (RKB-CRS) [5] uses Universal Resource Identifier (URI) lists to compute links between universities and conferences. Another domain-specific tool is GNAT [12], which discovers links between

[1] LIMES stands for Link Discovery Framework for Metric Spaces. An online demo of the framework can be found at http://limes.sf.net

music data sets by using audio fingerprinting. Further simple or domain-specific approaches can be found in [14, 11].

Universal LD frameworks are designed to carry out mapping tasks independent from the domain of the source and target knowledge bases. For example, RDF-AI [13] implements a five-step approach that comprises the preprocessing, matching, fusion, interlinking and post-processing of data sets. SILK [7] (Version 2.3) implements a time-efficient and lossless approach that maps complex configurations to a multidimensional metric space. A blocking approach is then used in the metric space to reduce the number of comparisons by generating overlapping blocks. The original LIMES approach [10] presupposes that the datasets to link are in a metric space. It then uses the triangle inequality to portion the metric space so as to compute pessimistic approximations of distances. Based on these approximations, it can discard a large number of computations without losing links.

Although LD is closely related with record linkage [17, 4] and deduplication [3], it is important to notice that LD goes beyond these two tasks as LD aims to provide the means to link entities via arbitrary relations. 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 [8]. In addition, very time-efficient approaches have been proposed to compute string similarities for record linkage, including AllPairs [1], PPJoin and PPJoin+ [18]. However, these approaches alone cannot deal with the diversity of property values found on the Web of Data as they cannot deal with numeric values. In addition, most time-efficient string matching algorithms can only deal with simple link specifications, which are mostly insufficient when computing links between large knowledge bases.

The novel version of the LIMES framework goes beyond the state of the art (including previous versions of LIMES [10]) by integrating PPJoin+ and extending this algorithm so as to enable it to deal with complex configurations. In addition, LIMES0.5 integrates the fully novel HYPPO algorithm, 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 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 τ. 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 [10, 16], which translates into the following formal specification.

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 $\tau \in [0, 1]$, the goal of LD is to compute the set of pairs of instances $(s, t) \in S \times T$ such that $\sigma(s, t) \geq \tau$.*

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 $\delta(s, t) \leq \theta$.*

Note that a normed similarity function σ can always be derived from a distance function δ by setting $\sigma(x, y) = (1 + \delta(x, y))^{-1}$. Furthermore, the distance threshold θ can be transformed into a similarity threshold τ by setting $\tau = (1 + \theta)^{-1}$. 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 over several datatypes (numeric, strings, ...) 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. Consequently, linking on the Data Web demands frameworks that support complex link specifications.

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 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, $\equiv$ to denote the equivalence of two specifications and $A \sqsubseteq B$ to denote that the set of links that results from specification A is a subset of the set of links that results from specification B.

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. A similarity measure m is either an atomic similarity measure $atomicMeasure$ or the combination of two similarity measures via operators OP 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. $m \rightsquigarrow OP(m_1, m_2)$

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 $\sigma(s, t) \geq \tau$. A link specification $spec(m, \tau)$ is either an atomic link specification or the combination of two link specifications via operations 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). Thus, the following grammar for specifications holds :

1. $spec(m, \theta) \rightsquigarrow atomicSpec(m, \theta)$
2. $spec(m, \theta) \rightsquigarrow AND(spec(m_1, \theta_1), spec(m_2, \theta_2))$
3. $spec(m, \theta) \rightsquigarrow OR(spec(m_1, \theta_1), spec(m_2, \theta_2))$
4. $spec(m, \theta) \rightsquigarrow XOR(spec(m_1, \theta_1), spec(m_2, \theta_2))$
5. $spec(m, \theta) \rightsquigarrow DIFF(spec(m_1, \theta_1), spec(m_2, \theta_2))$

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

Note that while we can derive equivalent conditions on a smaller number of dimensions for the first two operations, 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 premise by using *filters*. Given these transformations, we can reduce all complex specifications that abide by our grammar to a sequence of set and filter operations on the results of atomic measures. Consequently, we can apply very 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. In the following, we present the approaches used by our framework on strings and numerical values.

5 Processing Simple Configurations

Our framework implements a hybrid approach to LD. The first approach implemented in our framework deals exclusively with strings by harnessing the near-duplicate detection algorithm PPJoin+ [18]. Instead of mapping strings to a vector space, PPJoin+ uses a combination of three main insights to implement a very time-efficient string comparison approach. First, it uses 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 PPJoin+. Finally, the algorithm makes 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+ can discard a large number of non-matches. The integration of the PPJoin+ algorithm 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+ is that it can only operate on one dimension [8]. However, by applying the transformations of configurations specified above, we make PPJoin+ 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*.

The HYPPO algorithm addresses the problem of efficiently mapping instance pairs $(s, t) \in S \times T$ described by 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 $\omega = (\omega_1, ..., \omega_n)$ and $x = (x_1, ..., x_n)$ 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, distance metrics can be decomposed into the combination of functions $\phi_{i,i \in \{1...n\}}$ which operate on exactly one dimension of $\Omega : \delta = f(\phi_1, ..., \phi_n)$. For example, for Minkowsky 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 Minkowsky 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 $\delta(x, \omega) = \sqrt{(x - \omega)\Gamma(x - \omega)^T}$, 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:

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

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 $V(\omega, \theta) = \{x \in \Omega : \forall i \in \{1...n\}, \phi_i(x_i, \omega_i) \leq \theta\}$ 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 tiles 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 [8]. In addition, HYPPO is guaranteed not to lose any link, as H is completely enclosed in V, while most blocking techniques are not lossless.

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 $\forall i \in \{1...n\}, c_i\Delta \leq \omega_i < (c_i + 1)\Delta$, $(c_1, ..., c_n) \in \mathbb{N}^n$. We call the vector $(c_1, ..., c_n)$ the coordinates of the cube C. Each point $\omega \in \Omega$ lies in the cube $C(\omega)$ with coordinates $(\lfloor \omega_i/\Delta \rfloor)_{i=1...n}$. 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 $\forall i \in \{1...n\} : |c_i - c(\omega)_i| \leq \alpha$. Figure 1 shows examples of space tilings for different values of α.

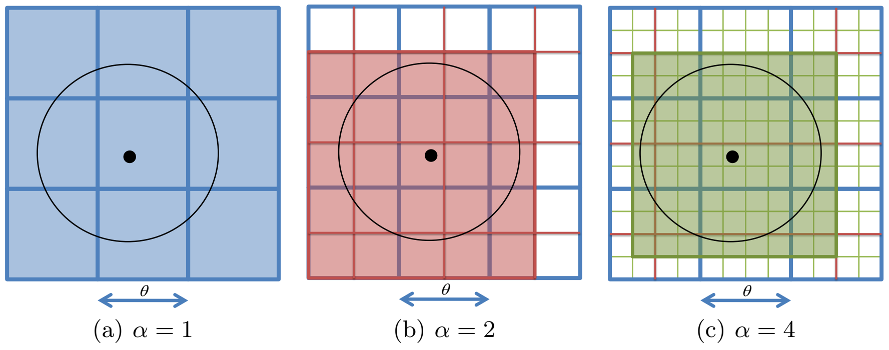

Fig. 1. Space tiling for different values of α. The colored squares show the set of elements that must be compared with the instance located at the black dot. The points within the circle lie within the distance θ of the black dot.

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 $V_{\mathfrak{C}}(\alpha, \theta) = ((2\alpha + 1)\Delta)^n = \left(\frac{2\alpha+1}{\alpha}\theta\right)^n$ that approximates $H(\omega, \theta)$. The volume $V_H(\theta)$ of $H(\omega, \theta)$ is given by $\frac{S_n \theta^n}{n}$, where S_n is the volume of a unit sphere in n dimensions, i.e., 2 for $n = 1$, π for $n = 2$, $\frac{4\pi}{3}$ for $n = 3$ and so on. The approximation ratio

$$\frac{V_{\mathfrak{C}}(\alpha, \theta)}{V_H(\theta)} = \frac{n}{S_n} \left(\frac{2\alpha + 1}{\alpha}\right)^n, \tag{2}$$

permits to determine the accuracy of HYPPO's approximation as shown in Figure 2 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.

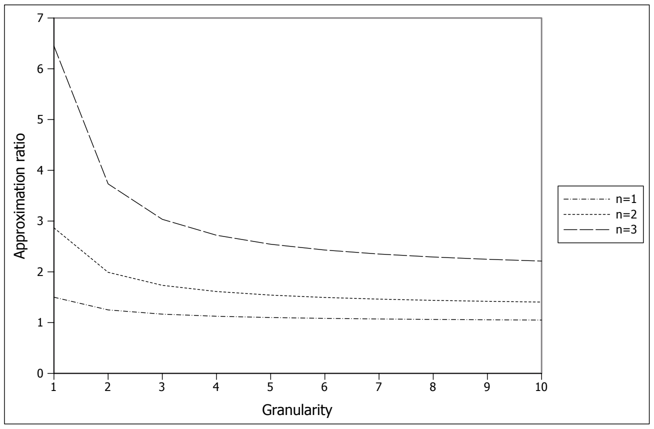

Fig. 2. Approximation ratio for $n \in \{1, 2, 3\}$. The x-axis shows values of α while the y-axis shows the approximation ratios.

6 Evaluation

We compared our framework (i.e., LIMES Version 0.5) with SILK version 2.3. in three large-scale experiments of different complexity based on geographic data. 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. We ran all experiments on the same computer running a Windows 7 Enterprise 64-bit installation on a 2.8GHz i7 processor with 8GB RAM. The JVM was

Algorithm 1 Current implementation of HYPPO

```
Require: $S$, $T$, $\theta$, $\delta$, $\alpha$ as defined above
  Mapping $M := \emptyset$
  $\Delta = \theta/\alpha$
  for $\omega \in S \cup T$ do
    $\mathrm{C}(\lfloor\omega_1/\Delta\rfloor, ..., \lfloor\omega_n/\Delta\rfloor) := \mathrm{C}(\lfloor\omega_1/\Delta\rfloor, ..., \lfloor\omega_n/\Delta\rfloor) \cup \{\omega\}$
  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 $M$
```

allocated 7.4GB RAM. For each tool we measured exclusively the time needed for computing the links. All experiments were carried out 5 times except when stated otherwise. In all cases, we report best runtimes. Experiments marked with an asterisk would have lasted longer than 48 hours when using SILK and were not computed completely. Instead, SILK's runtime was approximated by extrapolating the time needed by the software to compute 0.1% of the links.

We chose to use geographic datasets because they are large and allow the use of several attributes for linking. 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 by using 4 combinations of criteria for comparing entities: their longitude (`wgs84:long`), latitude (`wgs84:lat`), preferred names and names.

Experiment	$\|S\|$	$\|T\|$	Dims	Complexity	Source	Target	Thresholds
Villages*	26717	103175	2	3.8×10^9	DBpedia	LGD	τ_s, θ_p
Cities*	36877	39800	3	1.5×10^9	Geonames	DBpedia	τ_s, θ_p
Geo-Locations*	50031	74458	4	3.7×10^9	LGD	GeoNames	τ_s, θ_p, θ_l

Table 1. Summary of experimental setups for LIMES and SILK. Dims stands for dimensions.

The setup of the experiments is summarized in Table 1. 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 7.4G of RAM allocated to both tools and led to swapping, thus falsifying the evaluation of the runtimes. In all setups, we use the trigrams similarity metrics for strings and the euclidean distance for numeric values.

Our results (see Figure 3) confirm that we outperform SILK by several orders of magnitude in all setups. In the Cities experiment, we are more than 6 orders of magnitude faster than SILK. We compared the runtimes of LIMES for different values of α as shown in Figure 4. Our results show that our assumption 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% (see Geo-Locations, $\alpha = 4$). In general, setting α to values between 2 and 4 leads to an improved performance in all experiments.

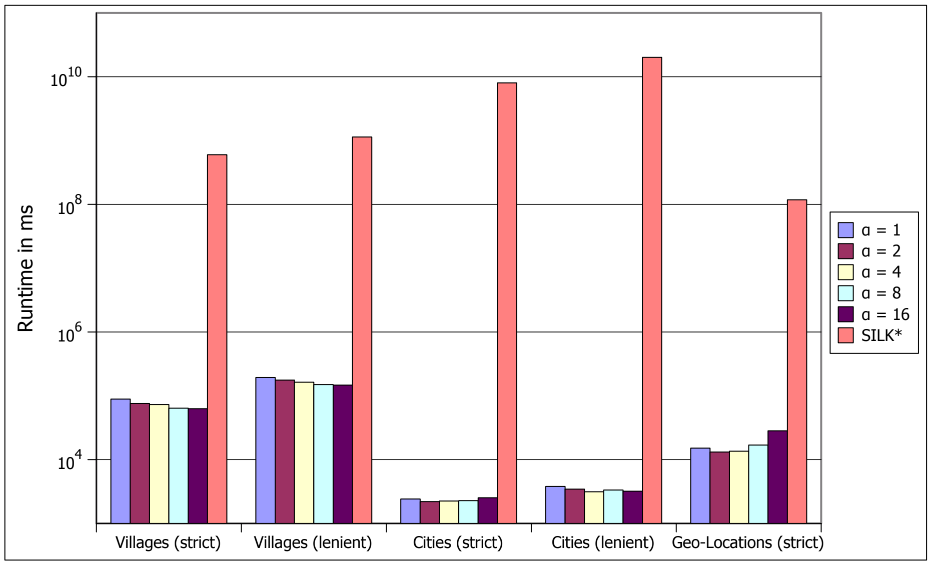

Fig. 3. Comparison of the runtime of LIMES and SILK on large-scale link discovery tasks.

7 Discussion and Future Work

In this paper, we introduced and evaluated a novel hybrid approach to LD. We presented original insights on the conversion of complex link specifications

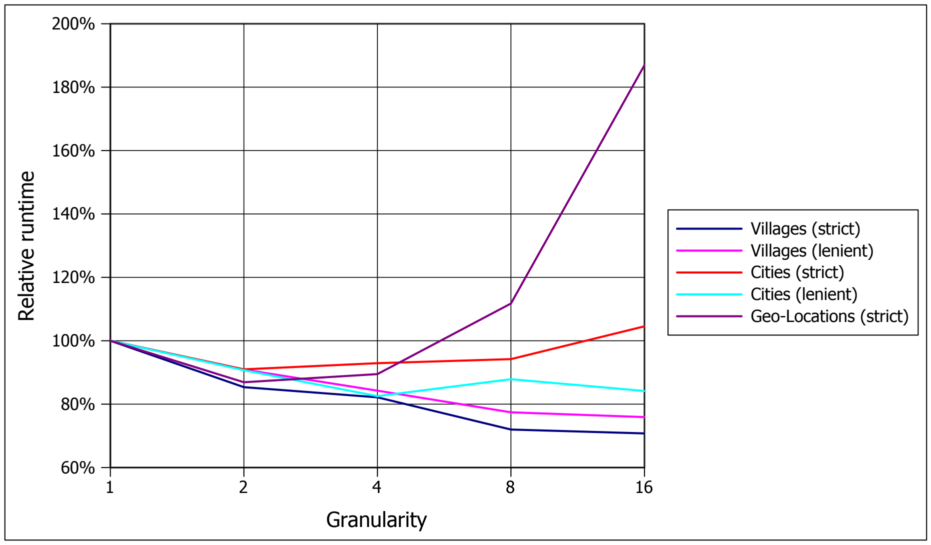

Fig. 4. Runtimes of LIMES relatively to the runtime for $\alpha = 1$.

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 two time-efficient approaches implemented in LIMES0.5 and showed how these approaches can be combined for time-efficient linking. A thorough evaluation of our framework in three large-scale experiments showed that we outperform SILK by more than 6 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 the similarity of pairs of entities that are totally ordered, i.e., to all sets of entities $e = (e_1, ..., e_n) \in E$ 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 very 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.

Acknowledgement

This work was supported by the Eurostars grant SCMS E!4604 and a fellowship grant of the Mainz University.

References

1. Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. Scaling up all pairs similarity search. In *WWW*, pages 131–140, 2007.
2. David Ben-David, Tamar Domany, and Abigail Tarem. Enterprise data classification using semantic web technologies. In *ISWC*, 2010.
3. Jens Bleiholder and Felix Naumann. Data fusion. *ACM Comput. Surv.*, 41(1):1–41, 2008.
4. Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19:1–16, 2007.
5. Hugh Glaser, Ian C. Millard, Won-Kyung Sung, Seungwoo Lee, Pyung Kim, and Beom-Jong You. Research on linked data and co-reference resolution. Technical report, University of Southampton, 2009.
6. Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, 2011.
7. R. Isele, A. Jentzsch, and C. Bizer. Efficient Multidimensional Blocking for Link Discovery without losing Recall. June 2011.
8. Hanna Köpcke, Andreas Thor, and Erhard Rahm. Comparative evaluation of entity resolution approaches with fever. *Proc. VLDB Endow.*, 2(2):1574–1577, 2009.
9. Vanessa Lopez, Victoria Uren, Marta Reka Sabou, and Enrico Motta. Cross ontology query answering on the semantic web: an initial evaluation. In *K-CAP '09: Proceedings of the fifth international conference on Knowledge capture*, pages 17–24, New York, NY, USA, 2009. ACM.
10. Axel-Cyrille Ngonga Ngomo and Sören Auer. A time-efficient approach for large-scale link discovery on the web of data. In *IJCAI*, 2011.
11. George Papadakis, Ekaterini Ioannou, Claudia Niedere, Themis Palpanasz, and Wolfgang Nejdl. Eliminating the redundancy in blocking-based entity resolution methods. In *JCDL*, 2011.
12. Yves Raimond, Christopher Sutton, and Mark Sandler. Automatic interlinking of music datasets on the semantic web. In *Proceedings of the 1st Workshop about Linked Data on the Web*, 2008.
13. Franois Scharffe, Yanbin Liu, and Chuguang Zhou. Rdf-ai: an architecture for rdf datasets matching, fusion and interlink. In *Proc. IJCAI 2009 workshop on Identity, reference, and knowledge representation (IR-KR), Pasadena (CA US)*, 2009.
14. Jennifer Sleeman and Tim Finin. Computing foaf co-reference relations with rules and machine learning. In *Proceedings of the Third International Workshop on Social Data on the Web*, 2010.
15. Jacopo Urbani, Spyros Kotoulas, Jason Maassen, Frank van Harmelen, and Henri Bal. Owl reasoning with webpie: calculating the closure of 100 billion triples. In *Proceedings of the ESWC 2010*, 2010.
16. Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Discovering and maintaining links on the web of data. In *ISWC*, pages 650–665, 2009.
17. William Winkler. Overview of record linkage and current research directions. Technical report, Bureau of the Census - Research Report Series, 2006.
18. Chuan Xiao, Wei Wang, Xuemin Lin, and Jeffrey X. Yu. Efficient similarity joins for near duplicate detection. In *WWW*, pages 131–140, 2008.