On the Efficient Execution of Bounded Jaro-Winkler Distances

Kevin Dreßler [a], Axel-Cyrille Ngonga Ngomo [a]
[a] *AKSW Research Group, University of Leipzig, Germany*

Abstract. Over the last years, time-efficient approaches for the discovery of links between knowledge bases have been regarded as a key requirement towards implementing the idea of a Data Web. Thus, efficient and effective measures for comparing the labels of resources are central to facilitate the discovery of links between datasets on the Web of Data as well as their integration and fusion. We present a novel time-efficient implementation of filters that allow for the efficient execution of bounded Jaro-Winkler measures. We evaluate our approach on several datasets derived from DBpedia 3.9 and LinkedGeoData and containing up to 10^6 strings and show that it scales linearly with the size of the data for large thresholds. Moreover, we also show that our approach can be easily implemented in parallel. We also evaluate our approach against SILK and show that we outperform it even on small datasets.

Keywords: Link Discovery, String similiarity measure

1. Introduction

The Linked Open Data Cloud (LOD Cloud) has developed to a compendium of more than 2000 datasets over the last few years.[1] For example, data sets pertaining to more than 14 million persons have already been made available on the Linked Data Web.[2] While this number is impressive on its own, it is well known that the population of the planet has surpassed 7 billion people. Hence, the Web of Data contains information on less than 1% of the overall population of the planet (counting both the living and the dead). The output of open-government movements[3], scientific conferences[4], health data[5] and similar endeavours yet promises to make massive amounts of data pertaining to persons available in the near future. Dealing with this upcoming increase of the number of resources on the Web of data requires providing means to integrate these datasets with the aim to facilitate statistical analysis, data mining, personalization, etc. However, while the number of datasets on the Linked Data Web grows drastically, the number of links between datasets still stagnates.[6] Addressing this lack of links requires solving two main problems: the quadratic time complexity of link discovery (efficiency) and the automatic support of the detection of link specifications (effectiveness). In this paper, we address the efficiency of the execution of bounded Jaro-Winkler measures,[7] which are known to be effective when comparing person names [10]. To this end, we derive equations that allow discarding a large number of computations while executing bounded Jaro-Winkler comparisons with high thresholds.

The contributions of this paper are as follows:

1. We derive length- and range-based filters that allow reducing the Frequency t that are compared with a string s .

[1] See `http://stats.lod2.eu` for an overview of the current state of the Cloud. Last access: July 11th, 2014.

[2] Data collected from `http://stats.lod2.eu`. Last access: July 11th, 2014.

[3] See for example `http://data.gov.uk/`.

[4] See for example `http://data.semanticweb.org/`

[5] `http://aksw.org/Projects/GHO`

[6] `http://linklion.org`

[7] We use bounded measures in the same sense as [15], i.e., to mean that we are only interested in pairs of strings whose similarity is greater than or equal to a given lower bound.

2. We present a character-based filter that allows detecting whether two strings s and t share enough resemblance to be similar according to the Jaro-Winkler measure.
3. We evaluate our approach w.r.t. to its runtime and its scalability with several threshold settings and dataset sizes.

The rest of this paper is structured as follows: In Section 2, we present the problem we tackled as well as the formal notation necessary to understand this work. In the subsequent Section 3, we present the three approaches we developed to reduce the runtime of bounded Jaro-Winkler computations. We then evaluate our approach in Section 4. Related work is presented in Section 5, where we focus on approaches that aim to improve the time-efficiency of link discovery. We conclude in Section 6. The approach presented herein is now an integral part of LIMES.[8] This paper is an extended version of [11].

2. Preliminaries

In the following, we present some of the symbols and terms used within this work.

2.1. Link Discovery

In this work, we use *link discovery* as a hypernym for deduplication, record linkage, entity resolution and similar terms used across literature. The formal specification of link discovery adopted herein is tantamount to the definition proposed in [18]: Given a set S of source resources, a set T of target resources and a relation R, our goal is to find the set $M \subseteq S \times T$ of pairs (s, t) such that $R(s, t)$. If R is `owl:sameAs`, then we are faced with a *deduplication task*. Given that the explicit computation of M is usually a very complex endeavour, M is most commonly approximated by a set $M' = \{(s, t, \delta(s, t)) \in S \times T \times \mathbb{R}^+ : \sigma(s, t) \geq \theta\}$, where σ is a (potentially complex) similarity function and $\theta \in [0, 1]$ is a similarity threshold. Given that this problem is in $O(n^2)$, using naïve algorithms to compare large S and T is most commonly impracticable. Thus, time-efficient approaches for the computation of bounded measures have been developed over the last years for measures such as the Levenshtein distance, Minkowski distances, trigrams and many more [17].

[8] `http://limes.sf.net`

In this paper, we thus study the following problem: Given a threshold $\theta \in [0, 1]$ and two sets of strings S and T, compute the set $M' = \{(s, t, \delta(s, t)) \in S \times T \times \mathbb{R}^+ : \sigma(s, t) \geq \theta\}$. Two categories of approaches can be considered to improve the runtime of measures: Lossy approaches return a subset M'' of M' which can be calculated efficiently but for which there are no guarantees that $M'' = M'$. Lossless approaches on the other hand ensure that their result set M'' is exactly the same as M'. In this paper, we present a lossless approach. To the best of our knowledge, only one other link discovery framework implements a lossless approach that has been designed to exploit the bound defined by the threshold θ to ensure a more efficient computation of the Jaro-Winkler distance, i.e., the SILK framework with the approach MultiBlock [9]. We thus compare our approach with SILK 2.6.0 in the evaluation section of this paper.

2.2. The Jaro-Winkler Similarity

Let Σ^* be the set of all possible strings over Σ. The Jaro measure $d_j : \Sigma^* \times \Sigma^* \to [0, 1]$ is a string similarity measure approach which was developed originally for name comparison in the U.S. Census. This measure takes into account the number of character matches m and the ratio of their transpositions t:

$$d_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s|} + \frac{m}{|t|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases} \quad (1)$$

Here two characters are considered to be a match if and only if (1) they are the same and (2) they are at most at a distance $w = \lfloor \frac{\max(|s|,|t|)}{2} \rfloor$ from each other. For example, for $s = "Spears"$ and $t = "Pears"$, the second s of s matches the s of t while the first s of s does not match the s of t.

The Jaro-Winkler measure [29] is an extension of the Jaro distance. This extension is based on Winkler's observation that typing errors occur most commonly in the middle or at the end of a word, but very rarely in the beginning. Hence, it is legitimate to put more emphasis on matching prefixes if the Jaro distance exceeds a certain "boost threshold" b_t, originally set to 0.7.

$$d_w = \begin{cases} d_j & \text{if } d_j < b_t \\ d_j + (\ell p\,(1 - d_j)) & \text{otherwise} \end{cases} \quad (2)$$

Here, ℓ denotes the length of the common prefix and p is a weighting factor. Winkler uses $p = 0.1$ and $\ell \leq 4$. Note that ℓp must not be greater than 1.

3. Improving the Runtime of Bounded Jaro-Winkler

The main principle behind reducing the runtime of the computation of measures is to reduce their reduction ratio. Here, we use a sequence of filters that allow discarding similarity computations while being sure that they would have led to a similarity score which would have been less than our threshold θ. To this end, we regard the problem as that of finding filters that return an upper bound estimation $\theta_e(s,t) \geq d_w(s,t)$ for some properties of the input strings that can be computed in constant time. For a given threshold θ, if $\theta_e(s,t) \leq \theta$, then we can safely ignore the input (s,t).

3.1. Length-based filters

In the following, we denoted the length of a string s with $|s|$. Our first filter is based on the insight that large length differences are a guarantee for poor similarity. For example, the strings $"a"$ and $"alpha"$ cannot have a Jaro-Winkler similarity of 1 by virtue of their length difference. We can formalize this idea as follows: Let s and t be strings with respective lengths $|s|$ and $|t|$. Without loss of generality, we will assume that $|s| \leq |t|$. Moreover, let m be the number of matches across s and t. Because $m \leq |s|$, we can substitute m with $|s|$ and gain the following upper bound estimation for $d_j(s,t)$:

$$\begin{aligned} d_j &= \frac{1}{3}\left(\frac{m}{|s|} + \frac{m}{|t|} + \frac{m-t}{m}\right) \\ &\leq \frac{1}{3}\left(1 + \frac{|s|}{|t|} + \frac{|s|-t}{|s|}\right) \end{aligned} \tag{3}$$

Now the lower bound for the number t of transpositions is 0. Thus, we obtain the following equation.

$$d_j \leq \frac{1}{3}\left(1 + \frac{|s|}{|t|} + 1\right) \leq \frac{2}{3} + \frac{|s|}{3|t|} \tag{4}$$

The application of this approximation on Winkler's extension is trivial:

$$\begin{aligned} d_w &= d_j + \ell \cdot p \cdot (1 - d_j) \\ &\leq \frac{2}{3} + \frac{|s|}{3|t|} + \ell \cdot p \cdot \left(\frac{1}{3} - \frac{|s|}{3|t|}\right) = \theta_e^{length} \end{aligned} \tag{5}$$

Consider the pair $s = "bike"$ and $t = "bicycle"$ and a threshold $\theta = 0.9$. Applying the estimation for Jaro we get $d_j \leq \frac{2}{3} - \frac{4}{3 \cdot 7} = 0.857$. This exceeeds the boost threshold, so we use equation 5 to compute $\theta_e^{length}(s,t) = 0.885$. Now we do not have to actually compute $d_w(s,t)$, since $\theta_e^{length}(s,t) < \theta$.

By using this approach we can decide in $O(1)$[9] if a given pairs score is greater than a given threshold, which saves us the much more expensive score computation for a big number of pairs, provided that the input strings sufficiently vary in length.

3.2. Filtering ranges by length

The approach described above can be reversed to limit the number of pairs that we are going to be iterating over. To this end, we can construct an $index : \mathbb{N} \rightarrow \Sigma^*$ which maps string lengths $l \in \mathbb{N}$ to all strings s with $|s| = l$. With the help of this index, we can now determine the set of strings t that should be compared with the subset $S(l)$ of S that only contains strings of length l. We go about using this insight by computing the upper and lower bound for the length of a string t that should be compared with a string s. This is basically equivalent to asking what is the minimum length difference $||s| - |t||$ so that $\theta \geq \theta_e^{length}(s,t)$ is satisfied. We transpose equation 5 to the following for our lower bound:

$$|s| \geq \left\lfloor 3|t|\frac{\theta - \ell p}{1 - \ell p} - 2|t| \right\rfloor = \rho_{min} \tag{6}$$

Analogously, we can derive the following upper bound:

$$|s| \leq \left\lceil \frac{|t|}{3\frac{\theta - \ell p}{1 - \ell p} - 2} \right\rceil = \rho_{max} \tag{7}$$

For example, consider a list of strings S with equally distributed string lengths $(4, 7, 11, 18)$ and $\theta = 0.9$ Using Equation 6 and Equation 7 we obtain Table 1. Taking into account the last column of the table, we will save a total of $\frac{3}{8}$ comparisons.
Note that for practical usage of this bounds, the threshold must be chosen greater than $\tau = \frac{2+\ell p}{3}$, because ρ_{max} has a pole at $\theta = \tau$ and ρ_{min}, ρ_{max} are of negative value for $\theta < \tau$.

[9]In most programming languages, especially Java (which we used for our implementation), the length of string is stored in a variable and can thus be accessed in constant time.

Table 1
Bounds for distinct string lengths ($\theta = 0.9$)

$\|s\|$	ρ_{min}	ρ_{max}	sizes in range
4	2	8	$(4, 7)$
7	3	14	$(4, 7, 11)$
11	5	22	$(7, 11, 18)$
18	9	36	$(11, 18)$

3.3. *Filtering by character frequency*

An even more fine-grained approach can be chosen to filter out computations. Let $e : \Sigma \times \Sigma^* \rightarrow \mathbb{N}$ be the function which returns the number of occurrences of a given character in a string. For the strings s and t, the number of maximum possible matches m_{max} can be expressed as

$$m_{max} = \sum\nolimits_{c \in s} min(e(s,c), e(t,c)) \geq m \tag{8}$$

Consequently, we can now substitute m for m_{max} in the Jaro distance computation:

$$\begin{aligned} d_j(s,t) &= \frac{1}{3}\left(\frac{m_{max}}{|s|} + \frac{m_{max}}{|t|} + \frac{m_{max} - t}{m_{max}}\right) \\ &\leq \frac{1}{3}\left(\frac{m_{max}}{|s|} + \frac{m_{max}}{|t|} + 1\right) = \theta_e^{char} \end{aligned} \tag{9}$$

We can thus derive that $d_j(s,t) \geq \theta$ iff

$$m_{max} \geq \frac{(3\theta - 1)\,|s||t|}{|s| + |t|}. \tag{10}$$

For instance, let $s =$"astronaut", $t =$"astrochimp". The retrieval of m_{max} ist shown in Table 2.

3.3.1. *Naïve Implementation*

The naive implementation of the character-based filter consists of implementation the e function for each string using a map. As shown by our evaluation, the character-based filter leads to a significant reduction of the number of comparisons (see Figure 6) by more than 2 orders of magnitude. However, the runtime improvement achieved using this implementation is not substantial. This is simply due to the lookup into maps being constant in time complexity but still a large amount of time. Instead of regarding strings as monolithic entities, we thus extended our implementation to be more fine-grained and used a trie as explained in the subsequent section.

Table 2
Calculation of m_{max}

c	$e(s,c)$	$e(t,c)$	$min(e(s,c), e(t,c))$	m_{max}
a	2	1	1	1
c	0	1	0	1
h	0	1	0	1
i	0	1	0	1
m	0	1	0	1
n	1	0	0	1
o	1	1	1	2
p	0	1	0	2
r	1	1	1	3
s	1	1	1	4
t	2	1	1	5
u	1	0	0	5

3.3.2. *Implementation with Tries*

To overcome the need to perform character index lookups for every pair of strings we use a trie pruning technique. Therefore, we call this approach the trie filter below. We name the dataset with the longest string $\mathcal{D}_l$ and the other $\mathcal{D}_s$ respectively. We define a function $\sigma : \Sigma^+ \rightarrow \Sigma^+_{ordered}$, which maps a word onto an ordered permutation of itself, e.g. $\sigma(hello) = ehllo$. Let $\mathcal{T}$ be our trie. All strings $s_i, s_j \in \mathcal{D}_l (i \neq j)$ with $\sigma(s_i) = \sigma(s_j)$ are added to buckets $\mathcal{B}_{\sigma(s_x)}$. All strings $t_x \in \mathcal{D}_s$ are inserted into buckets that are associated with nodes at path $\sigma(t_x)$ of $\mathcal{T}$, so strings with the same σ representation also reside in the same bucket. From the Jaro-Winkler version of the upper bound estimation given in Equation 9 we obtain the following equation:

$$\begin{aligned} m_{max} &\geq m_{needed}(s,t) \\ &= \left\lceil \left(\theta - \ell p - \frac{1 - \ell p}{3}\right) \cdot \frac{3|s||t|}{(|s| + |t|)\,(1 - \ell p)} \right\rceil \end{aligned} \tag{11}$$

Where m_{needed} is the minimal number of matches so that the filter will not apply, i.e. it is possible that $d_w(s,t) \geq \theta$. For every $\mathcal{B}_u$, we traverse $\mathcal{T}$ depth-first in pre-order. The tree traversal stack $\mathcal{S}$ initially contains tuples in terms of $\langle \mathcal{C}, \mathcal{V}, m \rangle$ for every child of the root node. Herein, $\mathcal{C}$ is called the character stack and initially contains u, $\mathcal{V}$ is a reference to the current node and m is the number of matches up to this node. Since every path v in $\mathcal{T}$ is an ordered character sequence we can compute the maximum possible matches m_{max} between u and the current traversal path in every iter-

Algorithm 1 Trie filter

```
1:  procedure TrieFilter($\mathcal{D}_l, \mathcal{D}_s, \theta$)
2:      $\mathcal{T} \leftarrow$ ConstructTrie($\mathcal{D}_s$)
3:      $\mathcal{B} \leftarrow$ ConstructContainerMap($\mathcal{D}_l$)
4:      for all $\mathcal{B}_u \in \mathcal{B}$ do
5:          TrieTraverse($\mathcal{T}, \mathcal{B}_u, u, \theta$)
6:      end for
7:  end procedure

8:  function $d_w^{max}(l_1, l_2, m)$
9:      $p \leftarrow 4$
10:     $\ell \leftarrow 0.1$
11:     return $\frac{1-\ell p}{3}\left(\frac{m}{l_1} + \frac{m}{l_2} + 1\right) + \ell p$
12: end function

13: procedure TrieTraverse($\mathcal{T}, \mathcal{B}_x, u, \theta$)
14:     $v_{min} \leftarrow$ shortest path length to leaf in $\mathcal{T}$
15:     $v_{max} \leftarrow$ longest path length to leaf in $\mathcal{T}$
16:     $\mathcal{S} \leftarrow$ new Stack()
17:     AddRootChilds($\mathcal{T}, \mathcal{S}, u$)
18:     while $\mathcal{S} \neq \emptyset$ do
19:         $\langle \mathcal{C}, \mathcal{V}, m \rangle \leftarrow \mathcal{S}$.pop()
20:         $m_{max} \leftarrow min(|\mathcal{C}|, v_{max} - |\mathcal{V}| + 1) + m$
                                ▷ $|\mathcal{V}|$ denotes the level of $\mathcal{V}$
21:         if $m_{max} \geq m_{needed}(|u|, v_{min})$ then
22:             if $\mathcal{V}.key() \leq \mathcal{C}.top()$ then
23:                 if $\mathcal{V}.key() = \mathcal{C}.top()$ then
24:                     $\mathcal{C}.pop()$
25:                     $m \leftarrow m + 1$
26:                     $\alpha_1 \leftarrow \mathcal{V}.hasBucket()$
27:                     $\alpha_2 \leftarrow d_w^{max}(|\mathcal{V}|, |u|, m) \geq \theta$
28:                     if $\alpha_1 \wedge \alpha_2$ then
29:                         $\mathcal{A} \leftarrow \mathcal{V}.bucket()$
30:                         $\mathcal{M} \leftarrow \mathcal{M} \cup \langle \mathcal{A}, \mathcal{B}_x \rangle$
31:                     end if
32:                 end if
33:                 for all $x \in \mathcal{V}.children()$ do
34:                     $\mathcal{S}.push(\langle \mathcal{C}, x, m \rangle)$
35:                 end for
36:             else
37:                 $\mathcal{C}.pop()$
38:                 $\mathcal{S}.push(\langle \mathcal{C}, \mathcal{V}, m \rangle)$
39:             end if
40:         end if
41:     end while
42: end procedure
```

ation. If $m_{max} < m_{needed}$ we skip the current node, effectively pruning its subtree from $\mathcal{T}$. If the topmost character of $\mathcal{C}$, called $\mathcal{C}.top()$, and the character of the current trie node v_c are of equal order, we increment m and add its children to $\mathcal{S}$.
Else, if $\mathcal{C}.top() > v_c$, we just add the children. Finally, when $\mathcal{C}.top() < v_c$, we just pop $\mathcal{C}$ and push the current tuple back onto $\mathcal{S}$. Match candidates are inserted into a set $\mathcal{M}$ if the current node has an associated bucket and $d_w^{max}(l_1, l_2, m) \geq \theta$. The detailed logic of the tree traversal is given in Algorithm 1.
For instance, let $\theta = 0.95$, $D_l = \{$`nines`$\}$ and $D_s = \{$`mices, nices, niche, niece, since`$\}$. We compute $\sigma($`nines`$) =$ `einns`, which leads us to the container map with exactly one entry: $\mathcal{B}_{einns} = \{$`nines`$\}$. Furthermore, we obtain the following trie:

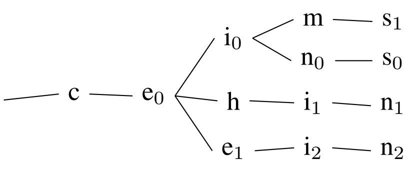

Fig. 1. Example Trie $\mathcal{T}$

In the given trie we have 4 buckets, to be specific s_0 has $\{$`nices, since`$\}$, s_1 has $\{$`mices`$\}$, n_1 has $\{$`niche`$\}$ and finally n_2 has $\{$`niece`$\}$.
Through use of Equation 11, we determine parameter $m_{needed}(|u|, v_{min}) = m_{needed}(4, 4) = 4$. The tree traversal stack $\mathcal{S}$ is initially populated with $\{([einns], *c, 0)\}$.
In the first iteration, $m_{max} = 4 \geq m_{needed}$ and $\mathcal{C}.top()$ is greater than v_c, hence we get $\mathcal{S} = \{([einns], *e_0, 0)\}$.
In the second iteration we get a character match, yielding $\mathcal{S} = \{([inns], *i_0, 1), ([inns], *h, 1), ([inns], *e_1, 1)\}$.
Using the algorithm described above, we eventually cut off subtrees at i_1, i_2 and m. Only the pairs (nines, nices) and (nines, since) are forwarded to actual Jaro-Winkler distance calculations.
Note that using the range filter prior to the trie filter is a key requirement towards its efficiency, that is, the more equally distributed the string lengths are, the better we get in terms of reduction ratio and runtime improvement. The worst case scenario is a big dataset with very large strings over a little alphabet and little to none variation in string lengths.

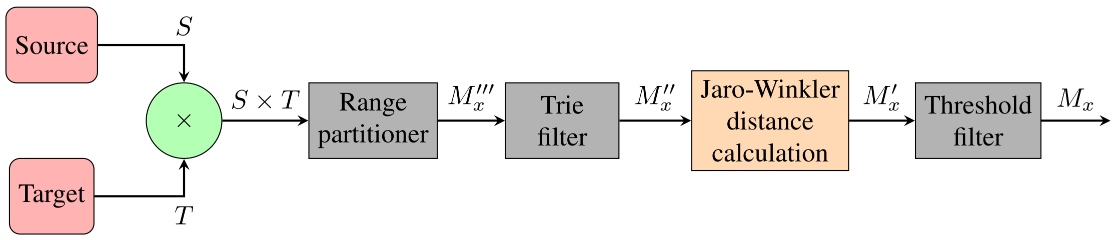

Fig. 2. Flowchart of parallel trie filter stack

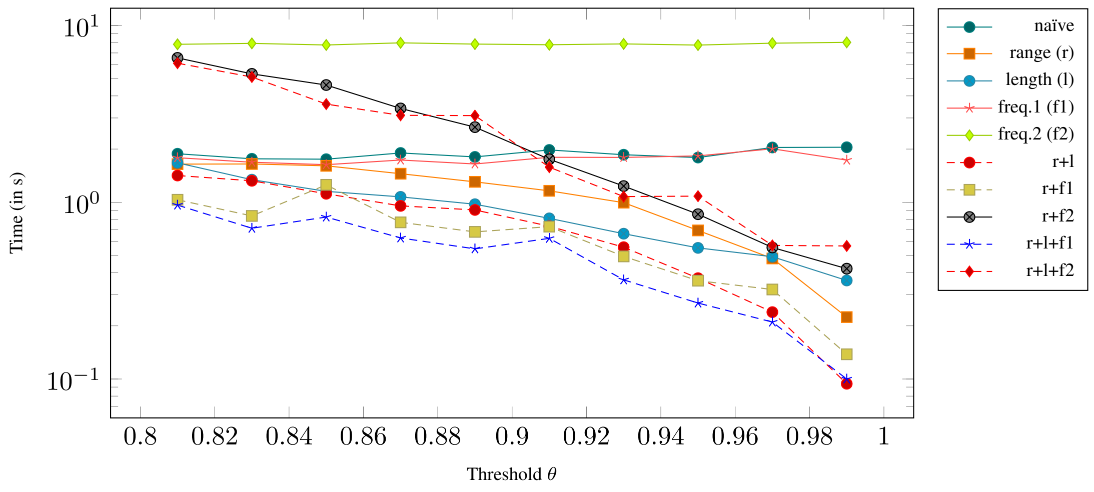

Fig. 3. Runtimes on sample of DBpedia `rdf:labels` with size 1000

3.4. Parallel Implementation

While our results suggest that the approach presented above scales well on one processor, we wanted to measure how well the partitioning induced by the upper and lower length bounds given in Equations 6 and 7 can be used to implementation our approach in parallel. Let $T_{i,j} = \{t \in T : i \leq |t| \leq j\}$ be set of strings whose length is larger or equal to i and less or equal to j. Moreover, let $S_k = \{s \in S : k = |s|\}$ be the subset of S which contains strings of length k. Finally, let $\mathcal{L}(S)$ be the set of distinct string lengths of elements of S. We distribute a subset $S \times T$ into sets $M_x''' = S_x \times T_{\rho_{min}(x), \rho_{max}(x)} \forall x \in \mathcal{L}(S)$. Note that due to the bounds in Equations 6 and 7, we are sure that all elements of T which abide by the similarity condition $\sigma(s,t) \geq \theta$ for $s \in S_x$ can be found in the subset $T_{\rho_{min}(x), \rho_{max}(x)}$. Moreover, not all elements of $S \times T$ are in the set M_x'''. In particular, pairs of strings that do not abide by the range restrictions are not considered as they are known not to be matches.

Now given the sets M_x''', we can parallelize our implementation by simply using a thread-pool-based approach. We initialize a user-given number of threads and assign each thread one of the sets M_x''' randomly. Once a thread has completed its computations, it is simply assigned a further set M_x'''. Note that we do not implement any load balancing as we were primarily interested in how well the range filter allows partitioning data. Figure 2 gives an overview of our parallel pipeline.

4. Evaluation

4.1. Experimental Setup

The aim of our evaluation was to study how well our approach performs on real data. We chose DBpedia 3.9 as a source of data for our experiments as it contains data pertaining to 1.1 million persons and thus allows for both fine-grained evaluations and scalability evaluations. As a second data source we chose LinkedGeoData to evaluate how well we perform on strings that have no relation to person names. We chose these datasets because (1) they ave been widely used in ex-

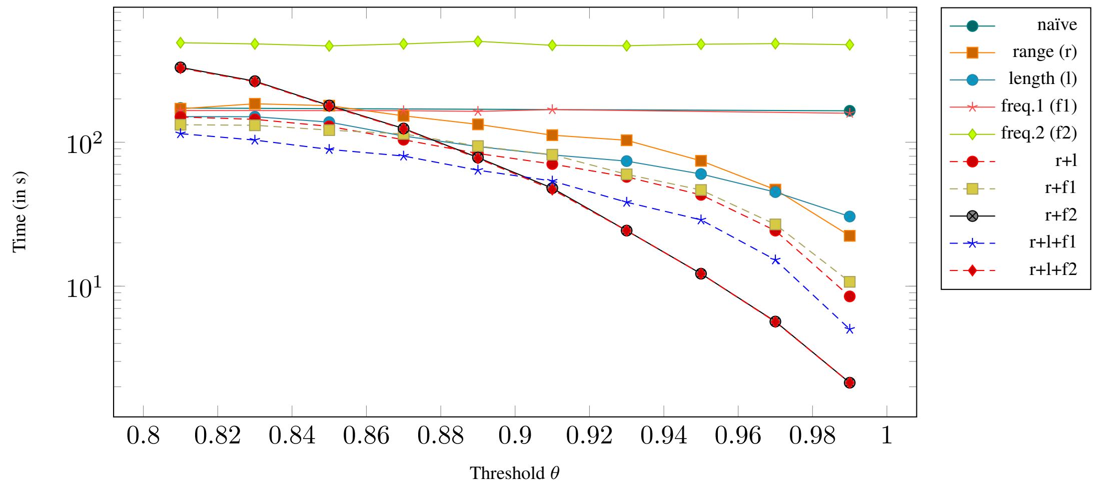

Fig. 4. Runtimes on sample of DBpedia `rdf:labels` with size 10^4

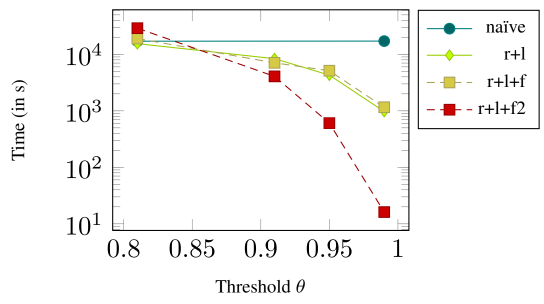

Fig. 5. Runtimes on sample of DBpedia `rdf:labels` with size 10^5, scaling threshold

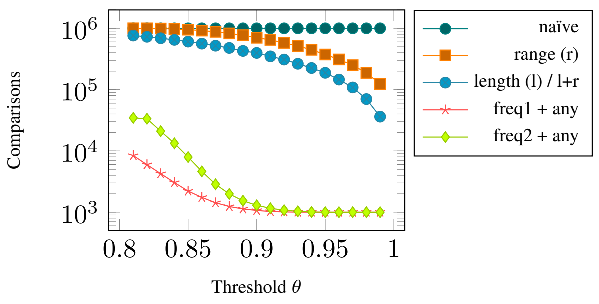

Fig. 6. Actual Jaro-Winkler computations, measured on sample of DBpedia `rdf:labels` with size 1000

periments pertaining to link discovery and (2) the distributions of string sizes in these datasets are significantly different (see Figures 7 and 8). All experiments where deduplication experiments, i.e., $S = T$. We considered the list of all `rdfs:label` in DBpedia in our runtime evaluation and added all `rdfs:labels` of the Places dataset from LinkedGeoData for scalability experiments. We also computed the number of actual Jaro-Winkler calculations carried out for 1000 strings from DBpedia. All runtime and scalability experiments were performed on a 2.5 GHz Intel Core i5 machine with 16GB RAM running OS X 10.9.3. The speedup of the parallel trie-based filter stack was measured on a Microsoft Azure VM instance with 16 CPUs and 112GB RAM.

4.2. Runtime Evaluation

In our first series of experiments, we evaluated the runtime of all filter combinations against the naïve approach on a small dataset containing 1000 labels from DBpedia. The results of our evaluation are shown in Figure 3. This evaluation suggests that all filter setups except those containing `f2` outperform the naïve approach. Moreover, the combination of all filters leads to the best overall runtime in most cases on this small sample. Overall, the results on this dataset already shows that we outperform the naïve approach by more than an order of magnitude when $\theta > 0.9$. Interestingly, the break-even point for `f1` and `f2` is reached when $\theta > 0.99$ on this small dataset. This is clearly due to the overhead necessary to create the trie overshadowing the runtime advantage engendered by using the trie to search for matches.

The runtimes on a larger sample of size 10^4 show an even better improvement (see Figure 4). This suggests that the relative improvement of our approach improves with the size of the problem. The most interesting results come from filter setups `r+f2` and `r+l+f2`. They are slower than the naïve approach on low θ but

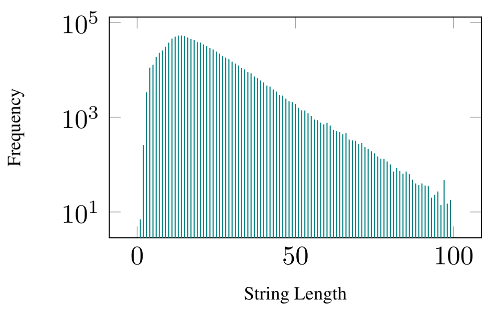

Fig. 7. String length distribution of 10^6 DBpedia `rdf:labels`

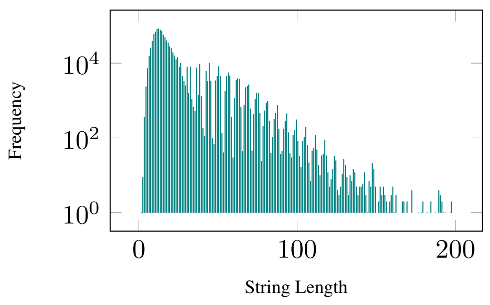

Fig. 8. String length distribution of 10^6 LinkedGeoData `rdf:labels`

after an break-even point around $0.89 \leq \theta \leq 0.91$ they outperforms the second best setup `r+l` by an order of magnitude.

4.3. *Scalability Evaluation*

The aim of the scalability evaluation was to measure how well our approach scales. In our first set of experiments, we looked at the growth of the runtime of our approach on datasets of growing sizes (see Figures 9 and 10). Our results show very clearly that `r+l+f2` is the best filter combination for datasets of large sizes. This result holds on both DBpedia and LinkedGeoData. `r+l+f2` is thus the default implementation of the Jaro-Winkler measure in LIMES. In addition, our results suggest that our approach grows linearly with the number of labels contained in S and T. This is one of our most important results as it makes clear that we can employ `r+l+f2` on large datasets and expect acceptable runtimes. Moreover, the behavior of the runtime of our default setup can be easily predicted for large datasets, which is of importance when asking users to wait for the results of the computation.

The second series of scalability experiments looked at the runtime behaviour of our approach on a large dataset with 10^5 labels (see Figure 5). Our results suggest that the runtime of our approach falls superlinearly with an increase of the threshold θ. This behaviour suggest that our approach is especially useful on clean datasets, where high thresholds can be used for link discovery.

In the third series of experiments we looked at the speedup we gain by parallelizing `r+l+f2` on the DBpedia dataset with input sizes of 10^5 and 10^6 (see Figures 11 and 12). Here, our results show that the current implementation scales up in a satisfactory manner on up to 4 processors. Running the parallel implementation on 8 and 12 processors does not bring about any considerable increase in speedup. The reason for this behavior is simply that we did not implement any load balancing. Hence, there is commonly one thread that is assigned a single large M_x''', leading to all other threads having completed their tasks but having to wait for this particular thread to terminate. Adding load balancing to the approach as well as splitting large M_x''' into smaller chunks should lead to an improvement of the scalability of our parallel implementation. These extensions will be implemented in future work.

4.4. *Comparison with existing approaches*

We compared our approach with SILK2.6.0. To this end, we retrieved all `rdfs:label` of instances of subclasses of `Person`. We only compared with SILK on small datasets (i.e., on classes with small numbers of instances) as the results on these small datasets already showed that we outperform SILK consistently.[10] Our results are shown in Table 3. They suggest that the absolute difference in runtime grows with the size of the datasets. Thus, we did not consider testing larger datasets against SILK as in the best case, we were already 4.7 times faster than SILK (Architect dataset, $\theta = 0.95$).

5. Related Work

The work presented herein is related to record linkage, deduplication, link discovery and the efficient computation of Hausdorff distances. An extensive amount of literature has been published by the database community on record linkage (see [12,6] for surveys). With regard to *time complexity*, time-efficient deduplication algorithms such as PPJoin+ [31], ED-

[10]We ran SILK with -Dthreads = 1 for the sake of fairness.

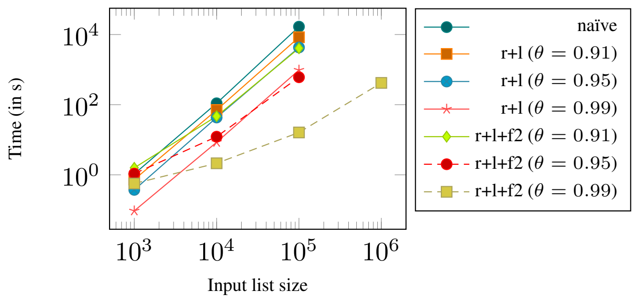

Fig. 9. Runtimes for growing input list sizes (DBPedia)

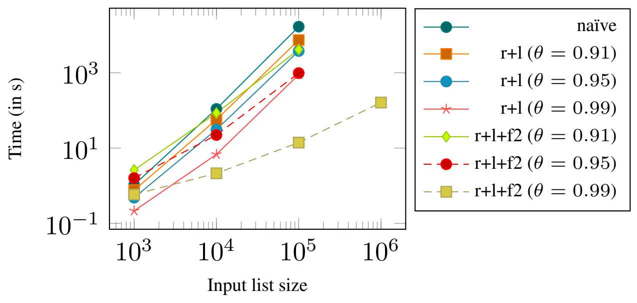

Fig. 10. Runtimes for growing input list sizes (LinkedGeoData)

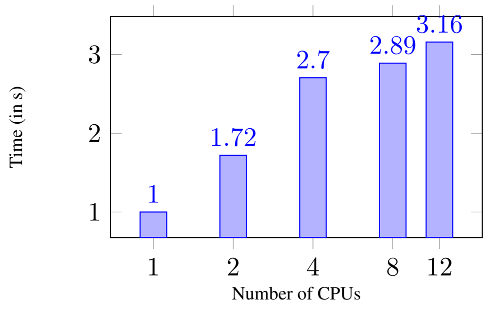

Fig. 11. Speedup of parallel algorithm (DBPedia, 10^5)

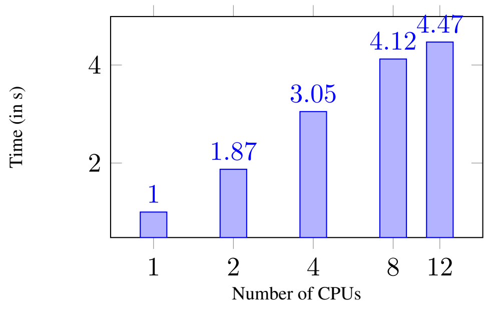

Fig. 12. Speedup of parallel algorithm (DBPedia, 10^6)

Table 3

Runtimes (in seconds) of our approach (OA) and SILK 2.6.0

DBpedia Class	Size	OA(0.8)	OA(0.9)	OA(0.95)	SILK(0.8)	SILK(0.9)	SILK(0.95)
Actors	9509	15.07	10.13	6.38	27	25	25
Architect	3544	5.58	5.48	2.32	11	11	11
Criminal	5291	11.54	7.77	4.52	18	18	18

Join [30], PassJoin [13] and TrieJoin [28] were developed over the last years. Several of these were then integrated into the hybrid link discovery framework LIMES [18]. Moreover, dedicated time-efficient approaches were developed for LD. For example, RDF-AI [26] implements a five-step approach that comprises the preprocessing, matching, fusion, interlink and post-processing of data sets. [19] presents an approach based on the Cauchy-Schwarz that allows discarding a large number of unnecessary computations. The approaches HYPPO [16] and $\mathcal{HR}^3$ [17] rely on space tiling in spaces with measures that can be split into independent measures across the dimensions of the problem at hand. Especially, $\mathcal{HR}^3$ was shown to be the first approach that can achieve a relative reduction ratio r' less or equal to any given relative reduction ratio $r > 1$. Standard blocking approaches were implemented in the first versions of SILK and later replaced with MultiBlock [9], a lossless multi-dimensional blocking technique. KnoFuss [22] also implements blocking techniques to achieve acceptable runtimes. Further approaches can be found in [27,4,23,24,7].

In addition to addressing the runtime of link discovery, several machine-learning approaches have been developed to learn link specifications (also called linkage rules) for link discovery. For example, machine-learning frameworks such as FEBRL [2] and MARLIN [1] rely on models such as Support Vector Machines [3] and decision trees [25] to detect classifiers for record linkage. RAVEN [20] relies on active learning to detect linear or Boolean classifiers. The EAGLE approach [21] combines active learning and genetic programming to detect link specifications. KnoFuss [22] goes a step further and presents an unsupervised approach based on genetic programming for finding accurate link specifications. Other record deduplication approaches based on active learning and genetic programming are presented in [5,8].

6. Conclusion and Future Work

In this paper, we present a novel approach for the efficient execution of bounded Jaro-Winkler computations. Our approach is based on three filters which allow discarding a large number of comparisons. We showed that our approach scales well with the amount of data it is faced with. Moreover, we showed that our approach can make effective use of large thresholds by reducing the total runtime of the approach considerably. We also compared our approach with the state-of-the-art framework SILK 2.6.0 and showed that we outperform it on all datasets. In future work, we will test whether our approach improves the accuracy of specification detection algorithms such as EAGLE. Moreover, we will focus on improving the quality of matches. To this end we will split input strings into tokens and use a hybrid approach as proposed by Monge and Elkan [14], which adds to complexity of the algorithm, hence allowing for further runtime improvements. Furthermore, we will extend the parallel implementation with load balancing.

References

[1] Mikhail Bilenko and Raymond J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 39–48, New York, NY, USA, 2003. ACM.

[2] Peter Christen. Febrl -: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *KDD*, pages 1065–1068, 2008.

[3] Nello Cristianini and Elisa Ricci. Support vector machines. In *Encyclopedia of Algorithms*. 2008.

[4] Philippe Cudré-Mauroux, Parisa Haghani, Michael Jost, Karl Aberer, and Hermann de Meer. idmesh: graph-based disambiguation of linked data. In *WWW*, pages 591–600, 2009.

[5] J. De Freitas, G.L. Pappa, A.S. da Silva, M.A. Gonçalves, E. Moura, A. Veloso, A.H.F. Laender, and M.G. de Carvalho. Active learning genetic programming for record deduplication. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[6] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[7] Jérôme Euzenat, Alfio Ferrara, Willem Robert van Hage, et al. Results of the Ontology Alignment Evaluation Initiative 2011. In *OM*, 2011.

[8] Robert Isele and Christian Bizer. Learning expressive linkage rules using genetic programming. *PVLDB*, 5(11):1638–1649, 2012.

[9] Robert Isele, Anja Jentzsch, and Christian Bizer. Efficient Multidimensional Blocking for Link Discovery without losing Recall. In *WebDB*, 2011.

[10] Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. Journal of the American Statistical Association 84(406):414–420, 1989.

[11] kevin Dreßler and Axel-Cyrille Ngonga Ngomo. Time-efficient execution of bounded jaro-winkler distances. In *Ontology Matching workshop at ISWC*, 2014.

[12] Hanna Köpcke and Erhard Rahm. Frameworks for entity matching: A comparison. *Data Knowl. Eng.*, 69(2):197–210, 2010.

[13] Guoliang Li, Dong Deng, Jiannan Wang, and Jianhua Feng. Pass-join: a partition-based method for similarity joins. *Proc. VLDB Endow.*, 5(3):253–264, November 2011.

[14] Alvaro Monge and Charles Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records, 1997.

[15] Axel-Cyrille Ngonga Ngomo. Orchid - reduction-ratio-optimal computation of geo-spatial distances for link discovery. In *International Semantic Web Conference (1)*, pages 395–410, 2013.

[16] Axel-Cyrille Ngonga Ngomo. A Time-Efficient Hybrid Approach to Link Discovery. In *OM*, 2011.

[17] Axel-Cyrille Ngonga Ngomo. Link discovery with guaranteed reduction ratio in affine spaces with minkowski measures. In *International Semantic Web Conference (1)*, pages 378–393, 2012.

[18] Axel-Cyrille Ngonga Ngomo. On link discovery using a hybrid approach. *J. Data Semantics*, 1(4):203–217, 2012.

[19] Axel-Cyrille Ngonga Ngomo and Sören Auer. LIMES - A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data. In *IJCAI*, pages 2312–2317, 2011.

[20] Axel-Cyrille Ngonga Ngomo, Jens Lehmann, Sören Auer, and Konrad Höffner. Raven - active learning of link specifications. In *OM*, 2011.

[21] Axel-Cyrille Ngonga Ngomo and Klaus Lyko. Eagle: Efficient active learning of link specifications using genetic programming. In *ESWC*, pages 149–163, 2012.

[22] Andriy Nikolov, Mathieu d'Aquin, and Enrico Motta. Unsupervised learning of link discovery configuration. In *ESWC*, pages 119–133, 2012.

[23] Andriy Nikolov, Victoria Uren, Enrico Motta, and Anne Roeck. Overcoming schema heterogeneity between linked semantic repositories to improve coreference resolution. In *Proceedings of the 4th Asian Conference on The Semantic Web*, ASWC '09, pages 332–346, Berlin, Heidelberg, 2009. Springer-Verlag.

[24] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Themis Palpanas, and Wolfgang Nejdl. Eliminating the redundancy in blocking-based entity resolution methods. In *Proceedings of the 11th annual international ACM/IEEE joint conference on Digital libraries*, JCDL '11, pages 85–94, New York, NY, USA, 2011. ACM.

[25] S. R. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *Systems, Man and Cybernetics, IEEE Transactions on*, 21(3):660–674, 1991.

[26] Francois 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.

[27] Dezhao Song and Jeff Heflin. Automatically generating data linkages using a domain-independent candidate selection approach. In *International Semantic Web Conference (1)*, pages 649–664, 2011.

[28] Jiannan Wang, Guoliang Li, and Jianhua Feng. Trie-join: Efficient trie-based string similarity joins with edit-distance constraints. *PVLDB*, 3(1):1219–1230, 2010.

[29] William E. Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. In *Proceedings of the Section on Survey Research*, pages 354–359, 1990.

[30] Chuan Xiao, Wei Wang, and Xuemin Lin. Ed-Join: an efficient algorithm for similarity joins with edit distance constraints. *PVLDB*, 1(1):933–944, 2008.

[31] Chuan Xiao, Wei Wang, Xuemin Lin, and Jeffrey Xu Yu. Efficient similarity joins for near duplicate detection. In *WWW*, pages 131–140, 2008.