GFSF: A Novel Similarity Join Method Based on Frequency Vector

Ziyu Lin[1(✉)], Daowen Luo[1], and Yongxuan Lai[2]

[1] Department of Computer Science, Xiamen University, Xiamen, China
{ziyulin,luodw}@xmu.edu.cn
[2] School of Software, Xiamen University, Xiamen, China
laiyx@xmu.edu.cn

Abstract. String similarity join is widely used in many fields, e.g. data cleaning, web search, pattern recognition and DNA sequence matching. During the recent years, many similarity join methods have been proposed, for example Pass-Join, Ed-Join, Trie-Join, and so on, among which the Pass-Join algorithm based on edit distance can achieve much better overall performance than the others. But Pass-Join can not effectively filter those candidate pairs which are partially similar. Here a novel algorithm called GFSF is proposed, which introduces two additional filtering steps based on character frequency vector. Through this way, the number of pairs which are only partially similar are greatly reduced, thus greatly reducing the total time of string similarity join process. The experimental results show that the overall performance of the proposed method is better than Pass-Join.

1 Introduction

Similarity join is to find out string pairs satisfying a certain similarity threshold. It has a wide range of applications, such as coalition detect [1], fuzzy keyword matching [2], data integration [3], data cleaning [4], and so on. The classical similarity join algorithms include Ed-Join [5], Trie-Join [6], Pass-Join [7], and so on. They mainly consist of two process, namely, filtering and verification. In the verification process, at present, the widely used measurement function is edit distance [5–7], Jaccard distance [8–10], Cosine [11] and the variants [12,13] of the functions above. In the filtering process, the Ed-Join algorithm uses the n-grams-based [14] method to cut down the original string sets. Then the prefix filtering is used to filter the original candidate string pairs. Finally, the ultimate candidate pairs are verified. Trie-Join uses the form of Trie-tree structure. If the edit distance of the prefix of two branches is more than the threshold, there are no similarity strings in the descendants of the two branches. Among similarity join algorithms, the Pass-Join's overall performance is most desirable. The Pass-Join algorithm divides the string into some segments, and then restricts the

Supported by the Natural Science Foundation of China (61303004), the National Key Technology Support Program (2015BAH16F00/F01) and the Key Technology Program of Xiamen City (3502Z20151016).

B. Cui et al. (Eds.): WAIM 2016, Part II, LNCS 9659, pp. 506–518, 2016.
DOI: 10.1007/978-3-319-39958-4_40

number of segments. If there are a segment matching between two strings, the two strings are added into candidate set and finally verified. These algorithms are able to get correct results and reduce the number of candidate pairs so as to reduce the amount of edit distance calculation.

However, Pass-Join is still with some shortcomings. If there is a common segment in two strings, Pass-Join takes the two strings as a candidate pair. Evidently, many string pairs which can not be similar may also be added into the candidate sets. Because although a lot of strings have a common prefix or suffix, they may not be similar strings. For example, assuming that the string s_1 = "*through*", s_2 = "*thing*" and the threshold of edit distance is 2. Although s_1 and s_2 have the common prefix "th", they are not similar strings because of their edit distance being more than 2. So the candidate set which is produced by Pass-Join can be further filtered and the amount of candidate set can be further reduced.

In order to address the problem discussed above, we propose GFSF (Global-Filtering and Segment- Filtering) similarity join algorithm, which is based on Pass-Join algorithm. Compared to the original Pass-Join algorithm, here two additional filtering steps are introduced in GFSF. First, when there is a common segment in two strings, we calculate the difference of their character frequency vector which represents the degree of difference between the two strings. If the difference is relatively large, it means that it needs more editing steps to make the two strings matching. So, the two strings are mismatched and can be eliminated. Second, after the first step, we take off the common segment from the two strings and respectively calculate the difference of the character frequency vector between the two remaining segments just as the first step. By comparing the difference with a given threshold, we can further filter the two strings which have a common segment and the same characters but the order of characters of which are greatly different. It must be pointed out that although the two additional filtering steps may increases the time of filtering process, the candidate set is cut down greatly, thus greatly decreasing the time of verification. Overall, the reduction of the verification time is more than the increase of filtering time, and therefore our method can achieve much better overall performance than Pass-Join.

In a sum, the contributions of this paper can be summarized as follows:

- We propose GFSF similarity joins algorithm, which can greatly reduce the amount of candidate pairs and therefore reduce the time cost of the verification process.
- For the second filtering step in GFSF, a novel segment filtering method is used to greatly speed up the process of GFSF.
- We conduct extensive experiments to compare our algorithm with the Pass-Join algorithm, and the results show that the overall performance of our algorithm is better than Pass-Join.

The rest of the paper is organized as follows. Section 2 introduces the Pass-Join similarity join algorithm and discusses the deficiency of pass-Join algorithm. The proposed GFSF similarity join algorithm is described in detail in Sect. 3.

Section 4 gives the experimental results. Finally, we review related work in Sect. 5 and conclude in Sect. 6.

2 Pass-Join Algorithm

2.1 The Description of Pass-Join

Assuming that a string is denoted as s and the edit distance is denoted as τ. $|s|$ is the length of s. Pass-Join splits s into τ+1 segments, and then compare s with another string whose length is within the range of$[|s| - \tau, |s|]$. If the string s has a segment matching with another string whose length is within the range of $[|s| - \tau, |s|]$, the two strings will be taken as a candidate pair. Finally, we calculate the edit distance and determine whether the two strings are similar.

Figure 1 shows an example of Pass-Join framework. There are two strings s_1 = "kaushik chakrab", s_2 = "kaushuk chadhui" and the edit distance threshold $\tau = 3$. First, s_1 is spilt into $\tau + 1$ segments using the split method that Pass-Join proposes. Next, we find out whether s_2 contains a substring that matching one of the four segments of s_1. In Fig. 1, s_2 has segments "kau" and "_cha" that matching one of the four segments of s_1, so s_1 and s_2 are taken as a candidate pair. Finally we calculate their edit distance. Their edit distance is greater than τ, so they are not similar strings. Since s_1 and s_2 have the same length denoted as $len = 15$, so s_2 is also split into $\tau + 1$ segments and is added into the inverted index.

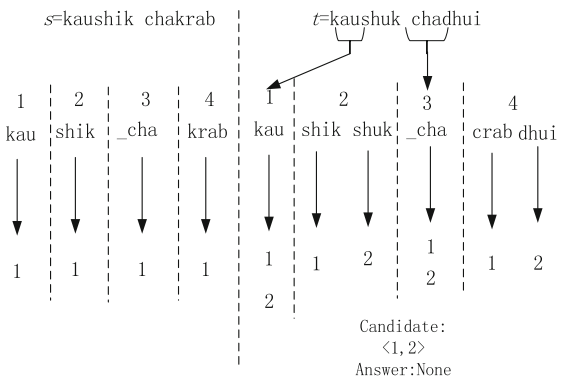


Fig. 1. An example of Pass-Join framework

String Splitting . According to the string split method of Pass-Join, we divide the string into τ+1 segments, but there are many different kinds of methods for segmentation. Pass-Join's segmentation strategy is to split the string into τ+1 fragments whose length are roughly equal. Assuming that a string s, $k = |s| - \lfloor \frac{|s|}{\tau+1} \rfloor * (\tau + 1)$, and then the length of the last k fragments is $\lceil \frac{|s|}{\tau+1} \rceil$, the length of the first $\tau + 1 - k$ fragments is $\lfloor \frac{|s|}{\tau+1} \rfloor$. For a fixed length string,

the length of each segment is fixed. Thus we build the inverted index L, $L_l^i(w)$ represents string sets in which the length of string is l and the ith segment is w.

The main steps of the Pass-Join algorithm are as follows:

1. Sort all of the strings according to the length of the sequence, if the length is equal, sorted according to the lexicographical order;
2. Split strings and build inverted index;
3. Start from the first string to do the join operation: set the length of string s is $|s|$. First list all the segments w of s, then calculate the edit distance between any one of the $L_l^i(w)$ and s, finally get the similar string pairs, l and i are limited as $|s| - \tau \leq l \leq |s|$, $1 \leq i \leq \tau + 1$.

Segment Selection. The third step of the Pass-Join is a segments selection process. Let W(s,L_l^i) represents all the strings that s find in index L_l^i, W(s,l) represents all the segments that s find in $L_l^i (1 \leq i \leq \tau + 1)$. If we enumerate all the segments that s has, then $|W(s, L_l^i)| = \sum_{i=1}^{|s|}(|s| - i + 1)$. That is equal to $\frac{|s|*(|s|+1)}{2}$. This is vary large if s is a long string. And it will greatly influence the efficiency of the algorithm. Therefore, Pass-Join designs four methods for segments selection.

1. **Length-based Method**. In L_l^i, the segments have the same length, so $|W(s,l)| = (\tau + 1)(|s| + 1) - l$;
2. **Shift-based Method**. As for L_l^i, s only selects the segments whose offset to ith segment is in $[-\tau, \tau]$. So $|W(s,l)| = (\tau + 1)(2\tau + 1)$;
3. **Position-aware Substring Selection**. As for L_l^i, s only select the segments whose offset to ith segment is in $[-\lfloor \frac{\tau - \triangle}{2} \rfloor, \lfloor \frac{\tau + \triangle}{2} \rfloor]$. So the $|W(s,l)| = (\tau + 1)^2$;
4. **Multi-match-aware Substring Selection**. This method selects the count of segments is $|W(s,l)| = (\tau + 1)^2$, the detail derivation see paper [6];

2.2 The Disadvantage of Pass-Join

Pass-Join designs four methods to select segments, among which, the fourth is the optimal. However, inefficient filtering is an evident shortcoming of Pass-Join, namely, many string pairs which can not be similar and should be cut, still exist in the resulting candidate set.

Firstly, when the two strings have a segment matching with each other, it is arbitrary for Pass-Join to take the two strings as similar candidate string pair. The two strings are likely to be only partially similar, and then the rest is completely different. So for Pass-Join, to determine whether the candidate pair is similar is too restrictive. For example, there are two stings s_1 = "vasdlym" and s_2 = "vahijkx", the edit distance threshold $\tau = 2$. The two strings contain "va" segment, so the Pass-Join algorithm will take the two strings as a candidate pair. However, the edit distance of the two strings is $6 > \tau$, so the two strings are not similar. Thus, Whether two strings are similar is related to the all the characters that the two strings contain.

Secondly, if two strings have the same characters and the order of the characters is also the same, they are not always similar. For example, there are the two string s_1 = "vasdlym" and s_2 = "myldsva". Although they have the same characters and the count of character is also the same, it's obvious that they are not similar. They only have a character matching, but they don't have a segment matching whose length is more than 2. So the edit distance of the two strings are more than 2, they are not similar strings. Therefore whether two strings are similar is also related to the order of the characters.

3 GFSF Algorithm

In order to address the problem of the Pass-Join algorithm, here a novel algorithm called GFSF(Global-Filtering and Segment-Filtering) is proposed to further filter candidate set generated by Pass-Join Selection Algorithm. In the following, the overview of GFSF is described in Sect. 3.1. Then the two additional filtering steps, namely, Global Filtering and Segment Filtering, are discussed in Sects. 3.2 and 3.3 respectively.

3.1 An Overview of GFSF

Compared with Pass-Join, two additional filtering steps are introduced in GFSF, namely, Global Filtering and Segments Filtering. Global Filtering mainly filters the candidate pairs which are only partially similar. Segment Filtering filters the two strings which have the same characters but the order of characters are different. Figure 2 shows where we add the two filtering steps in Pass-Join.

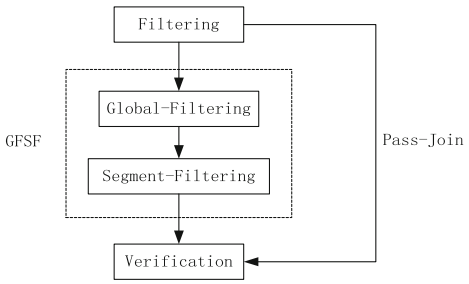


Fig. 2. The overview of GFSF

3.2 Global Filtering

The main idea of the Global Filtering is that when two strings have a segment matching, we do not take them as candidate pair like Pass-Join. Instead we calculate their γ-distance introduced next. If their γ-distance is less than a threshold discussed next, it shows that the difference of their character frequency is not very large. Thus they are likely to become similar strings. Next we will introduce character frequency vector and discussed the γ-distance threshold.

Character Frequency Vector : α_i is any character of alphabet Σ, and how many times the α_i appears is called the frequency of α_i, denoted as $f_i(s)$. $< f_1(s), f_2(s), ..., f_{|\Sigma|}(s) >$ is called s's character frequency vector, denoted as $f(s)$, which is ordered by lexicographical order.

γ-distance: There are two vectors, $U = (u_1, u_2, ..., u_m), V = (v_1, v_2, ..., v_m)$, their γ- distance is $||U - V||_\gamma = \sum_{i=1}^{m} |u_i - v_i|$.

Theorem 1. *There are two strings s and t, if* $||f(s) - f(t)||_\gamma > 2\tau$, *the edit distance between* s *and* t *is greater than* τ.

We know that every edit operation will result in 2 differences of frequency vector between two strings, thus if the edit distance between s and t is less than τ, then $||f(s) - f(t)|| \leq 2\tau$. Subsequently, if $||f(s) - f(t)||_\gamma > 2\tau$, the edit distance between s and t is more than τ.

γ-distance threshold: If s's length is equal to t's length,the difference of γ-distance between s and t is completely resulted by the kind of character. So we can get the following theorem.

Theorem 2. *If the γ-distance between s and t is L, then the edit distance between the two is at least L/2, which is the number of edit operations required to convert s to t.*

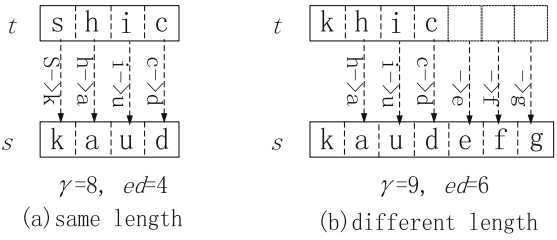


Fig. 3. The conversion process between two strings

Here, we do not take the order of character into count. For example, there are two strings s = "kaud" and t = "shic". The two strings are completely different, so the γ-distance between the two strings is 8, then the edit distance between them is at least 4. Figure 3(a) shows the conversion process.

If s's length is not equal to t's length, the length difference between the two strings must satisfy $||s| - |t|| \leq \tau$. Since Pass-Join compares the current string with other strings whose length are in the range of $[|s| - \tau, |s|]$. If the length difference is greater than τ, the two strings must not be similar strings, because the operation to offset length difference is greater than the edit distance threshold.

Theorem 3. *Given two strings s and t with different length, assuming that* $|s| > |t|$ *and their* γ*-distance is L. If the length difference is equal to len , then the edit distance is at least* $\frac{(L-len)}{2} + len$.

Algorithm $Global - Filtering$(**s,**$L_l^i(w)$)
Input: a string s;
a inverted list $L_l^i(w)$ that contains the strings which contain segment *w*;
Output: a new inverted list $L\_new_l^i(w)$;

1.**Begin**
2. // s_vector is *s*'s character frequency vector;
3. **for** each $r \in L_l^i(w)$ **do**
4. //r_vector is *r*'s character frequency vector;
5. //v_dist is the difference between s_vector and r_vector;
6. **if**$(|s| \doteq |r| and \frac{V\_dist}{2} \leq \tau)$
7. $L\_new_l^i(w) \rightarrow add(r)$
8. **endif**
9. **if**$(|s| \neq |r| and \frac{V\_dist - (||s|-|r||)}{2} + (||s| - |r||) \leq \tau)$
10. $L\_new_l^i(w) \rightarrow add(r)$
11. **endif**
12. **endfor**
13.**End**

Fig. 4. The algorithm of Global Filtering

Since the length difference between the two strings is *len*, it needs at least *len* edit operations to offset the length difference. *s* deducts the length difference and the γ-distance of the rest with t is resulted by the kind of character just as the above discussion. So the edit distance between the s and t is at least $\frac{(L-len)}{2} + len$. Here we do not take the order of character into account. For example, two strings $s =$ "kaudefg" and $t =$ "khic". Their γ-distance is 9 and length difference is 3, then $\frac{(L-len)}{2} + len = 3$, so their edit distance is at least 6. Figure 3(b) shows the conversion process.

The global filtering mainly uses the γ-distance and γ-distance threshold to filter candidate sets. The main steps of GFSF is as follows:

1. Select all the segments of s, denoted as $W(s, L_i^l)$, by the means of Pass-Join's selection algorithm.
2. Match all the segments of $W(s, L_i^l)$ with the segments in inverted index, if not matched, iterate next string, or go to step 3.
3. Calculate the character frequency vector of the two strings.
4. Calculate the γ-distance between the two strings, if the γ-distance is less than a threshold τ, it can judge that the two strings are candidate pairs.

From the above discussion, we know that we can further filter the candidate set by γ-distance. So we use γ-distance to filter the candidate set produced by Pass-Join.

Algorithm 1 in Fig. 4 formally describes global filtering. When Pass-Join selection method gets the segment sets of s and the inverted list $L_l^i(w)$ which

contains the strings that also contains segment w, we do not calculate the edit distance between s and strings in $L_l^i(w)$. Instead, we iterate every strings denoted as r in $L_l^i(w)$ and calculate γ-distance between s and r. Then we continue the process as follows:

1. The lengths of the two strings are equal. If $\frac{\gamma}{2} \leq \tau$, we reserve them, else discard them;
2. The length of the two strings are not equal. If $\frac{\gamma-(||s|-|t||)}{2} + ||s| - |t|| \leq \tau$, we reserve them, else discard them.

3.3 Segment Filtering

Although global filtering can filter some candidate pairs which are taken as candidate similar string pairs because of their partial similarity, it does not take the character order into count. Two strings with completely same characters and with a segment matching are either not likely to be similar, although Pass-Join will take them as candidate pair. For example, two strings $s =$ "abcdexyz" and $t =$ "xyzdeabc" and assuming that $\tau \doteq 2$. Although they have completely same characters and have a common segment"de", they are not similar strings. Pass-Join will take them as a candidate pair because of common segment "de". But it can be filtered by a certain method. Next we will introduce the method and call it segment filtering.

Segment filtering is the further filtering process after global filtering. The main idea of segment filtering is that if two strings have a common segment, then the first segment of the two strings will calculate their γ-distance respectively and judge whether they meet the threshold. The second segment of the two strings will also be processed as the first segment. If they meet the threshold, then we will take the two strings as candidate pair ultimately. Global filtering analyzes the problem from the global perspective, but will ignore the order of character. Segment filtering analyzes the problem from local perspective and will well solve the problem of character order. Next we show the steps of segment filters:

1. If s's first segment and t's first segment do not meet the threshold, then they must not be similar and there is no need to calculate the γ-distance of the second segment of the two strings.
2. If s's first segment with the t's first segment meet the threshold, we denoted their γ-distance as v_1, so we will calculate the γ-distance of s's second segment and t's second segment, denoted as v_2. If $v_1 + v_2 \leq threshold$,then we will take them as candidate pair ultimately. If $v_1 + v_2 > threshold$, we will discard them.

Here we use the example previously to explain the segment filtering. Given two strings $s =$ "abcdexyz" and $t =$ "xyzdeabc", and assuming that $\tau \doteq 2$. They have a common segment "de", So we calculate the γ-distance between the first segment of the two strings. Because the γ-distance between the "abc" and "xyz" is 6, the edit distance is at least 3. Since $3 > \tau$, s and t must not

Algorithm $Segment - Filtering(s, L_l^i(w))$
Input: a string s;
a inverted list $L_l^i(w)$ that contains the strings which contain segment *w*;
Output: a new inverted list $L\_new_l^i(w)$;

14.**Begin**
15. s_seg[2]=$\{se|se \in s$ and $se \neq w\}$ // se is the segment of s except segment *w*;
16. **for** each $r \in L_l^i(w)$ **do**
17. r_seg[2]=$\{re|re \in r$ and $re \neq w\}$ // re is the segment of r except segment *w*;
18. // *v1_dist* is the γ-distance between s_seg[1] and r_seg[1];
19. **if**(*v1_dist* is not meet the threshold)
20. **continue**;
21. **endif**
22. **if**(*v1_dist* is meet the threshold)
23. // *v2_dist* is the γ-distance between s_seg[2] and r_seg[2];
24. $v\_dist = v1\_dist + v2\_dist$;
25. **if**(*v_dist* is meet the threshold)
26. $L\_new_l^i(w) \rightarrow add(r)$
27. **endif**
28. **endif**
29. **endfor**
30.**End**

Fig. 5. The algorithm of Segment Filtering

be similar strings. Now we can discard s and t. If the two strings are changed to s = "abmdenxy" and t = "xymdenab". Because the γ-distance between the first segment "ab" and"xy" is $v_1 = 4$, edit distance is at least 2. Since $2 \leq \tau$, we calculate γ-distance of the second segment. The γ-distance between"xy" and "ab" is $v_2 = 4$, so the edit distance is at least 2. Since $\frac{v_1+v_2}{2} \doteq 4 > \tau$, the two strings are not similar strings, and therefore we discard them.

Algorithm 2 in Fig. 5 formally describes the segment filtering. First, we get the two segments of string s except the common segment w. And then, we iterate every string in $L_l^i(w)$, denoted as r. We also get the two segments of string r except the common segment w. Next we calculate the γ-distance between s_seg [1] and r_seg [1], denoted as *v1_dist* . If *v1_dist* does not meet the threshold discussed in global filtering, we discard the two strings. If *v1_dist* meets the threshold discussed in global filtering, we calculate the γ-distance between s_seg [2] and r_seg [2], denoted as *v2_dist*. We let *v_dist* be equal to the sum of *v1_dist* and *v2_dist*. If *v_dist* meets the threshold, we take them as candidate pair ultimately, else we discard them.

3.4 Performance Analysis

The calculation of edit distance is a time-consuming work, so many algorithms including Pass-Join for similarity join pay much attention to the filtering process,

so as to greatly reduce the amount of string pairs and decrease the time of the verification process. Although GFSF introduces two additional filtering steps, the decrease of verification is more than the increase of the time resulting from the two filtering steps. So the overall performance of GFSF is much better than Pass-Join, which can be shown in experimental results in Sect. 4.

4 Empirical Study

In this section, we conduct experiments to verify the efficiency of our approach. We mainly use the number of candidate pairs to demonstrate performance difference. We compare the Pass-Join, Pass-Join with Global-Filtering and GFSF. If the number of the candidate pairs of GFSF is the least, it means that GFSF is effective for filtering candidate set. We also compare the elapsed time of the GFSF with Pass-Join. We will show that GFSF algorithm can not only filter candidate set but also speed up Pass-Join algorithm.

4.1 Environmental Setup

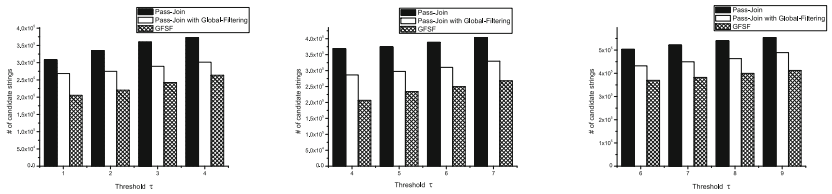
All the algorithms are implemented in C++ and compiled using GCC 3.4 with -O3 flag. All the experiments run on a Ubuntu machine with an Intel(R) Core(TM) i3 3.10GHZ processor and 6G memory. We use the same datasets with PASS-Join, but some details are different. We also use strings with three kinds of length, so as to show that our algorithm is effective for Pass-Join in different cases. Table 1 shows the detail of datasets.

Table 1. Datasets

Datasets	Cardinality	Avg Len	MAX Len	Min Len
Author	423178	13.348	42	6
QueryLog	469427	46.742	497	28
Author + *Title*	642094	110.482	893	23

4.2 Evaluating Filtering Efficiency

We first compared the number of candidate pairs. As Fig. 6 shows, when the threshold increases, the number of candidate pairs also increases. The reason is that the larger the threshold is, the more difference the candidate pairs can contain. It can also be seen from Fig. 6 that, compared with the size of candidate set produced by Pass-Join, the size of candidate sets produced by Pass-Join with global filtering and GFSF, are reduced by 19 percent and 33 percent respectively. So GFSF has a good effect for filtering string pairs and can filter many string pairs that Pass-Join take as candidate pairs.

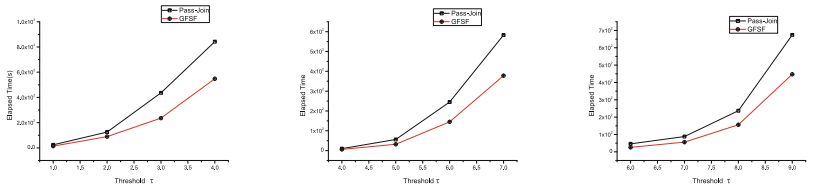


(a) Author(Avg Len=13) (b) QueryLog(Avg Len=47) (c) AuthTil(Avg Len=110)

Fig. 6. The number of candidate pairs in three cases

4.3 Evaluating Elapsed Time

Now we show that GFSF algorithm can not only filter many string pairs, but also can speed up the whole similarity join process than Pass-Join. We conduct three different experiments with three different experimental datasets, namely, *Author*, *QueryLog*, *Author* + *Title*. From the Fig. 7, we know that as the edit distance threshold becomes larger, the elapsed time of Pass-Join and GFSF also become larger because of the increase of candidate set. And compared with the elapsed time of Pass-Join, the elapsed time of GFSF is reduced by 23 percent. So we can get that GFSF can speed up the similarity join process much better than Pass-Join.



(a) Author(Avg Len=13) (b) QueryLog(Avg Len=47) (c) AuthTil(Avg Len=110)

Fig. 7. The elapsed time in three cases

5 Related Work

In recently years, various approaches have been proposed to deal with similarity joins. Most of the them take the Filter-And-Refine as their framework. The main idea of this framework is that in the filter step, they will use a special index structure and generate a small number of candidate pairs that may be similar. In the refine step, they will use edit distance algorithm or other measure functions to verify the candidate string pairs and then find the similar string pairs. Recently, ED-Join, Trie-Join and Pass-Join are the three mainstream approaches with high efficiency to deal with the similarity strings.

ED-Join is based on q-gram. If two q-grams are matching, it means that they have the common token and the position offsets of their positions in string

do not exceed the edit distance threshold. Based on q-gram matching, ED-Join algorithm proposed three kinds of string similar filtering methods, namely, Prefix Filtering, Location-based Filtering and Content-based Filtering. Through the three filter methods, ED-Join can filter many string pairs which can not be similar.

Trie-Join has completely different filtering algorithm compared with ED-Join. Trie-Join use trie structure to store every string. Every path from root to leaf represents a string, so every node represents a character of string. Trie-Join uses active node to filter strings. If a node is not an active node of a certain string, then we can filter the descendants of active node of the string. This is the filtering principle of Trie-Join. Trie-Join is efficient for short strings.

Recently, MapReduce is introduced to improve the efficiency of similar join, such as [15–17]. First, strings are divided into groups. Second, in the map process, every group are calculated and candidate sets are found. In the reduce process, every candidate set are merged to be a ultimate candidate set. By using the parallel computation of MapReduce, it can greatly speed up the process of similarity join. In addition, Simrank [18] and SPB-tree [19] structure are also used to process similarity join, they can also get a good result.

6 Conclusion

In this paper, we propose GFSF algorithm to overcome the Pass-Join's shortcoming. On one hand, global filtering filters the string pairs that are only partially similar; on the other hand, segment filtering filters the string pairs which have the same characters but the order of characters are different. By the additional filtering steps, GFSF can filter more strings and speed up the process of verification. The experimental results show that GFSF algorithm is more efficient than Pass-Join algorithm. Our future work includes further improving the performance of GFSF algorithm and using GFSF in some applications.

References

1. Metwally, A., Agrawal, D., Abbadi, A.E.: Detectives: Detecting coalition hit inflation attacks in advertising networks streams. In: Proceedings of 16th International Conference on World Wide Web, pp. 241–250. ACM Press, New York (2007)
2. Ji, S., Li, G., Li, C., et al.: Efficient interactive fuzzy keyword search. In: Proceedings of the 18th International Conference on World Wide Web, pp. 371–380. ACM Press, New York (2009)
3. Dong, X., Halevy, A., Yu, C.: Data integration with uncertainty. Int. J. Very Large Data Bases **18**(2), 469–500 (2009)
4. Chaudhuri, S., Ganti, V., Kaushik, R.: A primitive operator for similarity joins in data cleaning. In: Proceedings of the 22nd International Conference on Data Engineering, p. 5. IEEE Press (2006)
5. Xiao, C., Wang, W., Lin, X.: Ed-Join: an efficient algorithm for similarity joins with edit distance constraints. PVLDB **1**(1), 933–944 (2008)

6. Wang, J., Li, G., Feng, J.: Trie-Join: efficient trie-based string similarity joins with edit-distance constraints. PVLDB **3**(1), 1219–1230 (2010)
7. Li, G., Deng, D., Wang, J., et al.: Pass-Join: a partition-based method for similarity joins. Proc. VLDB Endow. **5**(3), 253–264 (2011)
8. Sarwagi, S., Kirpal, A.: Efficient set joins on similarity predicates. In: Proceedings of ACM SIGMOD International Conference on Management of data, pp. 743–754. ACM Press, New York (2004)
9. Xiao, C., Wang, W., Lin, X., et al.: Efficient similarity joins for near-duplicate detection. ACM Trans. Database Syst. **36**(3), 15 (2011)
10. Vernica, R., Carey, M.J., Li, C.: Efficient parallel set-similarity joins using MapReduce. In: Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 495–506. ACM Press, New York (2010)
11. Bayardo, R.J., Ma, Y., Srikant, R.: Scaling up all pairs similarity search. In: Proceedings of the 16th International WWW Conference, pp. 131–140 (2007)
12. Wang, J., Li, G., Fe, J.: Fast-join: an efficient method for fuzzy token matching based string similarity join. In: Proceedings of the 27th IEEE International Conference on Data Engineering, pp. 458–469. IEEE Press (2011)
13. Chaudhuri, S., Ganjam, K., Ganti, V., et al.: Robust, efficient fuzzy match for online data cleaning. In: Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 313–324. ACM Press, New York (2003)
14. Gravano, L., Ipeirotis, P., Jagadish, H., et al.: Approximate string joins in a database (almost) for free. In: Proceedings of the International Conference on Very Large Databases, pp. 491–500 (2001)
15. Metwally, A., Faloutsos, C.: V-SMART-Join: A scalable MapReduce framework for all-pair similarity joins of multisets and vectors. PVLDB **5**(8), 704–715 (2012)
16. Deng, D., Li, G., Hao, S., Wang, J., Feng, J.: MassJoin: A MapReduce-based method for scalable string similarity joins. In: ICDE 2014, pp. 340–351 (2014)
17. Huang, J., Zhang, R., Buyya, R., Chen, J.: MELODY-JOIN: Efficient Earth Mover's Distance similarity joins using MapReduce. In: ICDE 2014, pp. 808–819 (2014)
18. Chen, L., Gao, Y., Li, X., Jensen, C.S., Chen, G.: Effcient metric indexing for similarity search. In: Proceedings of IEEE 31st International Conference on Data Engineering, pp. 591–602, April 2015
19. Maehara, T., Kusumoto, M., Kawarabayashi, K.: Scalable SimRank join algorithm. In: 2015 IEEE 31st International Conference on Data Engineering (ICDE), pp. 603–614 (2015)