Rapid Execution of Weighted Edit Distances

Tommaso Soru and Axel-Cyrille Ngonga Ngomo

Department of Computer Science
University of Leipzig
Augustusplatz 10, 04109 Leipzig
{tsoru|ngonga}@informatik.uni-leipzig.de

Abstract. The comparison of large numbers of strings plays a central role in research areas as diverse as record linkage, link discovery and knowledge extraction. While several standard string distance and similarity measures have been developed with these explicit goals in mind, similarities and distances learned out of the data have been shown to often perform better with respect to the F-measure that they can achieve. Still, the practical use of data-specific measures is often hindered by one major factor: their runtime. While time-efficient algorithms that allow scaling to millions of strings have been developed over the last years, data-specific versions of these measures are usually slow to run and require significantly more time for the same task. In this paper, we present an approach for the time-efficient execution of weighted edit distances. Our approach is based on a sequence of efficient filters that allow reducing the number of candidate pairs for which the weighted edit distance has to be computed. We also show how existing time-efficient deduplication approaches based on the edit distance can be extended to deal with weighted edit distances. We compare our approach with such an extension of PassJoin on benchmark data and show that we outperform it by more than one order of magnitude.

1 Introduction

The computation of string similarities plays a central role in manifold disciplines ranging from computational biology [4] to link discovery on the Web of Data[1] [21]. Over the last decades, manifold domain-specific string similarities have been developed for improving the accuracy of automatic techniques that rely on them. For example, the Jaro-Winkler similarity was developed especially to perform well on person names [22]. Still, newer works in machine learning have shown that learning string similarities directly from data can lead to algorithms with a performance superior to that of those which rely on standard similarity measures. Especially, work on link discovery on the Web of Data [21] has shown that data-specific weighted edit distances can lead to higher F-measures for link specifications.

[1] Throughout this paper, we use the expression "link discovery" to mean the discovery of typed relations that link instances from knowledge bases on the Web of Data. This discipline is related to entity resolution and deduplication and known from databases.

One main problem has yet plagued the approaches which rely on string similarity measures learned from data: their runtime. While dedicated algorithms for the time-efficient comparison of large volumes of data have been developed over the last years (e.g., PPJoin+ [24], EDJoin [23], PassJoin [13] and TrieJoin [7]), the time-efficient computation of data-specific string similarities has been paid little attention to. Thus, running the data-specific counterparts of standard similarity measures is often orders of magnitude slower. Previous work have circumvented this problem in manifold ways, including the execution of approximations of the data-specific similarity measure. For example, weighted edit distances are sometimes approximated by first computing the edit distance between two strings A and B and only subsequently applying the weight of each of the edit operations [11]. Other approximations can be found in [3, 2].

In this paper, we address the problem of the time-efficient computation of weighted edit distances by presenting a novel approach, REEDED. Our approach uses weight bounds on the input cost matrix to efficiently discard similarity computations that would lead to dissimilar pairs. By these means, REEDED can outperform state-of-the-art approaches for the computation of edit distances by more than one order of magnitude on real datasets. We explain our approach on one of its prime areas of application (i.e., link discovery [21]) by using the data shown in Table 1 as example. Here, the task is to detect possible pairs $(s, t) \in S \times T$ such that s `owl:sameAs` t, where S is a set of source resources and T is a set of target resources.

The contributions of our paper can be summarized as follows:

- We present the REEDED approach for the time-efficient computation of weighted edit distances.
- We prove the completeness and correctness of REEDED's results formally.
- We compare REEDED with a weighted version of the state-of-the-art approach PassJoin on 4 datasets and show that we outperform it by more than one order of magnitude.

The rest of this paper is structured as follows: In Section 2, we present preliminaries to our work. Thereafter, we give some insights into the intuitions behind our work (Section 3). We then present the REEDED approach formally in Section 4 and prove that our results are both complete and correct. In Section 6, we evaluate our approach on four datasets and show that we outperform the state of the art in all settings. Finally, we conclude with Section 8 after giving a brief overview of related work in Section 7.

2 Preliminaries

2.1 Notation and Problem Statement

Let Σ be an alphabet and Σ^* be the set all sequences that can be generated by using elements of Σ. We call the elements of Σ characters and assume that Σ contains the empty character ϵ. The edit distance – or Levenshtein distance –

of two strings $A \in \Sigma^*$ and $B \in \Sigma^*$ is the minimum number of edit operations that must be performed to transform A into B [12]. An *edit operation* can be the insertion or the deletion of a character, or the substitution of a character with another one. In a *plain* edit distance environment, all edit operations have a cost of 1. Thus, the distance between the strings "Generalized" and "Generalised" is the same as the distance between "Diabetes Type I" and "Diabetes Type II". Yet, while the first pair of strings is clearly semantically equivalent for most applications, the elements of the second pair bears related yet significantly different semantics (especially for medical applications). To account for the higher probability of edit operations on certain characters bearing a higher semantic difference, weighted edit distances were developed. In a *weighted* edit distance environment, a cost function $cost : \Sigma \times \Sigma \rightarrow [0, 1]$ assigned to each of the possible edit operations. The totality all of costs can be encoded in a *cost matrix* M. The cost matrix is quadratic and of dimensions $|\Sigma| \times |\Sigma|$ for which the following holds:

$$\forall i \in \{1, \ldots, |\Sigma|\}\ m_{ii} = 0 \tag{1}$$

The entry m_{ij} is the cost for substituting the i^{th} character c_i of Σ with the j^{th} character c_j of the same set. Note that if $c_i = \epsilon$, m_{ij} encode the insertion of c_j. On the other hand, if $c_j = \epsilon$, m_{ij} encode the deletion of c_i.

In most applications which require comparing large sets of strings, string similarities are used to address the following problem: Given a set S of source strings and a set T of target strings, find the set $\mathcal{R}(S, T, \delta_p, \theta)$ of all pairs $(A, B) \in S \times T$ such that

$$\delta_p(A, B) \leq \theta \tag{2}$$

where θ is a distance threshold and δ_p is the plain edit distance. Several scalable approaches have been developed to tackle this problem for plain edit distances [13, 24, 23]. Still, to the best of our knowledge, no scalable approach has been proposed for finding all $(A, B) \in S \times T$ such that $\delta(A, B) \leq \theta$ for weighted edit distances δ. In this paper we address exactly this problem by presenting REEDED. This approach assumes that the computation of weighted edit distances can be carried out by using an extension of the dynamic programming approach used for the plain edit distance.

2.2 Extension of Non-Weighted Approaches

All of the approaches developed to address the problem at hand with the plain edit distance can be easily extended to deal with weighted edit distances for which the dynamic programming approach underlying the computation of the plain edit distance still holds. Such an extension can be carried out in the following fashion: Let

$$\mu = \min_{0 \leq i,j \leq |\Sigma|} m_{ij}. \tag{3}$$

Then, if the weighted edit distance between two strings A and B is d, then at most d/μ edit operations were carried out to transform A into B. By using this

insight, we can postulate that for any weighted edit distance δ with cost matrix M, the following holds

$$\forall A \in \Sigma^* \; \forall B \in \Sigma^* \; \delta(A,B) \leq \theta \rightarrow \delta_p(A,B) \leq \frac{\theta}{\mu}. \tag{4}$$

Thus, we can reduce the task of finding the set $\mathcal{R}(S,T,\delta,\theta)$ to that of first finding $\mathcal{R}(S,T,\delta_p,\theta/\mu)$ and subsequently filtering $\mathcal{R}(S,T,\delta_p,\theta/\mu)$ by using the condition $\delta(A,B) \leq \theta$. To the best of our knowledge, PassJoin [13] is currently the fastest approach for computing $\mathcal{R}(S,T,\delta_p,\theta)$ with plain edit distances. We thus extended it to deal with weighted edit distances and compared it with our approach. Our results show that we outperform the extension of PassJoin by more than one order of magnitude.

3 The REEDED Approach

3.1 Overview

Our approach REEDED (Rapid Execution of Weighted Edit Distances) aims to compute similar strings using weighted edit distance within a practicable amount of time. The REEDED approach is basically composed of three nested filters as shown in Figure 1, where each filter takes a set of pairs as input and yields a subset of the input set according to a predefined rule. In the initial step of REEDED, the input data is loaded from S, a source data set, and T, a target data set. Their Cartesian product $S \times T$ is the input of the first length-aware filter. The output of the first filter $\mathcal{L}$ is the input of the second character-aware filter. The weighted edit distance will be calculated only for the pairs that pass through the second filter, i.e. set $\mathcal{N}$. The final result $\mathcal{A}$ is the set of pairs whose weighted edit distance is less or equal than a threshold θ. Note that pairs are processed one by one. This ensures that our algorithm performs well with respect to its space complexity.

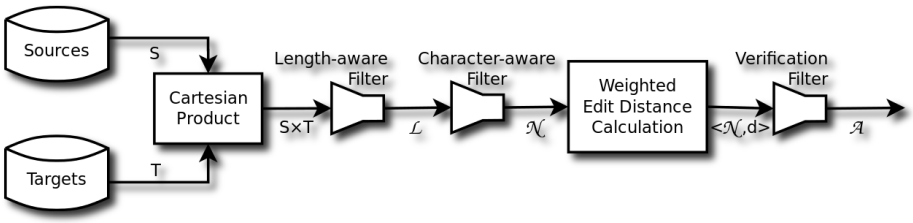


Fig. 1. Flowchart of the REEDED approach.

More formally, our algorithm for computing $\mathcal{R}(S,T,\delta,\theta)$ works as described in Alg.1. The filtering is performed at rows 4, 8, and 10. Note that the functions

Algorithm 1 Main algorithm.

Require: S, T, p, q, θ
Ensure: $\mathcal{A}$: a set of pairs

$\tau \leftarrow \theta / \min(m_{ij})$
for all $s \in S$ **do**
  for all $t \in T$ **do**
    if $||s| - |t|| \leq \tau$ **then**
      $C_s \leftarrow$ CharsOf(s)
      $C_t \leftarrow$ CharsOf(t)
      $C \leftarrow C_s \oplus C_t$
      if $\lceil |C|/2 \rceil \leq \tau$ **then**
        $\delta \leftarrow$ WeightedEditDistance(s, t)
        if $\delta \leq \theta$ **then**
          $\mathcal{A} \leftarrow \mathcal{A} \cup \langle s, t \rangle$
        end if
      end if
    end if
  end for
end for
return $\mathcal{A}$

function CharsOf(s)
  for $i = 0 \rightarrow |s| - 1$ **do**
    $C \leftarrow C \cup s[i]$
  end for
  return C
end function

function WeightedEditDistance(a, b)
  $\forall i, j \; L[i, j] \leftarrow 0$
  for $i = 1 \rightarrow |a|$ **do**
    $L[i, 1] \leftarrow i$
  end for
  for $j = 1 \rightarrow |b|$ **do**
    $L[1, j] \leftarrow j$
  end for
  for $i = 1 \rightarrow |a|$ **do**
    for $j = 1 \rightarrow |b|$ **do**
      $L[i, j] \leftarrow min\{L[i-1, j-1] + \text{GetSubCost}(a[i], b[j]),$
        $L[i-1, j] + \text{GetDelCost}(a[i]), L[i, j-1] + \text{GetInsCost}(b[j])\}$
    end for
  end for
  return $L[|a|, |b|]$
end function

GetSubCost, GetDelCost and GetInsCost refer to the costs assigned to every edit operation. These values are stored into the *cost matrix*. In the following, we explain each of the key steps of our algorithm by using the data sets shown in Table 1 as example. We will assume the costs $sub(c, C) = sub(t, T) = 0.5$, $ins(s) = 0.6$ and $sub(1, 2) = sub(2, 1) = 0.7$. All other substitutions, deletions and insertions will be assumed to have a cost of 0.

Table 1. Example data sets.

Sources (S)	Targets (T)
id name	*id name*
s_1 Basal cell carcinoma	t_1 Basal Cell Carcinoma
s_2 Blepharophimosi	t_2 Blepharophimosis
s_3 Blepharospasm	t_3 Blepharospasm
s_4 Brachydactyly type A1	t_4 Brachydactyly Type A1
s_5 Brachydactyly type A2	t_5 Brachydactyly Type A2

3.2 Key Assumption

Similarly to the extensions of plain edit distances for weighted edit distances, REEDED assumes the dynamic programming approach commonly used for computing plain edit distances can be used for computing the weighted edit distance described by the cost matrix M. With respect to M, this assumption translates to the weights in the matrix being such that there is no sequence of two edit operations m_{ij} and $m_{i'j'}$ that is equivalent to a third edit operation $m_{i''j''}$ with

$$m_{i''j''} > m_{ij} + m_{i'j'}. \tag{5}$$

for $(i \neq j) \wedge (i' \neq j') \wedge (i'' \neq j'')$. Formally, we can enforce this condition of the cost matrix M by ensuring that

$$\exists k > 0 \ \forall m_{ij} : k < m_{ij} \leq 2k. \tag{6}$$

Given that the entries in cost matrices are usually bound to have a maximal value of 1, we will assume without restriction of generality that

$$\forall i \in \{1, \ldots, |\Sigma|\} \forall j \in \{1, \ldots, |\Sigma|\} \ i \neq j \rightarrow 0.5 < m_{ij} \leq 1. \tag{7}$$

Thus, in the following, we will assume that $\forall m_{ij} : m_{ij} > 0.5$. We discuss the case where this assumption is not given in Section 5 of the paper.

3.3 Length-aware Filter

The *length-aware filter* is the first filter of REEDED. Once the data sets have been loaded, the Cartesian product

$$S \times T = \{\langle s, t\rangle : s \in S, t \in T\} \tag{8}$$

is computed, which in our example corresponds to $\{\langle s_1, t_1\rangle, \langle s_1, t_2\rangle, \ldots, \langle s_5, t_5\rangle\}$. The basic insight behind the first filter is that given two strings s and t with lengths $|s|$ resp. $|t|$, we need at least $||s| - |t||$ edit operations to transform s into t. Now given that each edit operation costs at least μ, the cost of transforming s to t will be at least $\mu||s| - |t||$. Consequently, the rule which the filter relies on is the following:

$$\langle s, t\rangle \in \mathcal{L} \;\Rightarrow\; \langle s, t\rangle \in S \times T \wedge ||s| - |t|| \leq \theta/\mu. \tag{9}$$

In the following, we will set $\tau = \theta/\mu$, where μ is as defined in Equation (3).

In our example, let us assume $\theta = 1$ and $m_{ij} \in (0.5, 1.0]$. Then, $\tau = 2$. If we assume that *S.name* has been mapped to *T.name*, then at the end of this step, 13 of the 25 initial pairs in $S \times T$ are dismissed. The remaining 8 pairs are:

$$\mathcal{L} = \{\langle s_1, t_1\rangle, \langle s_2, t_2\rangle, \langle s_3, t_3\rangle, \langle s_4, t_4\rangle, \langle s_5, t_5\rangle, \langle s_2, t_3\rangle, \langle s_4, t_5\rangle, \langle s_5, t_4\rangle\}\,. \tag{10}$$

3.4 Character-aware Filter

The second filter is the *character-aware filter* which only selects the pairs of strings that do not differ by more than a given number of characters. The intuition behind the filter is that given two strings s and t, if $|C|$ is the number of characters that do not belong to both strings, we need at least $\lceil |C|/2 \rceil$ operations to transform s into t. As above, the cost of transforming s to t will be at least $\mu\lceil |C|/2 \rceil$.

The characters of each string are collected into two sets, respectively C_s for the source string and C_t for the target string. Since s and t may contain more than one occurrence of a single character, characters in C_s and C_t are enumerated. Then, the algorithm computes their exclusive disjunction C:

$$C = C_s \oplus C_t. \tag{11}$$

Finally, the filter performs the selection by applying the rule:

$$\langle s, t\rangle \in \mathcal{N} \iff \langle s, t\rangle \in \mathcal{L} \wedge \left\lceil \frac{|C|}{2} \right\rceil \leq \tau. \tag{12}$$

In our example, a further pair can be dismissed by these means, leading to the set of remaining pairs being as follows:

$$\mathcal{N} = \{\langle s_1, t_1\rangle, \langle s_2, t_2\rangle, \langle s_3, t_3\rangle, \langle s_4, t_4\rangle, \langle s_5, t_5\rangle, \langle s_4, t_5\rangle, \langle s_5, t_4\rangle\}$$

The pair that is rejected is $\langle s_2, t_3\rangle$, for which $C = \{h_1, i_1, o_1, i_2, a_1, s_1\}$, which leads to the rule not being satisfied. Note that pair $\langle s_3, t_2\rangle$ could have passed through the filter, but it was not in the length-aware selection.

3.5 Verification

For all the pairs left in $\mathcal{N}$, the weighted edit distance among is calculated. After that, the third filter selects the pairs whose distance is less or equal than a threshold θ.

$$\langle s,t\rangle \in \mathcal{A} \iff \langle s,t\rangle \in \mathcal{N} \wedge \delta(s,t) \leq \theta \tag{13}$$

In our example data sets, the set

$$\mathcal{A} = \{\langle s_1,t_1\rangle, \langle s_2,t_2\rangle, \langle s_3,t_3\rangle, \langle s_4,t_4\rangle, \langle s_5,t_5\rangle\} \tag{14}$$

is the final result of the selection. Note that the pairs $\langle s_4,t_5\rangle$ and $\langle s_5,t_4\rangle$ are discarded, because their distance (1.2) is greater than the threshold (1.0).

4 Correctness and Completeness

In the previous section, we showed that our approach performs as it should on an example. In this section, we prove formally that REEDED does so on any pair of datasets. We achieve this goal by proving that REEDED is both correct and complete, where:

- We say that an approach is *correct* if the output O it returns is such that $O \subseteq \mathcal{R}(S,T,\delta,\theta)$.
- Approaches are said to be *complete* if their output O is a superset of $\mathcal{R}(S,T,\delta,\theta)$, i.e., $O \supseteq \mathcal{R}(S,T,\delta,\theta)$.

As stated in Section 3, REEDED consists of three nested filters, each of which creates a subset of pairs.

$$\mathcal{A} \subseteq \mathcal{N} \subseteq \mathcal{L} \subseteq S \times T \tag{15}$$

For the purpose of clearness, we name each filtering rule:

$$R_1 \Leftrightarrow ||s| - |t|| \leq \tau$$

$$R_2 \Leftrightarrow \left\lceil \frac{|C|}{2} \right\rceil \leq \tau$$

$$R_3 \Leftrightarrow \delta(s,t) \leq \theta$$

Using the definitions at (9, 12, 13), each subset can be redefined as follows:

$$\mathcal{L} = \{\langle s,t\rangle \in S \times T : R_1\} \tag{16}$$

$$\mathcal{N} = \{\langle s,t\rangle \in S \times T : R_1 \wedge R_2\} \tag{17}$$

$$\mathcal{A} = \{\langle s,t\rangle \in S \times T : R_1 \wedge R_2 \wedge R_3\} \tag{18}$$

We then introduce $\mathcal{A}^*$ as the set of pairs whose weighted edit distance is less or equal than the threshold θ.

$$\mathcal{A}^* = \{\langle s,t\rangle \in S \times T : \delta(s,t) \leq \theta\} = \{\langle s,t\rangle \in S \times T : R_3\} \tag{19}$$

Lemma 1 *The REEDED approach is correct and complete.*

Proof. This lemma is equivalent to showing that $\mathcal{A} = \mathcal{A}^*$. Let us consider all the pairs in $\mathcal{A}$. REEDED's correctness follows directly from (18). All the pairs $\langle s, t\rangle \in \mathcal{A}$ satisfy rule R_3, which also defines $\mathcal{A}^*$. Thus $\mathcal{A} \subseteq \mathcal{A}^*$. All we need to show now is REEDED's completeness, i.e., that none of the pairs discarded by the filters actually belongs to $\mathcal{A}^*$.

We are given two strings s and t. Their length differs of $||s| - |t||$ characters, which is also the smallest number of edit operations (all insertions or all deletions) that must be performed to transform s into t. Therefore, the lower bound of their weighted edit distance is the product among the difference and the minimum edit cost:

$$||s| - |t|| \cdot \mu \leq \delta(s, t) \tag{20}$$

When rule R_3 applies, we have $\delta(s, t) \leq \theta$, so from inequality (20):

$$||s| - |t|| \leq \tau \tag{21}$$

which leads to $R_3 \Rightarrow R_1$. Thus, set $\mathcal{A}$ can be rewritten as:

$$\mathcal{A} = \{\langle s, t\rangle \in S \times T : R_2 \wedge R_3\} \tag{22}$$

Now, starting from two strings s and t, we call C the exclusive disjunction of their characters.

$$C = C_s \oplus C_t \tag{23}$$

so the number of the operations performed to transform s into t is included between $\lceil |C|/2 \rceil$ (all substitutions plus an insertion or a deletion if $|C|$ is odd) and $|C|$ (all insertions or deletions). In other words, the lower bound of their weighted edit distance is

$$\lceil |C|/2 \rceil \, \mu \leq \delta(s, t) \tag{24}$$

Again, when rule R_3 applies, we have $\delta(s, t) \leq \theta$, so from inequality (24):

$$\lceil |C|/2 \rceil \leq \tau \tag{25}$$

which leads to $R_3 \Rightarrow R_2$. Therefore, we rewrite again set $\mathcal{A}$ as:

$$\mathcal{A} = \{\langle s, t\rangle \in S \times T : R_3\} \tag{26}$$

which is the definition of $\mathcal{A}^*$ (19). Thus, $\mathcal{A} = \mathcal{A}^*$.

5 Extension to all weighted edit distances

As already stated in Section 3.2, our approach assumes that there is no sequence of two or more operations that leads to the same result as a single operation but

is less expensive. This scenario never occurs on plain edit distances, because all the operations have the same cost. Yet, when working with weighted edit distances, it is central to take into account that each operation is equivalent to concatenations (denoted as $\circ$) of operations as expressed in the following equivalence rules:

$$sub(x, y) \equiv del(x) \circ ins(y) \tag{27}$$
$$sub(x, y) \equiv sub(x, z) \circ sub(z, y) \tag{28}$$
$$ins(x) \equiv ins(z) \circ sub(z, x) \tag{29}$$
$$del(x) \equiv sub(x, z) \circ del(z) \tag{30}$$

where $x, y, z \in \Sigma$. For example, if $sub(a, b) = 0.9$, $sub(a, c) = 0.1$ and $sub(c, b) = 0.1$, then the sequence of operations $sub(a, b) \circ sub(b, c)$ has to be preferred to $sub(a, b)$. This implies that if the condition expressed in Eq. (5) does not hold, then each equivalence rule presented above has to be considered during the construction of the Levenshtein matrix L (Alg.1 row 34). One way of going about addressing this problem is simply by replacing the cost of $sub(x, y)$ with the cost of the cheapest sequence of operations seq that is equivalent to $sub(x, y)$ if seq's cost is smaller than $sub(x, y)$'s cost. This goal can be achieved by implementing a preprocessing of the matrix M which computes all the possible sequences of operations that can potentially lead to $sub(x, y)$ but are less costly. Such a preprocessing can be implemented efficiently by using the A* algorithm with $h(c_i, c_j) = m_{ij}$ as heuristic function. Moreover, the depth of the search tree can be limited to $d = \lfloor 1/\mu \rfloor$.

6 Evaluation

The goal of our evaluation was to quantify how well REEDED performs in comparison to the state of the art. We thus compared REEDED with the extension of PassJoin as proposed in [13]. We chose PassJoin because it was shown to outperform other approaches for the efficient computation of edit distances, incl. EDJoin [23] and TrieJoin [7]. Note that we did run an implementation of EdJoin on the DBLP dataset presented below and it required approximately twice the runtime of PassJoin.

6.1 Experimental Setup

We compared the approaches across several distance thresholds on four different datasets that were extracted from real data (see Table 2).[2] The first two of these data sets contained publication data from the datasets DBLP and ACM. The third and fourth dataset contained product labels from the product catalogs Google Products and ABT [10]. We chose these datasets because they were

[2] The data used for the evaluation is publicly available at http://dbs.uni-leipzig.de/en/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution.

extracted from real data sources and because of the different string length distribution across them. By running our experiments on these data sets, we could thus ensure that our results are not only valid on certain string length distributions. As weight matrix we used a *confusion matrix* built upon the frequency of typographical errors presented in [9]. The original confusion matrices report the number of occurrences f for each error:

$$\Phi^S = \{f_{ij} : \text{substitution of } i \text{ (incorrect) for } j \text{ (correct)}\} \tag{31}$$

$$\Phi^I = \{f_{ij} : \text{insertion of } j \text{ after } i\} \tag{32}$$

$$\Phi^D = \{f_{ij} : \text{deletion of } j \text{ after } i\} \tag{33}$$

For insertion and deletion, we calculate the total frequency:

$$\omega_j^I = \sum_i \Phi_{ij}^I \tag{34}$$

$$\omega_j^D = \sum_i \Phi_{ij}^D \tag{35}$$

The weights of our weight matrix are thus defined as:

$$m_{ij} = \begin{cases} 1 - \frac{\Phi_{ij}^S}{2\max(\Phi^S)} & : i \neq \epsilon \wedge j \neq \epsilon \\ 1 - \frac{\omega_j^I - \min(\omega^I)}{2(\max(\omega^I) - \min(\omega^I))} & : i = \epsilon \wedge j \neq \epsilon \\ 1 - \frac{\omega_i^D - \min(\omega^D)}{2(\max(\omega^D) - \min(\omega^D))} & : i \neq \epsilon \wedge j = \epsilon \end{cases} \tag{36}$$

In other words, the higher the probability of an error encoded in m_{ij}, the lower its weight.

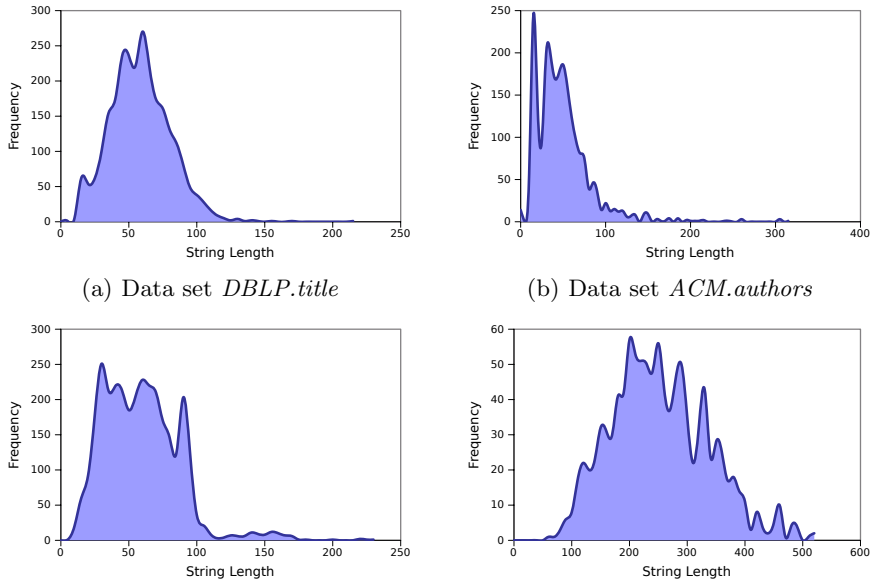
Table 2. Data sets

Source	Size	Property used	Average string length
DBLP	2,616	Title	56.36
ACM	2,295	Authors	46.64
Google Products	3,226	Product name	57.02
ABT	1,081	Product description	248.18

All experiments were carried out on a 64-bit server running Ubuntu 10.0.4 with 4 GB of RAM and a 2.5 GHz XEON CPU. Each experiment was run 5 times.

6.2 Results

In Figure 2 we show the string length distribution in the data sets. The results of our experiments are shown in Table 3. Our results show clearly that REEDED



(a) Data set *DBLP.title* (b) Data set *ACM.authors*

(c) Data set *GoogleProducts.name* (d) Data set *ABT.description*

Fig. 2. Distribution of string lengths.

outperforms PassJoin in all experimental settings. On the DBLP dataset (average string length = 56.36), REEDED is already 2 times faster than PassJoin for the threshold 2. For $\theta = 4$, we reach an order of magnitude in difference with runtimes of 25.91 (REEDED) and 246.93 (PassJoin). The runtime of REEDED seems to grow quasi-linearly with increasing values of θ. The results on ACM corroborate the results for the two algorithms. Here, we are 2.16 times faster than PassJoin for $\theta = 2$ and 6.57 times faster for $\theta = 5$. We achieve similar results on the Google Products dataset and are an order of magnitude faster than PassJoin for $\theta = 4$ already. The results we achieve the ABT dataset allow deriving further characteristics of REEDED. Here, the algorithm scales best and requires for $\theta = 5$ solely 1.58 times the runtime it required for $\theta = 1$. This is clearly due to the considerably longer strings contained in this dataset.

We analyzed the results on each of the filters in our approach and measure the reduction ratio (given by $1 - |\mathcal{N}|/|S \times T|$) achieved by the length-aware and character-aware filters. Table 4 shows the set sizes at each filtering step. Both the first and the second filter reduce the number of selected pairs by one or two orders of magnitude for all the datasets. As expected, the length-aware filter is most effective on datasets with large average string lengths. For example, only 1.9% of the Cartesian product of the ABT dataset makes it through the first filter for $\theta = 1$ while the filter allows 6.8% of the DBLP Cartesian product through for $\theta = 1$. One interesting characteristic of the approach is that the size of the $\mathcal{L}$ grows quasi linearly with the value of θ. The character-aware filter

Table 3. Runtime results in seconds.

		PassJoin		**REEDED**	
Dataset	θ	average	st.dev.	average	st.dev.
DBLP.title	1	10.75	± 0.92	10.38	± 0.35
	2	30.74	± 5.00	15.27	± 0.76
	3	89.60	± 1.16	19.84	± 0.14
	4	246.93	± 3.08	25.91	± 0.29
	5	585.08	± 5.47	37.59	± 0.43
ACM.authors	1	9.07	± 1.05	6.16	± 0.07
	2	18.53	± 0.22	8.54	± 0.29
	3	42.97	± 1.02	12.43	± 0.47
	4	98.86	± 1.98	20.44	± 0.27
	5	231.11	± 2.03	35.13	± 0.35
GoogleProducts.name	1	17.86	± 0.22	15.08	± 2.50
	2	62.31	± 6.30	20.43	± 0.10
	3	172.93	± 1.59	27.99	± 0.19
	4	475.97	± 5.34	42.46	± 0.32
	5	914.60	± 10.47	83.71	± 0.97
ABT.description	1	74.41	± 1.80	24.48	± 0.41
	2	140.73	± 1.40	27.71	± 0.29
	3	217.55	± 7.72	30.61	± 0.34
	4	305.08	± 4.78	34.13	± 0.30
	5	410.72	± 3.36	38.73	± 0.44

seems to have the opposite behavior to the length-aware filter and can discard more string pair on data with small average string lengths. For example, less than 1% of $\mathcal{L}$ makes it through the filter for $\theta = 1$ on the DBLP dataset while 5.1% of $\mathcal{L}$ makes it through the same filter for $\theta = 1$ on ABT.

We also measured the runtime improvement as well as the precision and recall we achieved by combining REEDED with the ACIDS approach and applying this combination to the datasets reported in [21]. The results are shown in Table 5. For the datasets on which the edit distance can be used, the approach achieves a superior precision and recall than state-of-the-art approaches (such as MARLIN [5] and Febrl [6]) which do not rely on data-specific measures. Yet, on more noisy datasets, the approach leads to poorer results. In particular, the edit distance has been shown not to be a good measure when the strings to be compared are too long. Also, the words contained in the source string may be completely different from the words contained in the target string, yet referring to the same meaning. A notable shortcoming of the ACIDS approach is the runtime, wherein the learning system iterated for at least 7 hours to find the weight configuration of the weighted edit distance and optimize the classification [21]. As shown in Table 5, REEDED enhances the execution time of ACIDS reducing the total runtime by 3 orders of magnitude on the DBLP–ACM and the ABT–Buy dataset.

Table 4. Numbers of pairs (s, t) returned by each filter. RR stands for the reduction ratio achieved by the combination of length-aware and character-aware filters.

DBLP.title	$\theta = 1$	$\theta = 2$	$\theta = 3$	$\theta = 4$	$\theta = 5$
$\lvert S \times T \rvert$	6,843,456	6,843,456	6,843,456	6,843,456	6,843,456
$\lvert \mathcal{L} \rvert$	465,506	832,076	1,196,638	1,551,602	1,901,704
$\lvert \mathcal{N} \rvert$	4,320	4,428	5,726	11,382	30,324
$\lvert \mathcal{A} \rvert$	4,315	4,328	4,344	4,352	4,426
$RR(\%)$	99.94	99.94	99.92	99.83	99.56
ACM.authors	$\theta = 1$	$\theta = 2$	$\theta = 3$	$\theta = 4$	$\theta = 5$
$\lvert S \times T \rvert$	5,262,436	5,262,436	5,262,436	5,262,436	5,262,436
$\lvert \mathcal{L} \rvert$	370,538	646,114	901,264	1,139,574	1,374,482
$\lvert \mathcal{N} \rvert$	3,820	5,070	24,926	104,482	218,226
$\lvert \mathcal{A} \rvert$	3,640	3,708	3,732	3,754	3,946
$RR(\%)$	99.93	99.90	99.53	98.01	95.85
GooglePr.name	$\theta = 1$	$\theta = 2$	$\theta = 3$	$\theta = 4$	$\theta = 5$
$\lvert S \times T \rvert$	10,407,076	10,407,076	10,407,076	10,407,076	10,407,076
$\lvert \mathcal{L} \rvert$	616,968	1,104,644	1,583,148	2,054,284	2,513,802
$\lvert \mathcal{N} \rvert$	4,196	4,720	9,278	38,728	153,402
$\lvert \mathcal{A} \rvert$	4,092	4,153	4,215	4,331	4,495
$RR(\%)$	99.96	99.95	99.91	99.63	95.53
ABT.description	$\theta = 1$	$\theta = 2$	$\theta = 3$	$\theta = 4$	$\theta = 5$
$\lvert S \times T \rvert$	1,168,561	1,168,561	1,168,561	1,168,561	1,168,561
$\lvert \mathcal{L} \rvert$	22,145	38,879	55,297	72,031	88,299
$\lvert \mathcal{N} \rvert$	1,131	1,193	1,247	1,319	1,457
$\lvert \mathcal{A} \rvert$	1,087	1,125	1,135	1,173	1,189
$RR(\%)$	99.90	99.90	99.89	99.88	99.87

7 Related Work

Our work is mostly related to the rapid execution of similarity functions and link discovery. Time-efficient string comparison algorithms such as PPJoin+ [24], EDJoin [23], PassJoin [13] and TrieJoin [7] were developed for the purpose of entity resolution and were integrated into frameworks such as *LIMES* [16]. In addition to time-efficient string similarity computation approaches for entity resolution, approaches for the efficient computation string and numeric similarities were developed in the area of link discovery. For example, [17] presents an approach based on the Cauchy-Schwarz inequality. The approaches HYPPO [14] and $\mathcal{HR}^3$ [15] 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 $r > 1$. Another way to go about computing $\mathcal{R}(S, T, \delta, \theta)$ lies in the use of lossless blocking approaches such MultiBlock [8].

Manifold approaches have been developed on string similarity learning (see, e.g., [20, 5, 3, 2]). [5] for example learns edit distances by employing batch learn-

Table 5. Results for the combination of ACIDS and REEDED. The runtimes in the 2 rows at the bottom are in seconds.

	DBLP–ACM		**DBLP–Scholar**		**ABT–Buy**	
Labeled examples	20	40	20	40	20	40
F-score (%)	88.98	97.92	70.22	87.85	0.40	0.60
Precision (%)	96.71	96.87	64.73	91.88	0.20	0.30
Recall (%)	82.40	99.00	76.72	84.16	100.00	100.00
Without REEDED	27,108	26,316	30,420	30,096	44,172	43,236
With REEDED	14.25	14.24	668.62	668.62	13.03	65.21

ing and SVMs to record deduplication and points out that domain-specific similarities can improve the quality of classifiers. [3, 2] rely on a theory for good edit distances developed by [1] to determine classifiers based on edit distances that are guaranteed to remain under a given classification error. Yet, to the best of our knowledge, REEDED is the first approach for the time-efficient execution of weighted edit distances.

8 Conclusion

In this paper we presented REEDED, an approach for the time-efficient comparison of sets using weighted distances. After presenting the intuitions behind our approach, we proved that it is both correct and complete. We compared our approach with an extension of PassJoin for weighted edit distances and showed that we are more than an order of magnitude faster on 4 different data sets. REEDED is the cornerstone of a larger research agenda. As it enable to now run weighted edit distances on large datasets within acceptable times, it is also the key to developing active learning systems for link discovery that do not only learn link specifications but also similarity measures directly out of the data. As shown in [21], this combination promises to outperform the state of the art, which has relied on standard measures so far. In future work, we will thus combine REEDED with specification learning approaches such as EAGLE [19] and RAVEN [18] and study the effect of weighted edit distances on these approaches.

References

1. Maria-Florina Balcan, Avrim Blum, and Nathan Srebro. Improved guarantees for learning via similarity functions. In *COLT*, pages 287–298, 2008.
2. Aurélien Bellet, Amaury Habrard, and Marc Sebban. Good edit similarity learning by loss minimization. *Machine Learning*, 89(1-2):5–35, 2012.
3. Aurlien Bellet, Amaury Habrard, and Marc Sebban. Learning good edit similarities with generalization guarantees. In *Proceedings of the ECML/PKDD 2011*, 2011.
4. Matthias Bernt, Alexander Donath, Frank Jühling, Fabian Externbrink, Catherine Florentz, Guido Fritzsch, Jörn Pütz, Martin Middendorf, and Peter F. Stadler.

MITOS: Improved *de novo* metazoan mitochondrial genome annotation. *Mol. Phylog. Evol.*, 2012.
5. Mikhail Bilenko and Raymond J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD*, pages 39–48, 2003.
6. Peter Christen. Febrl: a freely available record linkage system with a graphical user interface. In *Proceedings of the second Australasian workshop on Health data and knowledge management - Volume 80*, HDKM '08, pages 17–25, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.
7. Jianhua Feng, Jiannan Wang, and Guoliang Li. Trie-join: a trie-based method for efficient string similarity joins. *The VLDB Journal*, 21(4):437–461, August 2012.
8. Robert Isele, Anja Jentzsch, and Christian Bizer. Efficient Multidimensional Blocking for Link Discovery without losing Recall. In *WebDB*, 2011.
9. Mark D. Kernighan, Kenneth Ward Church, and William A. Gale. A spelling correction program based on a noisy channel model. In *COLING*, pages 205–210, 1990.
10. Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. *PVLDB*, 3(1):484–493, 2010.
11. S. Kurtz. Approximate string searching under weighted edit distance. In *Proc. WSP*, volume 96, pages 156–170. Citeseer, 1996.
12. V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR 163 (4)*, pages 845–848, 1965.
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. Axel-Cyrille Ngonga Ngomo. A Time-Efficient Hybrid Approach to Link Discovery. In *OM*, 2011.
15. Axel-Cyrille Ngonga Ngomo. Link Discovery with Guaranteed Reduction Ratio in Affine Spaces with Minkowski Measures. In *ISWC*, pages 378–393, 2012.
16. Axel-Cyrille Ngonga Ngomo. On Link Discovery using a Hybrid Approach. *Journal on Data Semantics*, 1:203 – 217, 2012.
17. 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.
18. Axel-Cyrille Ngonga Ngomo, Jens Lehmann, Sören Auer, and Konrad Höffner. RAVEN – Active Learning of Link Specifications. In *Sixth International Ontology Matching Workshop*, 2011.
19. Axel-Cyrille Ngonga Ngomo and Klaus Lyko. Eagle: Efficient active learning of link specifications using genetic programming. In *Proceedings of ESWC*, 2012.
20. E. S. Ristad and P. N. Yianilos. Learning string-edit distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(5):522–532, 1998.
21. Tommaso Soru and Axel-Cyrille Ngonga Ngomo. Active learning of domain-specific distances for link discovery. In *Proceedings of JIST*, 2012.
22. William E. Winkler. Overview of record linkage and current research directions. Technical report, BUREAU OF THE CENSUS, 2006.
23. 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.
24. Chuan Xiao, Wei Wang, Xuemin Lin, and Jeffrey Xu Yu. Efficient similarity joins for near duplicate detection. In *WWW*, pages 131–140, 2008.