A High-Performance Approach to String Similarity using Most Frequent K Characters

Andre Valdestilhas, Tommaso Soru, and Axel-Cyrille Ngonga Ngomo

AKSW/DICE, University of Leipzig, Germany
{valdestilhas,tsoru,ngonga}@informatik.uni-leipzig.de

Abstract. The amount of available data has been growing significantly over the last decades. Thus, linking entries across heterogeneous data sources such as databases or knowledge bases, becomes an increasingly difficult problem, in particular w.r.t. the runtime of these tasks. Consequently, it is of utmost importance to provide time-efficient approaches for similarity joins in the Web of Data. While a number of scalable approaches have been developed for various measures, the Most Frequent k Characters (MFKC) measure has not been tackled in previous works. We hence present a sequence of filters that allow discarding comparisons when executing bounded similarity computations without losing recall. Therewith, we can reduce the runtime of bounded similarity computations by approximately 70%. Our experiments with a single-threaded, a parallel and a GPU implementation of our filters suggest that our approach scales well even when dealing with millions of potential comparisons.

Keywords: Similarity Search; Blocking; String Matching

1 Introduction

The problem of managing heterogeneity at both the semantic and syntactic levels among various information resources [12,10] is one of the most difficult problems on the information age. This is substantiated by most of the database research self-assessment reports, which acknowledge that the hard question of semantic heterogeneity, that is of handling variations in meaning or ambiguity in entity interpretation, remains open [10]. In knowledge bases, Ontology Matching (OM) solutions address the semantic heterogeneity problem in two steps: (1) matching entities to determine an alignment, i.e., a set of correspondences, and (2) interpreting an alignment according to application needs, such as data translation or query answering. Record Linkage (RL) and, more recently, Link Discovery[1] (LD) solutions on the other hand aim to determine pairs of entries that abide by a given relation R. In both cases, string similarities are used to compute

[1] The expression "link discovery" in this paper means the discovery of typed relations that link instances from knowledge bases on the Web of Data. We never use it in the sense of graph theory.

candidates for alignments. In addition to being central to RL and LD, these similarities also play a key role in several other tasks such as data translation, ontology merging and navigation on the Web of Data [10,2].

One of the core tasks when developing time-efficient RL and LD solutions hence lies in the development of time-efficient string similarities. In this paper, we study the MFKC similarity function [8] and present an approach for improving the performance of similarity joins. To this end, we develop a series of filters which guarantee that particular pairs of resources do not abide by their respective similarity threshold by virtue of their properties.

The contributions of this paper are as follows:

1. We present two nested filters, (1) First Frequency Filter and (2) Hash Intersection filter, that allow to discard candidates before calculating the actual similarity value, thus giving a considerable performance gain.
2. We present the *k similarity filter* that allows detecting whether two strings s and t are similar in a fewer number of steps.
3. We evaluate our approach with respect to its runtime and its scalability with several threshold settings and dataset sizes.
4. We present several parallel implementations of our approach and show that they work well on problems where $|D_s \times D_t| \geq 10^5$ pairs.

The rest of the paper is structured as follows: In Section 2 related work is presented, where we focus on approaches that aim to improve the time-efficiency of the link discovery task. In Section 3, we present our nested filters, followed by the Section 4 with the Correctness and Completeness. Section 5 with the evaluation. In Section 6, we conclude.

2 State of the Art and related work

Our approach can be considered an extension of the state-of-the-art algorithm introduced in [8], which describes a string-based distance function (SDF) based on string hashing [9,7]. The naive approach of MFKC [8] is a metric for string comparison built on a hash function, which gets a string and outputs the most frequent two characters with their frequencies. This algorithm was used for text mining operations. The approach can be divided into two parts: (1) The hashing function is applied to both input strings, where the output is a string that contains the two most frequent characters; the first and third elements keep the characters and second and fourth elements keep the frequency of these characters. (2) The hashes are compared, where will return a real number between 0 and lim. By default $lim = 10$, since the probability of having ten occurrences of the two most frequent characters in common between two strings is low. If the output of the function is 10, this case indicates that there is no common character and any value below 10 means there are some common characters shared among the strings.

Our work is similar to the one presented in [13], which features a parallel processing framework for string similarity using filters to avoid unnecessary comparisons. Among the several types of string similarity, emerging works have been

done for measures such as Levenshtein-distance [6], which is a string distance function that calculates the minimum number of edit operations (i.e., delete, insert or update) to transform the first into the second string. The Jaccard Index [5], also called Jaccard coefficient, works on the bitwise operators, where the strings are treated at bit level. REEDED [11] was the first approach for the time-efficient execution of weighted edit distances.

3 Approach

Let us call $NaiveMFKC$ the function which computes the MFKC algorithm as described in [8]. Such function works with three parameters, i.e. two strings s and t and an integer lim and returns the sum of frequencies, where $\mathrm{f}(c_i, s)$ is a function that returns the frequency of the character c_i in the string s and $s \supseteq \{c_1, ..., c_n\}$, i.e $\mathrm{f}(a, "andrea") = 2$, because the character a has been found twice and the hash functions $h(s)$ and $h(t)$ containing the characters and their frequencies. The output of function is always positive, as shown in Equation (1).

$$NaiveMFKC(s, t, lim) = lim - \sum_{c_i \in h(s) \cap h(t)}^{2} f(c_i, s) + f(c_i, t) \tag{1}$$

Our work aims to reduce the runtime of computation of the MFKC similarity function. Here, we use a sequence of filters, which allow discarding similarity computations and imply in a reduction of runtime. As input, the algorithm receives datasets D_s and D_t, an integer number representing the k most frequent characters and a threshold $\theta \in [0, 1]$. The similarity score of the pair of strings from the Cartesian product from D_s and D_t must have a score greater or equal the threshold θ to be considered a good pair, i.e. for a given threshold θ, if the similarity function has a pair of strings with similarity score less than the threshold, $\sigma(s, t) < \theta$, we can discard the computation of the MFKC score for this pair. Our final result is a set which contains the pairs having similarity score greater than or equal to the threshold, i.e. $\sigma(s, t) \geq \theta$.

Our work studies the following problem: Given a threshold $\theta \in [0, 1]$ and two sets of strings D_s and D_t, compute the set $M' = \{(s, t, \sigma(s, t)) \in D_s \times D_t \times \mathbb{R}^+ : \sigma(s, t) \geq \theta\}$. 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 that targets the MFKC algorithm. Equation (2) shows our definition for the string similarity function σ for the MFKC.

$$\sigma(s, t) = \frac{\sum_{c_i \in h(s,k) \cap h(t,k)} \mathrm{f}(c_i, s) + \mathrm{f}(c_i, t)}{|s| + |t|} \tag{2}$$

where s and t are strings, such that $s, t \in \sum^*$, $\mathrm{f}(c_i, s)$ is a function that returns the frequency of the character c_i in the string s, where $s \supseteq \{c_1, ..., c_n\}$, k represents the limitation of the elements that belongs to the hashes; set $h(s, k) \cap h(t, k)$

means the intersection between the keys of hashes $h(s,k)$ and $h(t,k)$ (i.e., the most frequent K characters). We expect two steps to obtain the similarity score:

1. Firstly, we transform the strings s and t in two hashes using Most Frequent Character Hashing [8], according to the following example with $k = 3$:
   $s = aabbbcc \rightarrow \mathrm{h}(s,k) = \{b = 3, a = 2, c = 2\}$
   $t = bbccddee \rightarrow \mathrm{h}(t,k) = \{b = 2, c = 2, d = 2\}$
2. We calculate the sum of the character frequencies of matching characters on the hashes $h(s,k)$ and $h(t,k)$, then, we normalize dividing by the sum of the length of $|s|$ and $|t|$ resulting in a similarity score from 0 to 1 according to the Equation (3) and the resulting score should be greater or equals the threshold θ.

$$\sigma(s,t,k,\theta) = \frac{\sum_{c_i \in \mathrm{h}(s,k) \cap \mathrm{h}(t,k)} \mathrm{f}(c_i,s) + \mathrm{f}(c_i,t)}{|s| + |t|} \geq \theta \tag{3}$$

3.1 Improving the Runtime

In this section, the runtime of MFKC defined in Equation (2) is improved using filters where $\mathcal{N}$ is the output of first frequency filter, $\mathcal{L}$ is the output of hash intersection filter and $\mathcal{A}$ represents the output of the k similarity filter.

First Frequency Filter As specified in the definition of MFKC [8] this filter assumes that the hashes are already sorted in an descending way according to the frequencies of characters, therefore the first element of each hash has the highest frequency.

Theorem 1. *Showing that:*

$$\sigma(s,t) = \frac{\sum_{c_i \in \mathrm{h}(s,k) \cap \mathrm{h}(t,k)} \mathrm{f}(c_i,s) + \mathrm{f}(c_i,t)}{|s| + |t|} \leq \frac{h_1(s,k)k + |t|}{|s| + |t|} \tag{4}$$

implies that $\sigma(s,t) < \theta$.

Proof (Theorem 1). Let the intersection between hashes $h(t,k)$ and $h(s,k)$ be a set of characters from c_1 to c_n, such that Equation (5):

$$h(t,k) \cap h(s,k) = \{c_1, ..., c_n\} \tag{5}$$

According to the definition of the frequencies $\mathrm{f}(c_i,t)$ we have Equation (6):

$$t \supseteq \{c_1, ..., c_1, ..., c_n, ...c_n\} \tag{6}$$

where each c_i appears $\mathrm{f}(c_i,t)$ times, therefore:

$$\mathrm{f}(c_1,t) + ... + \mathrm{f}(c_n,t) \leq |t| \tag{7}$$

Also, as $n \leq k$, because $t \supseteq \{c_1, ..., c_n\}$, and $\mathrm{f}(c_i,s) \leq h_1(s,k)\ \forall_{i=1,...,n}$, then:

$$\mathrm{f}(c_1,s) + ... + \mathrm{f}(c_n,s) \leq h_1(s,k) + ... + h_1(s,k) = n(k) \leq k(h_1(s,k)) \tag{8}$$

Therefore, from Equation (7) and Equation (8), we obtain the Equation (9):

$$\frac{\sum_{c_i \in \mathrm{h}(s,k) \cap \mathrm{h}(t,k)} \mathrm{f}(c_i, s) + \mathrm{f}(c_i, t)}{|s| + |t|} = \frac{\sum_{i=1}^{n} \mathrm{f}(c_i, t) + \sum_{i=1}^{n} \mathrm{f}(c_i, s)}{|s| + |t|} \leq \frac{h_1(s,k)k + |t|}{|s| + |t|} \tag{9}$$

Consequently, the rule which the filter relies on is the following.

$$\langle s, t \rangle \notin \mathcal{N} \Rightarrow \langle s, t \rangle \notin D_s \times D_t \wedge \frac{h_1(s,k)k + |t|}{|s| + |t|} \leq \theta \tag{10}$$

Hash Intersection Filter In this filter, we check if the intersection between two hashes is an empty set, then the MFKC, represented by σ, will return a similarity score of 0 and we can avoid the computation of similarity in this case. Consequently, the rule which the filter relies on is the following.

$$\langle s, t \rangle \in \mathcal{L} \Rightarrow \langle s, t \rangle \in D_s \times D_t \wedge |h(s) \cap h(t)| > 0 \tag{11}$$

we also can say that the Equation (12) represents a valid implication.

$$h(s) \cap h(t) = \emptyset \Rightarrow \sigma(s, t) = 0 \tag{12}$$

The Equation (12) means that if the intersection between $h(s, k)$ and $h(t, k)$ is a empty set, this implies that the similarity score will be 0. That means there is no character matching, then there is no need to compute the similarity for this pair of strings.

K Similarity filter For all the pairs left, the similarity score among them is calculated. After that, the third filter selects the pairs whose similarity score is greater or equal than a threshold θ.

$$\langle s, t \rangle \in \mathcal{A} \Leftrightarrow \langle s, t \rangle \in \mathcal{N} \wedge \sigma(s, t) \geq \theta \tag{13}$$

This filter provides a validation and we show that the score of previous k similarity is always lower than the next k, according to the Equation (14) and in some cases when the similarity score is been reached before compute all elements $\in \mathrm{h}(s, k) \cap \mathrm{h}(t, k)$, thus saving computation in these cases.

Here k is also used as a index of similarity function $\sigma_k(s, t)$ in order to get the similarity of all cases of k, from 1 to k, also to show the monotonicity.

Therefore we can say that the computation of similarity score occurs until $\sigma_k(s, t) \geq \theta$.

We will demonstrate that the similarity score of previous k similarity is always lower than the next k similarity, for all $k \in \mathbb{Z}^* : k \leq |s \cap t|$.

$$\sigma_{k+1}(s, t) \geq \sigma_k(s, t) \tag{14}$$

We rewrote the equation for the first iteration, according to Equation (15)

$$\sigma_k(s,t) = \frac{\sum_{c_i \in \mathrm{h}(s,k) \cap \mathrm{h}(t,k), i=1}^{k} \mathrm{f}(c_i, s) + \mathrm{f}(c_i, t)}{|s| + |t|} \tag{15}$$

Let $k \in \mathbb{Z}_+$ be given and suppose the equation Equation (14) is true for $k+1$.

$$\sum_{c_i \in \mathrm{h}(s,k) \cap \mathrm{h}(t,k)}^{k} \big[\mathrm{f}(c_i, s) + \mathrm{f}(c_i, t)\big] + \mathrm{f}(c_{k+1}, s) + \mathrm{f}(c_{k+1}, t) \geq \sum_{c_i \in \mathrm{h}(s,k) \cap \mathrm{h}(t,k)}^{k} \mathrm{f}(c_i, s) + \mathrm{f}(c_i, t) \tag{16}$$

Therefore, we can notice that the sum of frequencies will be always greater or equal 0, according to $\mathrm{f}(c_{k+1}, s) + \mathrm{f}(c_{k+1}, t) \geq 0$. Thus, Equation (14) holds true.

Filter sequence The sequence of the filters occurs basically in 4 steps, (1) We starting to make the Cartesian product with the pairs of strings from the datasets D_s and D_t, (2) Discarding pairs using the $First\ Frequency\ Filter(\mathcal{N})$, (3) Discarding pairs where there is no matching characters with the $Hash\ Intersection\ filter(\mathcal{L})$ and (4) With the Most Frequent Character Filter ($\mathcal{A}$) we will process only similarities greater or equal the threshold θ, if the similarity function $\sigma(s,t) \geq \theta$ in the first k characters we can stop the computation of the similarity of this pair, saving computation and add to our dataset with resulting pairs Dr, also shown in the Algorithm 1.

Algorithm 1 MFKC Similarity Joins

$GoodPairs = \{s_1; t_1, ..., s_n; t_n\} | \langle s, t \in \sum^* \rangle$
$hs = \{e_1, e_2, ..., e_n\} | e_i = \langle c, f(c, s) \rangle$
$ht = \{e_1, e_2, ..., e_n\} | e_i = \langle c, f(c, t) \rangle$
$i, freq \in \mathbb{N}^*$
procedure MFKC(D_s, D_t, θ, k)
  for all $s \in D_s$ **do (in parallel)**
    $hs = h(s, k)$
    for all $t \in D_t$ **do (in parallel)**
      $ht = h(t, k)$
      if $\frac{hs_1(k)+|t|}{|s|+|t|} < \theta$ **then**
        $Next\ t \in D_t$
      end if
      if $|hs \cap ht| = 0$ **then**
        $Next\ t \in D_t$
      end if
      for all $c_{freq} \in hs \cap ht$ **do**
        if $i \geq k$ **then**
          $Next\ t \in D_t$
        end if
        $freq = freq + c_{freq}$
        $\sigma_i = \frac{freq}{|s|+|t|}$
        if $\sigma_i \geq \theta$ **then**
          $GoodPairs.add(s, t)$
          $Next\ t \in D_t$
        end if
        $i = i + 1$
      end for
    end for
  end for
  return $GoodPairs$
end procedure

4 Correctness and Completeness

In this section, we prove formally that our MFKC is both correct and complete.

- We say that an approach is correct if the output O it returns is such that $O \subseteq R(D_s, D_t, \sigma, \theta)$.
- Approaches are said to be complete if their output O is a superset of $R(D_s, D_t, \sigma, \theta)$, i.e., $O \supseteq R(D_s, D_t, \sigma, \theta)$.

Our MFKC consists of three nested filters, each of which creates a subset of pairs, i.e. $\mathcal{A} \subseteq \mathcal{L} \subseteq \mathcal{N} \subseteq D_s \times D_t$. For the purpose of clearness, we name each filtering rule:

$$R_1 \triangleq \frac{h_1(s,k)k + |t|}{|s| + |t|} < \theta$$

$$R_2 \triangleq |h(s,k) \cap h(t,k)| \neq 0$$

$$R_3 \triangleq \sigma(s,t) \geq \theta$$

Each subset of our MFKC can be redefined as $\mathcal{N} = \{\langle s,t\rangle \notin D_s \times D_t : R_1$, $\mathcal{L} = \{\langle s,t\rangle \in D_s \times D_t : R_1 \wedge R_2$, and $\mathcal{A} = \{\langle s,t\rangle \in D_s \times D_t : R_1 \wedge R_2 \wedge R_3$. We then introduce $\mathcal{A}^*$ as the set of pairs whose similarity score is more or equal than the threshold θ.

$$\mathcal{A}^* = \{\langle s,t\rangle \in D_s \times D_t : \sigma(s,t) \geq \theta\} = \{\langle s,t\rangle \in D_s \times D_t : R_3\} \tag{17}$$

Theorem 2. *Our MFKC filtering algorithm is correct and complete.*

Proof (Theorem 2). Proving Theorem 2 is equivalent to showing that $\mathcal{A} = \mathcal{A}^*$. Let us consider all the pairs in A. While our MFKC's correctness follows directly from the definition of $\mathcal{A}$, it is complete iff none of pairs discarded by the filters actually belongs to A^*. Assuming that the hashes are sorted in a descending way according the frequencies of the characters, therefore the first element of each hash has the highest frequency. Therefore, once we have

$$\frac{h_1(s,k)k + |t|}{|s| + |t|} < \theta,$$

the pair of strings s and t can be discarded without calculating the entire similarity. When rule R_3 applies, we have $\sigma(s,t) < \theta$, which leads to $R_3 \Rightarrow R_1$. Thus, set $\mathcal{A}$ can be rewritten as:

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

We are given two strings s and t and the respective hashes $h(s,k)$ and $h(t,k)$, the intersection between the characters of these two hashes is a empty set. Therefore, there is no character matching, which implies that s and t cannot be considered to have a similarity score greater than or equal to threshold θ:

$$\mathrm{h}(s,k) \cap \mathrm{h}(t,k) = \emptyset \Rightarrow \sigma(s,t) = 0$$

When the rule R_3 applies, we have $\sigma(s,t) < \theta$, which leads to $R_3 \Rightarrow R_2$. Thus, set $\mathcal{A}$ can be rewritten as:

$$\mathcal{A} = \{\langle s,t\rangle \in D_s \times D_t : R_3\} \tag{19}$$

which is the definition of $\mathcal{A}^*$ in Equation (17). Therefore, $\mathcal{A} = \mathcal{A}^*$.

Time complexity In order to calculate the time complexity of our MFKC, firstly we considered the most frequent K characters from a string. The first step is to sort the string lexically. Then, we can reach a linear complexity after this sort, because the input with highest occurrences can be achieved with a linear time complexity. The first string can be sorted in $O(nlogn)$ and second string in $O(mlogm)$ times, as some classical sorting algorithms such as merge sort [3] and quick sort [4] that work in $O(nlogn)$ complexity. Thus, the total complexity is $O(nlogn) + O(mlogm)$, resulting in $O(nlogn)$ as upper bound in the worst case.

5 Evaluation

The aim of our evaluation is to show that our work outperforms the naive approach and the parallel implementation has a performance gain in large datasets with size greater than 10^5 pairs. A considerable number of pairs reach the threshold θ before reaching the last k most frequent character, that also is a demonstration about how much computation was avoided. Instead of all k's we just need $k - n$ where n is the n^{th} most frequent character necessary to reach the threshold θ. An example to show the efficiency of each filter can be found at Figures 1 and 1(a), where 10,273,950 comparisons from DBpedia+LinkedGeoData were performed and Performance Gain $(PG) = Recall(\mathcal{N}) + Recall(\mathcal{L})$. The recall[2] can be seen in Figure 2(c). This evaluation has the intention to show results of experiments on data from DBpedia[3] and LinkedGeoData[4]. We considered pairs of labels in order to do the evaluation. We have two motivations to chose these datasets: (1) they have been widely used in experiments pertaining to Link Discovery (2) the distributions of string sizes between these datasets are significantly different [1]. All runtime and scalability experiments were performed on a Intel Core i7 machine with 8GB RAM, a video card NVIDIA NVS4200 and running Ms Windows 10.

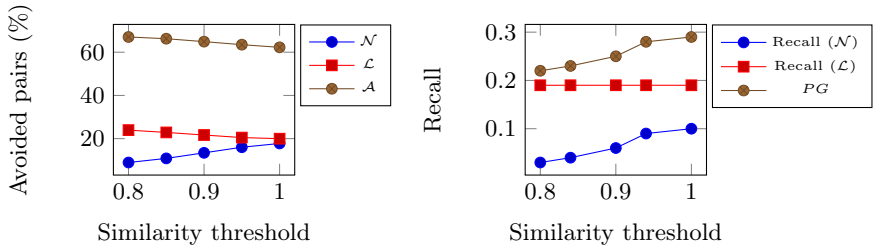
5.1 Parallel implementation

Our algorithm contains parallel code snippets with which we perform a load and balance of the data among CPU/GPU cores when available. To illustrate this

[2] Depicting DBPedia-Yago results. The YAGO was added to bring a reinforcement to our evaluations, due to the fact of this dataset have been widely used in experiments pertaining to Link Discovery. We also considered evaluations on recall, precision, and f-score.

[3] `http://wiki.dbpedia.org/`

[4] `http://linkedgeodata.org/`



(a) Avoided pairs(10,273,950). (b) Recall.

Fig. 1. Avoided pairs and recall.

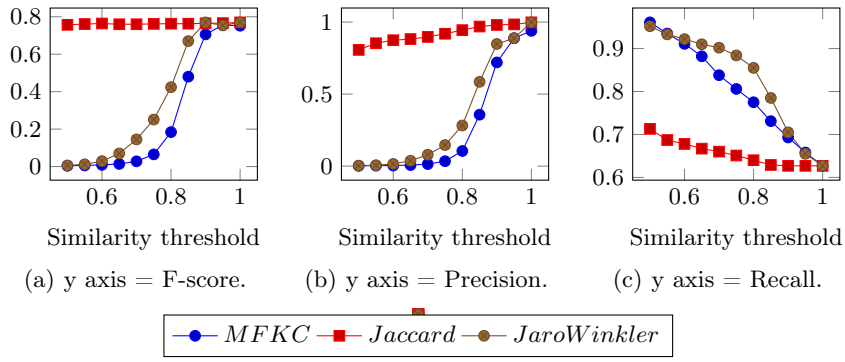
part of our idea, we can state: Given a two datasets S, T, that contains all the strings to be compared. Thus, make a Cartesian product of the strings $S \times T$, where each pair is the processed separately in threads that are spread among CPU/GPU cores. Thus, we process the each comparison in parallel. The parallel implementation works better in large datasets with size more than 10^5, that was more than one time faster than the approach without parallelism and two times faster than the naive approach as shown in Figures 3(a) to 3(c).

5.2 Runtime Evaluation

The evaluation in Figures 3(a) and 3(b) shows that all filter setup outperform the naive approach, and the parallel approach does not suffer significant changes related to the runtime according to the size of the dataset, as show Figure 3(c). The experiments related to the variance of k, also were considered, as show in Figure 3(d), the runtime varies according the size of k, indicating the influence of k with values from 1 to 120 with $1,001,642$ comparisons. The performance (run-time) was improved as shown in Figures 3(a) and 3(b) and according the recall with a performance gain of 26.07% as shown in Figure 1(a). The time complexity is based on two sort process O(n log n) + O(m log m) resulting in O(n log n) as a upper bound in the worst case.

5.3 Scalability Evaluation

In the experiments (see Figures 3(c), 3(e) and 3(f)), we looked at the growth of the runtime of our approach on datasets of growing sizes. The results show that the combination of filters ($\mathcal{N}+\mathcal{L}+\mathcal{A}$) is the best option for datasets of large sizes. This result holds on both DBpedia and LinkedGeoData, so our approach can be used on large datasets and achieves acceptable run-times. We also can realize the quantity of avoided pairs in each combination of filters in Figure 1, that consequently brings a performance gain. We looked at experiments with runtime behavior on a large dataset with more than 10^6 labels as shown in Figure 3(b). The results suggest that the runtime decreases according to the threshold θ



(a) y axis = F-score. (b) y axis = Precision. (c) y axis = Recall.

Fig. 2. Precision, Recall and F-Measure.

increment. Thus, one more point showing that our approach is useful on large datasets, where can be used with high threshold values for link discovery area. About our parallel implementation, Figure 3(c) shows that our GPU parallel implementation works better on large datasets with size greater than 10^5.

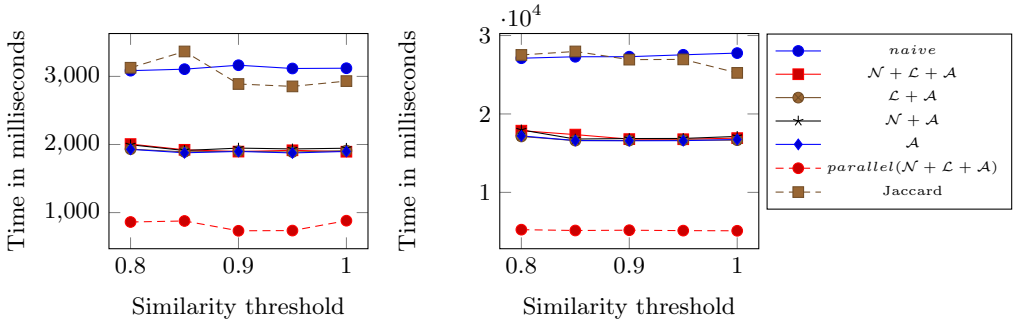
5.4 Comparison with existing approaches

Our work overcomes the naive approach [8], thus, in order to show some important points we compare our work not only with the state of the art, but with popular algorithms such as Jaccard Index [5]. As shown in Figures 3(c), 3(e) and 3(f), our approach outperforms not only the naive approach, but also Jaccard Index. We show that the threshold θ and k have a significant influence related to the runtime. The naive approach present some points to consider, among them, even if the naive approach states that they did experiments with $k = 7$, the naive algorithm was designed for only $k = 2$, there are some cases where $k = 2$ is not enough to get the similarity level expected, i.e. $s = mystring1$ and $t = mystring2$ limiting $k = 2$, we will have $\sigma_2(s, t) = 0.2$, showing that the similarity is very low, but when $k = 8$, the similarity is $\sigma_8(s, t) = 0.8$ showing that sometimes we can lose a good similarity case limiting $k = 2$. Our work fix all these problems and also has a better runtime, as show Figure 3(a), Figure 3(b) and Figure 3(c).

An experiment with labels from DBpedia and Yago[5] shows that the f-score indicates a significant potential to be used with success as a string similarity comparing with Jaccard Index and Jaro Winkler, as Figures 2(a) to 2(c) shows.

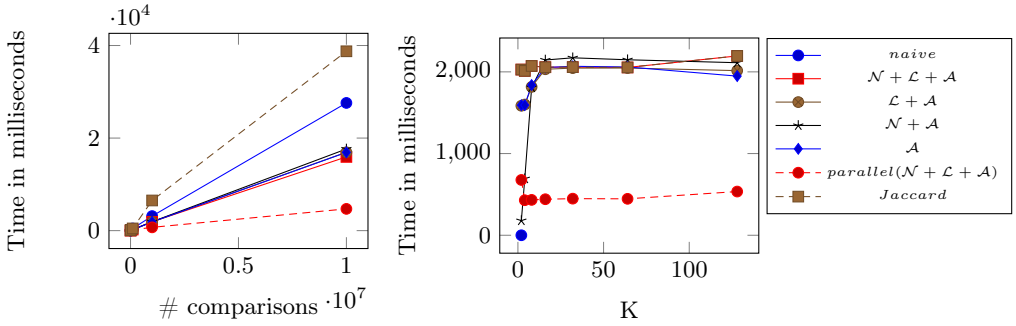
To summarize the key features that makes our approach outperform the naive approach are the following: We use more than two K most frequent characters in our evaluation, our run-time for more than 10^7 comparisons is shorter (27,594 against 16,916) milliseconds, we do have a similarity threshold, allowing us to

[5] `http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/`



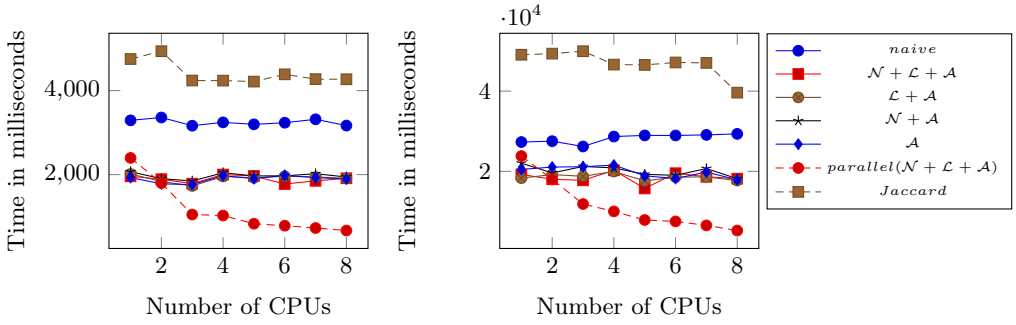
(a) Runtime 1,001,642 comparisons.

(b) Runtime 10,273,950 comparisons.



(c) The parallel approach improves the performance for more than 10^5 *comparisons*.

(d) Runtime k most frequent characters, with values of k from 1 to 120, over $1,001,642$ comparisons, $\theta = 0.95$.



(e) CPU Speedup of algorithm $(1,001,642$ comparisons).

(f) CPU Speedup of algorithm$(10,273,950$ comparisons).

Fig. 3. Run-time experiments results.

discard comparisons avoiding extra processing, and a parallel implementation, making our approach scalable. Jaccard does not show significant changes varying the threshold, MFKC and Jaro Winkler present a very similar increase of the f-score varying the threshold.

6 Conclusion and Future work

We presented an approach to reduce the computation runtime of similarity joins using the Most Frequent k Characters algorithm with a sequence of filters that allow discarding pairs before computing their actual similarity, thus reducing the runtime of computation. We proved that our approach is both correct and complete. The evaluation shows that all filter setup outperform the naive approach. Our parallel implementation works better in larger datasets with size greater than 10^5 pairs. It is also the key to developing systems for Record Linkage and Link Discovery in knowledge bases. As future work, we plan to integrate it in link discovery applications for the validation of equivalence links. The source code is free and available online[6]. Acknowledgments available on footnotes[7].

References

1. K. Dreßler and A.-C. N. Ngomo. On the efficient execution of bounded jaro-winkler distances. 2014.
2. J. Euzenat, P. Shvaiko, et al. *Ontology matching*, volume 333. Springer, 2007.
3. H. H. Goldstine and J. von Neumann. *Planning and coding of problems for an electronic computing instrument*. Institute for Advanced Study, 1948.
4. C. A. Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.
5. P. Jaccard. The distribution of the flora in the alpine zone. *New phytologist*, 11(2):37–50, 1912.
6. V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.
7. R. Rivest. The MD5 message-digest algorithm. 1992.
8. S. E. Seker, O. Altun, U. Ayan, and C. Mert. A novel string distance function based on most frequent k characters. *IJMLC*, 4(2):177–182, 2014.
9. S. E. Seker and C. Mert. A novel feature hashing for text mining. *Journal of Technical Science And Technologies*, 2(1):37–40, 2013.
10. P. Shvaiko and J. Euzenat. Ontology matching: State of the art and future challenges. *TKDE*, 25(1):158–176, 2013.
11. T. Soru and A.-C. Ngonga Ngomo. Rapid execution of weighted edit distances. In *Proceedings of the Ontology Matching Workshop*, 2013.
12. A. Valdestilhas, N. Arndt, and D. Kontokostas. DBpediaSameAs: An approach to tackle heterogeneity in dbpedia identifiers.
13. C. Yan, X. Zhao, Q. Zhang, and Y. Huang. Efficient string similarity join in multi-core and distributed systems. *PLOS ONE*, 12(3):1–16, 03 2017.

[6] `https://github.com/firmao/StringSimilarity/`

[7] Acknowledgments to *CNPq* Brazil under grants No. 201536/2014-5 and H2020 projects SLIPO (GA no. 731581) and HOBBIT (GA no. 688227) as well as the DFG project LinkingLOD (project no. NG 105/3-2), the BMWI Projects SAKE (project no. 01MD15006E) and GEISER (project no. 01MD16014).