Efficient Approximate Entity Matching Using Jaro-Winkler Distance

Yaoshu Wang(✉), Jianbin Qin, and Wei Wang

School of Computer Science and Engineering,
Univeristy of New South Wales, Sydney, Australia
{yaoshuw,jqin,weiw}@cse.unsw.edu.au

Abstract. Jaro-Winkler distance is a measurement to measure the similarity between two strings. Since Jaro-Winkler distance performs well in matching personal and entity names, it is widely used in the areas of record linkage, entity linking, information extraction. Given a query string q, Jaro-Winkler distance similarity search finds all strings in a dataset D whose Jaro-Winkler distance similarity with q is no more than a given threshold τ. With the growth of the dataset size, to efficiently perform Jaro-Winkler distance similarity search becomes challenge problem. In this paper, we propose an index-based method that relies on a filter-and-verify framework to support efficient Jaro-Winkler distance similarity search on a large dataset. We leverage e-variants methods to build the index structure and pigeonhole principle to perform the search. The experiment results clearly demonstrate the efficiency of our methods.

Keywords: Pigeonhole principle · Jaro-Winkler distance · e-variants

1 Introduction

Entity linking (EL) aims to link all mentions to potential entities in a knowledge base (KB) after named entity recognition (NER). However, large amounts of ment ions and entities become the obstacle of the efficient issue. At present, EL methods find mentions and entity candidates using a standard lexicon. Levenshtein distance is a popular distance function to filter strings, but it sometimes does not work on short strings, especially named entities. According to [4], Jaro-Winkler distance (d_{JW}) is an efficient distance function to measure name-matching task.

Mentions and entities are represented as (short) strings in lexicons. Our problem can be modeled as an approximate entity matching problem: Given a lexicon of entity strings and a list of mention strings, we aim to find all potential entity strings in the lexicon for each mention string. In our knowledge, [5] proposed the only approximate entity matching methods by using Jaro-Winkler distance. The idea is to use a trie structure to filter dissimilar strings and prune nodes in trie. The limitation in the work [5] is that it only considers the common characters of strings, and should traverse large amounts of nodes in the trie structure to reach the lower bound of Jaro-Winkler distance.

A. Bouguettaya et al. (Eds.): WISE 2017, Part I, LNCS 10569, pp. 231–239, 2017.
DOI: 10.1007/978-3-319-68783-4_16

In this paper, we solve the approximate entity matching in a new perspective. Our contribution can be summarized as follows.

- We proposed a new lower bound of Jaro-Winkler distance. Instead of indexing single characters, we combined characters into signatures and check the number of common signatures between strings.
- We designed a new index structure and efficient query processing algorithm to support Jaro-Winkler distance.
- We have conducted comprehensive experiments using several named entity datasets. The proposed method has been shown to achieve the best performance among all other ones.

2 Related Work

Jaro-Winkler distance is a general similarity metric used in entity linking [8,11], record linkage [2–4,6], and data cleaning [12]. This paper focuses on how to use Jaro-Winkler distance to performance efficient string similarity search.

For similarity metrics, such as Levenshtein distance, Jaccard similarity, cosine similarity, index-based methods (incl. inverted index based methods and trie based methods) are the most efficient methods of string similarity search so far.

Inverted Index Based Methods. Many state-of-the-art algorithms [13,16,17] adopted inverted index structure to performance query processing. PassJoin [10] partitioned strings according to pigeonhole principle into the set of substrings and indexed these substrings. PPJoin+ [17] utilized tokens as signatures to construct inverted index and introduced positional filtering and suffix filtering to prune false positives. EdJoin [16] and qGramChunk [13] extracted q-grams or q-chunks as signatures to build index, and they designed prefix filterings with shorter prefix size to prune strings. NGPP [15] combined pigeonhole principle and e-variants to generate variant strings, and indexed these strings in the inverted index to improve query processing. Inverted index based methods are very efficient in long strings.

Trie Based Methods. Trie based methods [5,7] adopted the trie structure to perform string similarity search and join. TrieJoin [7] utilized the trie structure to deal with edit distance metric. By adding some pruning strategies, TrieJoin could terminate its method early. LIMES [5] worked in Jaro-Winkler distance metric and counted the common characters between two strings by traversing the trie structure. Trie based methods adapt to short strings, such as short titles, person names and so on.

3 Problem Definition

Definition 1. *Given a list of query mentions $\mathcal{M}$ and a lexicon $\mathcal{D}$ of entities, the task of approximate entity matching with the Jaro-Winkler distance threshold τ_{JW} is to find all mention-and-entity pairs $< q_m, s_e >$ $(q_m \in M,\ s_e \in \mathcal{D})$ that $d_{JW}(q_m, s_e) \geq \tau_{JW}$.*

Notations. We denote the lexicon of entities as $\mathcal{D}$ and collection of mentions as $\mathcal{M}$. We use l_{min} and l_{max} to represent the minimal and maximal lengths of strings in $\mathcal{D}$. $\mathcal{D}[i]$ is represented as the i-th entity string in $\mathcal{D}$. τ_J is the threshold of Jaro distance, and τ_{JW} is the threshold of Jaro-Winkler distance.

4 Lower Bound of Jaro-Winkler Distance

In this section, we first explore the lower bound of Jaro distance and then add the constraint of Winkler distance.

Lower Bound of Jaro Distance. According to Jaro Distance d_J(Eq. 1), given strings s_1 and s_2, we know $0 \leq t \leq \frac{m}{2}$ according to [4]. Now we set $t = 0$ and design a lower bound of common characters of Jaro distance.

$$d_J = \frac{1}{3}(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}) \leq \frac{1}{3}(\frac{m}{|s_1|} + \frac{m}{|s_2|} + 1) = d_J^{lb} \tag{1}$$

If $d_J^{lb}(s_1, s_2) \geq \tau_J$, we have the number of common characters $m \geq T = \frac{(3\tau_j - 1)|s_1||s_2|}{|s_1|+|s_2|}$, which is the lower bound of strings s_1 and s_2. After calculating T, we utilize pigeonhole principle and e-variants [15] to filter strings. Due to no order of matching characters of Jaro distance, we sort all characters according to a specific order L (e.g. the alphabetical order). The construction of L is ignored due to limited space. We calculate the lower bound between s_1 and s_2 as follow.

1. We calculate T described above.
2. We utilize pigeonhole principle to separately partition s_1 and s_2 into k_1 and k_2 parts $P_1 = \{s_1^1, s_1^2, \ldots, s_1^{k_1}\}$ and $P_2 = \{s_2^1, s_2^2, \ldots, s_2^{k_2}\}$. In detail, we design special splitting characters SC to generate partitions. SC is a set of special characters used to partition strings according to L. s is sorted according to L, and then partitioned according to SC as several parts.
3. We separately allocate errors for P_1 and P_2 as $E_1 = \{e_1^1, e_1^2, \ldots, e_1^{k_1}\}$ and $E_2 = \{e_2^1, e_2^2, \ldots, e_2^{k_2}\}$, where $\sum_{i=1}^{k_1} e_1^i = |s_1| - T$ and $\sum_{i=1}^{k_2} e_2^i = |s_2| - T$.
4. We generate all e_1^i-variants for s_1^i of s_1 and e_2^j-variants for s_2^j of s_2 ($1 \leq i \leq k_1$, $1 \leq j \leq k_2$). $Var(s_1, T)$ and $Var(s_2, T)$ are denoted as sets of variants of all partitions of s_1 and s_2. For example, $Var(s_1, T) = \bigcup_{i=1}^{k_1} e_1^i\text{-}var(s_1^i)$.

Lemma 1. ***New lower bound of Jaro distance.*** *For strings s_1 and s_2, and the minimal common characters T, $Var(s_1, T) \bigcap Var(s_2, T) \neq \emptyset$.*

Lower Bound of Jaro-Winkler Distance. Given $d_{JW} \geq \tau_{JW}$ and the definition of d_{JW}, we can deduce the threshold τ_J of Jaro distance $d_J \geq \tau_J = \frac{\tau_{JW} - P' \cdot l}{1 - P' \cdot l}$. Given s_1 and s_2, we calculate P', and then calculate τ_J. Then we use the above lower bound to check common variants of s_1 and s_2.

5 Index Construction for Jaro-Winkler Distance

Given a collection of entities $\mathcal{D}$, the character sequence L, the set of splitting characters SC, and the minimum Jaro threshold τ_J^{min}, we first construct the index I_J of Jaro distance. L is partitioned into $L_1, L_2, \ldots L_K$ parts according to SC.

First, we sort each entity $s_i \in \mathcal{D}$ $(1 \leq i \leq |\mathcal{D}|)$ in the special order of L. If same characters exist in one string, we consider them as different characters. Then according to SC, we partition s_i into k_i non-overlapping substrings $P_i = \{s_i^1, s_i^2, \ldots, s_i^{k_i}\}$. Next, given the minimal threshold τ_J^{min}, we calculate the least common number of characters T_i^{min} for s_i (i.e., maximal error). After partition s_i into k_i partitions, we evenly allocate errors for k_i partitions as $E_i^{max} = \{e_i^1, e_i^2, \ldots, e_i^{k_i}\}$. Finally, we generate $Var(s_i, T_i^{min})$ for s_i where $Var(s_i, T_i^{min}) = \bigcup_{j=1}^{k_i} e_i^j\_var(s_i^j)$, and insert all elements $w \in Var(s_i, T_i^{min})$ as keys and tuples (i, s_i, k_i, e) as values into the inverted index, where i is the entity ID of s_i, and e is the number of deleted characters. After inserting all entities in I_J, we sort elements (entities) in lists according to the decreasing order of $|s_i| - e \cdot k_i$. The reason is to apply early termination in query processing step (See Sect. 6).

Winkler distance only considers the first four prefix characters of strings. We construct a 4-level trie structure to index four prefix characters of strings in $\mathcal{D}$. We enumerate all four prefix characters in lexicon $\mathcal{D}$, and insert them into I_{JW}. According to the definition of Jaro-Winkler distance, if $d_{JW}(s_1, s_2) \geq \tau_{JW}$ we can calculate the threshold of Jaro distance at i-th level as $\tau_J^i = \frac{\tau_{JW} - i \cdot l}{1 - i \cdot l}$. For example, the 2nd level of I_{JW} means two strings have 2 common prefix characters, and $\tau_J^2 = \frac{\tau_{JW} - 2l}{1 - 2l}$. Thus i-level node $(0 \leq i \leq 4)$ in the index I_{JW} denotes the index of Jaro distance I_J with threshold τ_J^i.

6 Online Query Processing

6.1 Query Processing of Jaro Distance

In this section, we solve the problem of online query processing of Jaro distance. Given a query $q \in \mathcal{M}$, and the threshold τ_J of Jaro distance, we retrieve all strings $s \in \mathcal{D}$ that $d_J(q, s) \geq \tau_J$.

For a query q, we sort and partition q into k_q partitions $P_q = \{q^1, q^2, \ldots, q^{k_q}\}$ according to L and SC. Considering l_{min}, we can get the minimal common characters between q and entities in $\mathcal{D}$ as $T_{min} = \frac{(3\tau_J - 1)|q| \cdot l_{min}}{|q| + l_{min}}$. Then we generate error allocation strategy as $E_{min} = \{e_{min}^1, e_{min}^2, \ldots, e_{min}^{k_q}\}$. Errors are evenly allocated to each partition. Finally, we generate the variant set $Var(q, T_{min})$, probe the index structure and extract corresponding inverted lists from I_J.

T_{min} is the minimal common characters, and the bound is not tight enough. In general, for any entity $s \in \mathcal{D}$, we can deduce the common character rate $R_T = \frac{T}{T_{min}} = \frac{|s| \cdot (|q| + l_{min})}{(|q| + |s|) \cdot l_{min}}$. Thus, the minimal number of common characters between

s and q are $\lceil R_T \cdot T_{min} \rceil$. Meanwhile, according to E_{min}, we have $\sum_{i=1}^{k_q} e^i_{min} = |q| - T_{min}$. For any entity $s \in \mathcal{D}$, we can allocate error as $E = \{e^1, e^2, \ldots, e^{k_q}\}$, where $\sum_{i=1}^{k_q} e^i = |q| - \lceil R_T \cdot T_{min} \rceil$. Due to even allocation strategy of errors, we assign e^i $(1 \leq i \leq k_q)$ as $e^i = \left\lfloor \frac{|q| - T_{min}}{|q| - \lceil R_T \cdot T_{min} \rceil} \cdot e^i_{min} \right\rfloor$.

We have the following strategies to accelerate query processing, and Algorithm 1 is the pseudo-code of query processing.

1. **Early Termination.** In any list H of I_J, we can get tuples $(eid, s, k, e) \in H$. Assume e is the smallest allocated error of k partitions of string s. Then we have $|s| - e \cdot k \geq T_{eid} \geq T_{min}$, where T_{eid} is the least number of common characters between s and q. According to Sect. 5, elements in list H of index I_J are all sorted according to the decreasing order of value $|s| - e \cdot k$. Thus, we sequentially scan elements in list H until $|s| - e \cdot k \leq T_{min}$.
2. **Skipping Elements.** We take the length of string in $\mathcal{D}$ into account. For an entry (eid, s, k, e) in list H of I_J, if $|s| - e \cdot k \geq C = \frac{(3\tau_J - 1)|s||q|}{|s| + |q|}$, we consider s or eid as potential candidate, otherwise, we discard and skip it.

Algorithm 1. $JS(\mathcal{D}, q, \tau_J, L, SC)$

```
1  Sort and partition query q into partition P = {q^1, q^2, ..., q^k} according to SC;
2  Calculate the minimal common characters T_min of q;
3  Allocate minimal errors E_min = {e^1_min, e^2_min, ..., e^k_min}, cand = {};
4  for q^i ∈ P do
5      Generate all variant set e_var(q^i) of q^i;
6      for w ∈ e_var(q^i) do
7          if I[w] ≠ ∅ then
8              H ← I[w], T_lb = ((3τ_J − 1)|q||s_min|) / (|q| + |s_min|), where s_min is the minimal length
               of string in H;
9              for (eid, s, k, e) ∈ H do
10                 if |s| − ⌈v · k⌉ ≤ T_lb then
11                     break;
12                 if |s| − ⌈v · k⌉ ≥ C and then
13                     cand = cand ∪ {s};
14 return cand;
```

6.2 Query Processing of Jaro-Winkler Distance

In the query processing of Jaro-Winkler distance, query q first traverses the trie structure to find sets of strings that share from 0 to 4 prefix characters with q. For each level, we calculate their corresponding thresholds of Jaro distance. The following step is to use Algorithm 1 to find results.

7 Experiments

7.1 Experimental Setup

The following algorithms are compared in the experiment.

- **JWS**, **JWS**$_N$ are our methods that apply Pigeonhole principle and e-variants. Here JWS$_N$ means we turn off filtering strategies of our methods.
- **LIMES** [1] [5] is the trie-based method based on Jaro-Winkler distance.
- **Scan** is the baseline method that perform sequential scan on the existing lexicon.

All experiments were conducted on a server with QuadCore AMD Opteron 8378@2.4GHz Processor and 96GB RAM whose system operation is Ubuntu 12.04. In our experiments, we select four publicly available datasets as follow.

- **AMiner-Author** [14] is a collection that contains 1.7 million author names.
- **AIDA-Lexicon** [9] is the lexicon of entities that contain approximate 9 million entities from Wikipedia. We randomly sample 1 million entities as our entity collection.
- **UKB-Lexicon** [1] is the lexicon of entities that contain nearly 4 million entities. We also randomly sample 1 million as one of our datasets
- **IMDB** is the collection that contains approximate 350 thousands actor names.

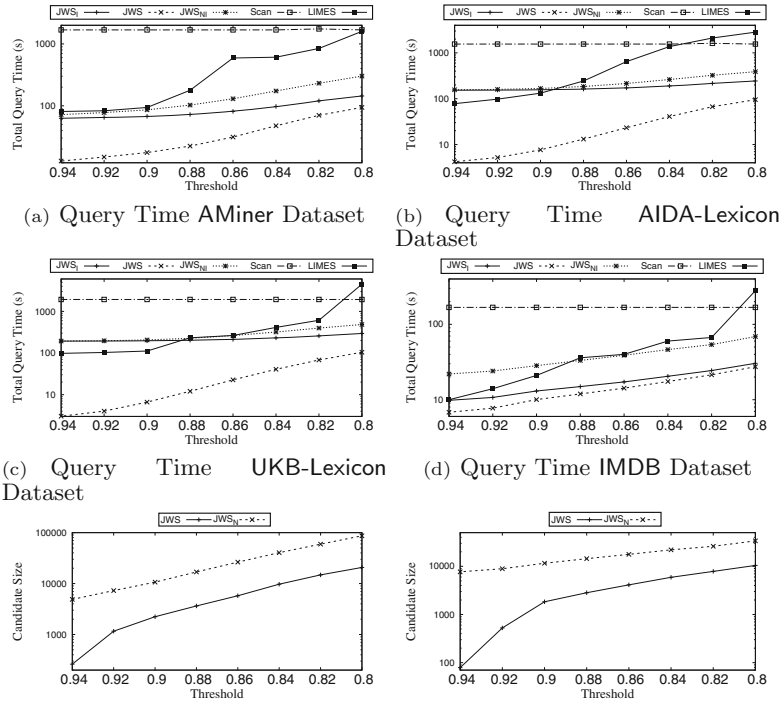
We measure the overall query response time and candidate size of each method in the following.

7.2 Query Performance

We show the total query running time of four datasets. Figure 1(a), (b), (c) and (d) are query time of four methods with Scan on four datasets. Because LIMES deals with string similarity join problem, we add index construction time into the total running time. JWS$_I$ and JWS$_{NI}$ are denoted as methods that involve the index construction time with and without filtering strategies.

Among all methods, Scan almost achieves the worst performance, which is at least 10 times slower than JWS$_I$. LIMES has the second worst query performance. For example, in Fig. 1(a), when $\tau = 0.84$, LIMES is 20 times slower than JWS$_I$. Meanwhile, LIMES does not work well with a small threshold. When $\tau = 0.8$, LIMES is very slow and cannot beat Scan. JWS$_I$ and JWS$_{NI}$ are our methods that involve the index construction time. For different τ, we always construct the index with $\tau_{min} = 0.8$. JWS$_{NI}$ is the method with index construction time but without any filtering strategies. When τ is small (e.g., 0.8, 0.82), JWS$_I$ runs almost 2 times faster than JWS$_{NI}$. This is because filtering strategies can largely prune many false positives. However, as τ increases, their gap is becoming smaller, because pigeonhole principle and e-variants have strong pruning power with high values of τ. JWS$_I$ and JWS$_{NI}$ are faster than LIMES,

[1] https://github.com/AKSW/LIMES-dev.



(a) Query Time AMiner Dataset (b) Query Time AIDA-Lexicon Dataset

(c) Query Time UKB-Lexicon Dataset (d) Query Time IMDB Dataset

(e) Avg. Candidates AMiner Dataset (f) Avg. Candidates IMDB Dataset

Fig. 1. Query running time and average candidate number

especially when τ is small. For example, in Fig. 1(b), JWS_I is nearly 3 times faster when $\tau = 0.84$. This is because LIMES needs to search the trie structure to the deeper levels to make sure all results are retrieved if τ is small. To better show the efficiency of Jaro-Winkler similarity search, JWS is the method that excludes index construction time. JWS is much faster than any other methods. For example, in Fig. 1(c), when $\tau = 0.88$, JWS has 12.06 s of total query time, which is at least 20 times faster than the best of others.

7.3 Candidate Number

We show the candidate number of JWS and JWS_N (See Fig. 1(e) and (f)). Without filtering strategies, JWS_N still has pruning power. For example, in Fig. 1(e), JWS_N can still prune 95% strings when $\tau = 0.8$. JWS is the method that involves filtering strategies, and its pruning power is much more powerful. In Fig. 1(e), JWS can prune 98.82% strings when $\tau = 0.8$, and its candidate number is approximately 4 times smaller than JWS_N. As τ increases, the candidate size of JWS decreases rapidly. This is because the filtering strategies work and dominate the whole filtering procedure with the large threshold.

8 Conclusion

In this paper, we proposed a new lower bound of Jaro-Winkler distance to improve query performance. By using signatures that consist of characters in the same partition, we proposed a method of index construction and online query processing. Moreover, we showed the query running time and candidate number. In future work, we consider to improve the verification cost of Jaro-Winkler distance and propose some parallel query processing methods to solve this problem.

Acknowledgement. This work was supported by ARC DP DP130103401 and DP170103710.

References

1. Agirre, E., Barrena, A., Soroa. A.: Studying the Wikipedia hyperlink graph for relatedness and disambiguation. CoRR, abs/1503.01655 (2015)
2. Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S.E., Widom, J.: Swoosh: a generic approach to entity resolution. VLDB J. **18**(1), 255–276 (2009)
3. Christen, P.: Febrl -: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In: SIGKDD, KDD 2008, New York, NY, USA, pp. 1065–1068. ACM (2008)
4. Cohen, W.W., Ravikumar, P., Fienberg, S.E.: A comparison of string distance metrics for name-matching tasks. In: Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03), 9–10 August 2003, Acapulco, Mexico, pp. 73–78 (2003)
5. Dreßler, K., Ngomo, A.N.: On the efficient execution of bounded jaro-winkler distances. Semant. Web **8**(2), 185–196 (2017)
6. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: a survey. IEEE Trans. Knowl. Data Eng. **19**(1), 1–16 (2007)
7. Feng, J., Wang, J., Li, G.: Trie-join: a trie-based method for efficient string similarity joins. VLDB J. **21**(4), 437–461 (2012)
8. Galárraga, L., Heitz, G., Murphy, K., Suchanek, F.M.: Canonicalizing open knowledge bases. In: CIKM '2014, New York, NY, USA, pp. 1679–1688. ACM (2014)
9. Hoffart, J., Yosef, M.A., Bordino, I., Fürstenau, H., Pinkal, M., Spaniol, M., Taneva, B., Thater, S., Weikum, G.: Robust disambiguation of named entities in text. In: EMNLP 2011, Stroudsburg, PA, USA, pp. 782–792. Association for Computational Linguistics (2011)
10. Li, G., Deng, D., Wang, J., Feng, J.: Pass-join: a partition-based method for similarity joins. Proc. VLDB Endow. **5**(3), 253–264 (2011)
11. Liu, Y., Shen, W., Yuan, X.: Deola: a system for linking author entities in web document with DBLP. In: CIKM (2016)
12. Prokoshyna, N., Szlichta, J., Chiang, F., Miller, R.J., Srivastava, D.: Combining quantitative and logical data cleaning. Proc. VLDB Endow. **9**(4), 300–311 (2015)
13. Qin, J., Wang, W., Lu, Y., Xiao, C., Lin, X.: Efficient exact edit similarity query processing with the asymmetric signature scheme. In: SIGMOD 2011, New York, NY, USA, pp. 1033–1044. ACM (2011)
14. Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., Su, Z.: Arnetminer: extraction and mining of academic social networks. In: KDD 2008, pp. 990–998 (2008)

15. Wang, W., Xiao, C., Lin, X., Zhang, C.: Efficient approximate entity extraction with edit distance constraints. In: SIGMOD 2009, New York, NY, USA, pp. 759–770. ACM (2009)
16. Xiao, C., Wang, W., Lin, X.: Ed-join: an efficient algorithm for similarity joins with edit distance constraints. Proc. VLDB Endow. **1**(1), 933–944 (2008)
17. Xiao, C., Wang, W., Lin, X., Yu, J.X., Wang, G.: Efficient similarity joins for near-duplicate detection. ACM Trans. Database Syst. **36**(3), 15:1–15:41 (2011)