More Efficient Filtration Method for Big Data Order-Preserving Matching

Wenchao Jiang(✉), Dexi Lin, Sui Lin, Chuanjie Li, and Aobing Sun

School of Computer Science and Technology, Guangdong University of Technology, Guangzhou 510006, China
jiangwenchao@gdut.edu.cn

Abstract. Data matching and retrieval aims at finding out similar substrings with the pattern P in the given data set T. This problem has wide applications in big data analysis. A liberalized verification rule is proposed first, and then a similarity computing based order preserving matching method is presented. Theory analysis indicates our method runs in linear. Furthermore, the experimental results show that our method can improve effectively the precision ratio and the recall ratio. More qualified matching results can be detected compared with the state of the art of this problem.

Keywords: Pattern recognition · Order-preserving matching · Similarity retrieval

1 Introduction

Fast and accurate data matching and retrieval is one of the key problems in big data applications such as video retrieval, stock analysis and prediction. Based on the context and application scenarios, the data objections can be abstracted into a series of vectors with different properties. Furthermore, the different properties can be integrated into a number through reduction or conversion. Consequently, data matching problem will be transformed into string or number matching problem which is one kind of well known problems in pattern recognition. Given a set of numbers T of length n and a pattern P of length m, both being numbers or strings over a finite alphabet $\sum$, the task of string matching is to find all the substrings u in T which have the same relative order as P, and $|u| = |P|$. For example, let P = (10, 22, 15, 30, 20, 18, 27) and T = (22, 85, 79, 24, 42, 27, 62, 40, 32, 47, 69, 55, 25), then the relative order of P matches the substring u = (24, 42, 27, 62, 40, 32, 47) of T [1].

Several online [3–7] and one offline solution [2] have been proposed for the string matching problem. Kim et al. [3] and Kubica et al. [4] presented solutions based on the Knuth-Morris-Pratt algorithm (KMP) [8], the KMP algorithm is mutated such that is determines if the text contains substring with the same relative order as that of the pattern using the order-borders table. Kim et al. [3] utilized the prefix representation method to find the rank of each number in the prefix, and this method was further optimized using the nearest neighbor representation to overcome the overhead involved in computing

Q. Zu and B. Hu (Eds.): HCC 2017, LNCS 10745, pp. 85–94, 2018.
https://doi.org/10.1007/978-3-319-74521-3_10

the rank function. Later, Cho et al. [5, 6] gave a sublinear solution based on the bad character heuristic of the Boyer-Moore algorithm [9]. Almost at the same time, Belazzougui et al. [7] derived an optimal sublinear solution. The state of the art for this problem is the filtration method [1] which is presented by Chhabra et al. All the verification rules in previous researching demanded the values and the positions of the numbers in both T and P must be coherent strictly. So, some actual matching results may be discarded. Moreover, almost all the earlier researching were focused on the time complexity analysis while ignored the precision ratio analysis and recall ratio analysis. They didn't research whether some actual matching results were missing despite the algorithm became more and more fast.

In view of the above problem, a similarity computing based string or number order preserving matching method is presented. Based on the preprocessing, i.e. binary transforming and filtration, our method proposes a novel verification rule which can guarantee more candidate results can be found out. Then, a similarity computing based sorting method is presented which can ensure all the matching results are listed according to the similarities with the pattern P. Theory analysis indicates our method is sublinear. The experimental results show that our method can improve effectively the precision ratio and the recall ratio compared with the newest method at present.

This paper is organized as follows. Section 2 presents our solution. Section 2.4 analyses our approach. The experimental results are given and discussed in Sect. 3. Section 4 concludes this paper.

2 Our Solution

2.1 Problem Description and Motivation

The state of the art for order-preserving matching is the filtration method [1] which was presented by Chhabra et al. We call the filtration method as MT.C in this paper. The MT.C transform the original data T and the pattern P into binary string T' and P' according to formulation (1). Then searching for the substring with the same relative order with P in T can be transformed into searching P' in the analogously T'. In the above example, P' = 101001 and T' = 100101001100. Each occurrence is a match candidate which is verified following the numerical order of the positions in the original pattern P.

$$t_i = \begin{cases} 1, & t_{i-1} < t_i \\ 0, & t_i \le t_{i+1} \end{cases}, \quad t_i \in T; \quad p_i = \begin{cases} 1, & p_{i-1} < p_i \\ 0, & p_i \le p_{i+1} \end{cases}, \quad p_i \in P \tag{1}$$

The MT.C method made the matching process simpler and more efficient than the earlier solutions. However, some deficiencies could be found because of the too harsh filtration and verification rules. The MT.C method required the order of the numbers is strictly coherent according to the values and the corresponding positions in P and T. For example, as shown in Fig. 1(a), the MT.C method can't find the differences between T and P if the max-number increases or the mini-number decreases immensely in T. Moreover, as shown in Fig. 1(b), the MT.C method can't find the differences either if

certain subsection data in T jumps suddenly while the variation trend maintains as a whole. Furthermore, to find out the matching substrings from the candidate strings which were produced through filtration algorithm, the numbers of P must be sorted, and the verification processing demands the positions and the size relations between T and P must be coherent strictly. This rule would result in the looseness of some actual matching substrings. As shown in Fig. 1(c), according to the MT.C method, the gray node x in T can only change between dashed line a and dashed line b which represent the right neighbor y and the left neighbor z respectively in the sorting of T. Once the gray node x changes above the dashed line a or down the dashed line b, the MT.C method will regard that T is not matching with P. Then the corresponding substring will be discarded consequently. Actually, we can find that this is not the case especially when |y-z| is small enough. For example, when x changes to x' which is slightly larger than y or changes to x" which is slightly smaller than z, the MT.C method will discard the substring. But we think the substring is still similar with P in most actual applications such as data retrieval.

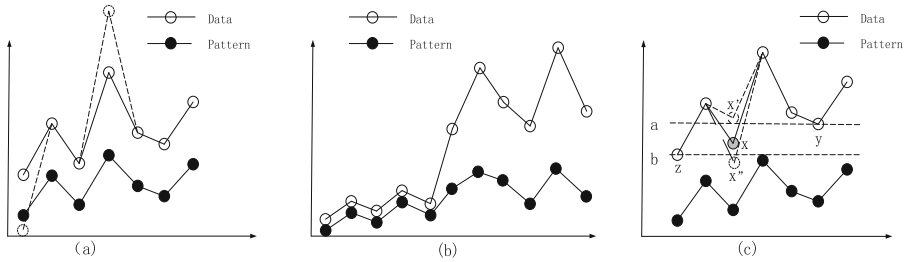

Fig. 1. Problem description

2.2 Our Solution

Problem definition of string or number matching in [1] is described as following: Two strings u = u1u2…um and v = v1v2…vm of the same length over $\sum$ are called order-isomorphic, written u ≈ v, if formulation (2) holds.

$$u_i \le u_j \Leftrightarrow v_i \le v_j \, for \; 1 \le i,j \le m. \tag{2}$$

This rule demands all numbers in u and v are strictly coherent with both the sizes and their positions. So, some actual matching results would be discarded as shown in Fig. 1(c). To overcome this deficiency, a different rule is presented as following. Two strings u = u1u2…um and v = v1v2…vm of the same length over $\sum$. The numbers of v = v1v2…vm are sorted firstly and the result is a sequential table r as formulation (3).

$$r = \{v_{r_{[i]}} | v_{r_{[i]}} \le v_{r_{[j]}} , \quad 1 \le i < j \le m\} \tag{3}$$

$$\begin{aligned} &\left|u_{r[i]} - u_{r[i]-1}\right| \le \frac{\left|u_{r[i]+1} - u_{r[i]-1}\right|}{2} \text{ or} \\ &\left|u_{r[i]+1} - u_{r[i]}\right| \le \frac{\left|u_{r[i]+1} - u_{r[i]-1}\right|}{2} \text{ for } v_{r[i]-1} \le v_{r[i]} \le v_{r[i]+1},\ 1 \le i \le m. \end{aligned} \tag{4}$$

Two strings u and v are called order-isomorphic, written u ≈ v, if formulation (4) holds. Apparently, multiple matching results would be found out according to formulation (4) once the pattern is given. But, the similar extent of the multiple matching results are different. To distinguish the similar level of multiple candidate results, a similarity computing method should be given based on the optimized verification method. Considering the possible concussion range of some numbers in T which have the same positions with relevant numbers in P. The similarity function f, as shown in Fig. 2, between the candidate substring u and the pattern P is defined as formulation (5).

$$f(u,P) = \sum |g(u_i) - g(p_i)|, \quad 1 \le i \le m. \tag{5}$$

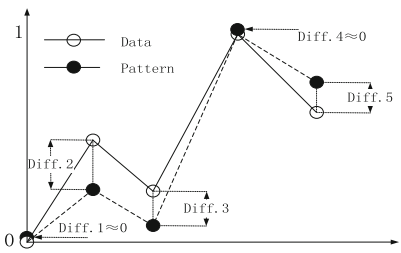

Fig. 2. Data reduction

Where, ui and pi are the relevant numbers which have the same position order in the candidate substring u and the pattern P. m is the length of u and P. g is the reduction function, shown in formulation (6), which can reduce all the numbers in u and p into the same zone [0,1]. Min(u) and Min (P) are separately the Min-values in u and P. Max(u) and Max(P) are separately the Max-values in u and P.

$$u_i - Min(u)/Max(u) - Min(u) \text{ or } p_i - Min(P)/Max(P) - Min(P) \tag{6}$$

As shown in Fig. 2, the similarity between the candidate substring u and the pattern P can be computed easily as the sum of all the differences, represented by Diff.i (1 ≤ i ≤ n), between the relevant nodes of u and P according to formulation (5).

2.3 The Proposed Algorithm

As described above, our solution includes four steps: binary transformation, filtration, verification and similarity computing. The binary transformation can be processed according to [1], and the filtration can be conducted using any exact string matching algorithm. Supposing the binary transformation and filtration have been finished, the

pseudo-code of our solution based on the optimization and the similarity computing can be described as Algorithm 1 and Algorithm 2.

Algorithm 1: Computing the similarity

1) X_t=AnticipationBinary (T_r);
2) X_p=AnticipationBinary (P);
3) For i=0 to length;
4) Sum+=Abs(X_t[i] - X_p[i]);
5) Return Sum.

Algorithm 2: AnticipationBinary is reduction function

1) intput A;
2) Max=A.Max;
3) Min=A.Min;
4) For i=0 to length
5) If (Max==Min);
6) B[i]=0;
7) else
8) B[i]=(A[i]-Min)/(Max-Min);
9) Return B.

2.4 Algorithm Analysis

Our solution, represented as MS, include four steps: Binary Transformation (BT), filtration, verification and Similarity Computing (SC). So, the time complexity of our solution O(MS) can be represented as following.

$$O(M_S) = O(BT) + O(filtration) + O(verification) + O(SC)$$

Compared with [1], the main differences of our solution are the verification and similarity computing. Furthermore, the main modification in verification is a liberalized verification rule is implemented and more matching results can be detected. So, the time complexity of verification doesn't be changed and only the O(SC) need be analyzed.

Supposing the numbers in P and T are integers and they are statistically independent of each other and the distribution of numbers is discrete uniform. Supposing LT = n, LP = m. In the worst case, the similarity computing process requires O(m(n-m)) similarity computing operations. In most cases, LT is usually very large while LP is usually a constant small enough. So, O(m(n-m)) ≈ O(nm) on average which is equal with MT.C method. So, we can conclude O(MS) = O(MT.C) which are all sublinear.

3 Experimental Results

Our experiments used linear string matching algorithm KMP [8] as the filtration method. The tests were run on single node of Tianhe-2 super computer with configuration CPU E5-2692 v2 12*2 2.20 GHz. All the algorithms were implemented in C# in the 64-bit mode.

3.1 Effectiveness

To explain the superiority of our method, two special data sets are generated based on one basic data set. The basic data set was given as T = (10, 14, 12, 15, 13, 28, 36, 32, 24, 38, 26) and P = (10, 14, 12, 15, 13, 19, 21, 20, 17, 22, 18). The first special data set shown in Fig. 3(a) was generated by multiplying n ($1 < n < 100$) to the last six numbers in T. The second special data set shown in Fig. 3(b) was generated by multiplying n ($1 < n < 100$) to only the 10th number in T.

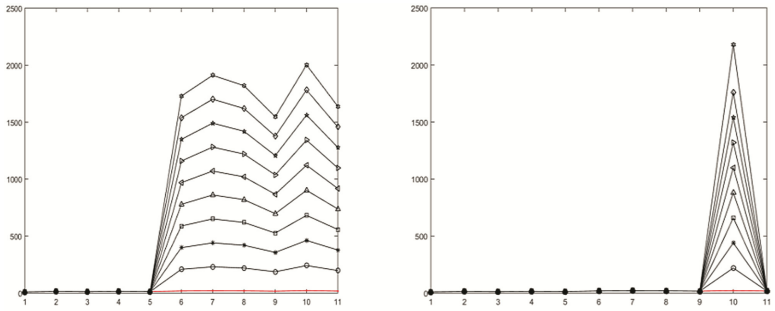

Fig. 3. Data set samples

From Fig. 3(a) and (b), we can find that the data T and the pattern P become more and more dissimilar with the increasing of n. But, the MT.C method couldn't find the differences among them. Through computing the similarities between T and P according to the technologies presented in Sect. 2, the variation trend of the similarity between T and P with the increasing of n was shown in Fig. 4. According to the definition of similarity shown as formulation (5), the more bigger the value of the similarity is, the more dissimilar the substring is compared with P. We can find that the similarity increases with the increasing of n and becomes converging with the infinite increasing of n. The point of inflection means that the data set T couldn't be considered similar with P according to our method. At the same time, we can find that the points of inflection are different with different data sets. So there is no stable point of inflection for different data sets in our method. The appearance of the point of inflection depend on the data set itself.

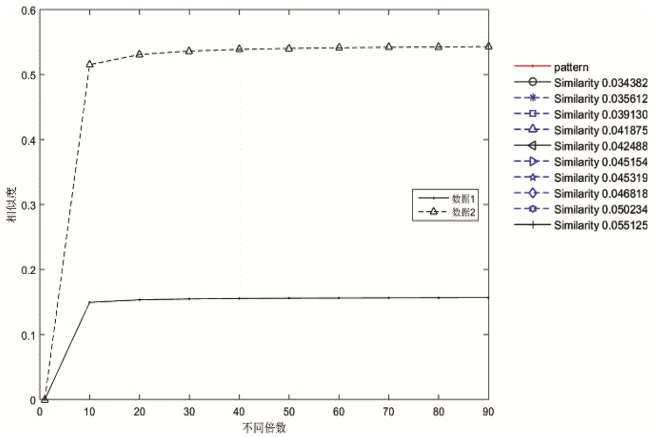

Fig. 4. Similarity convergence

Imitating the data generation method in [1], we generated two special data set and one random data set. The random data set contains 100000000 random integers between 0 and 230. The lengths of patterns (LP) were picked as 10, 15 and 30. The experimental results showed that our method can find more matching substrings than the MT.C method. For example, when the pattern length was picked as 10, the MT.C method can only find 15 results. However, our method can find 135 results which were sorted according to the similarities. The first 10 most similar results were shown in Figs. 5, 6 and 7 separately when the length of pattern were picked as 10, 15 and 30. The red line represents the pattern, and the blue lines represent the missing results using MT.C method while are detected using our solution. The black lines represent the matching results which can be detected using both MT.C method and our solution.

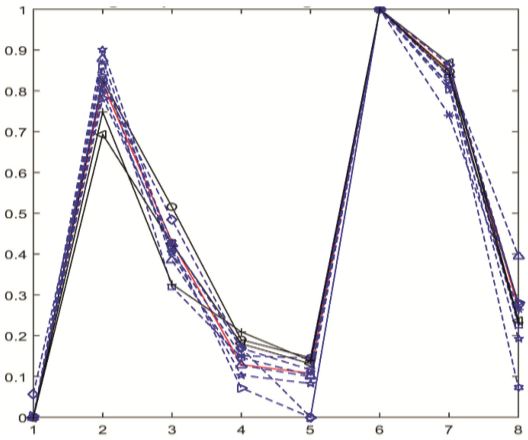

Fig. 5. The most similar 10 results ($L_P = 10$)

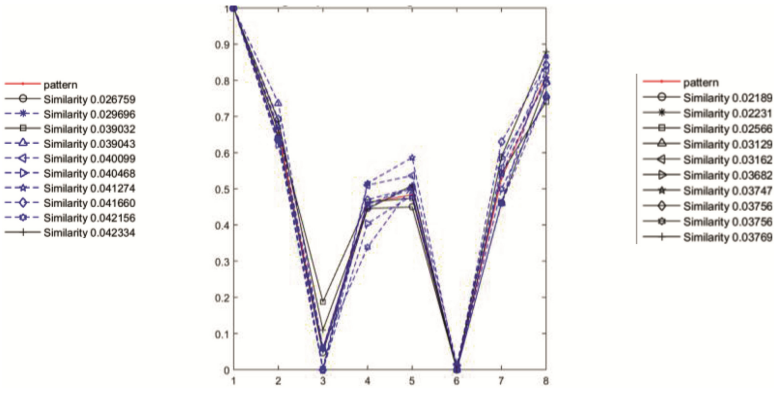

Fig. 6. The most similar 10 results ($L_P = 15$)

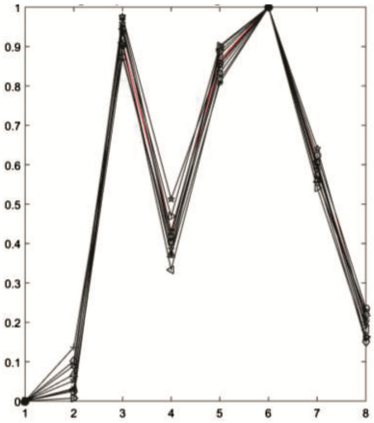

Fig. 7. The most similar 10 results ($L_P = 30$)

3.2 Time Consuming

Furthermore, to compare the time consuming between our method and the MT.C method. We enlarged the length of T from 10000000 to 49000000 gradually with an increment of 1000000. Ten experiments were conducted and the average time consuming was computed with the different lengths of patterns such as 5, 30 and 50. The statistical results were shown in Fig. 8. Because our method could find more results than the MT.C method, and more similarity computing operations were needed. So the overall executing time appear a small amount of growth on equal conditions. But, the executing time growth decreased quickly with the increasing of the length of the pattern. Because the number of matching results decreased quickly with the increasing of the length of the pattern.

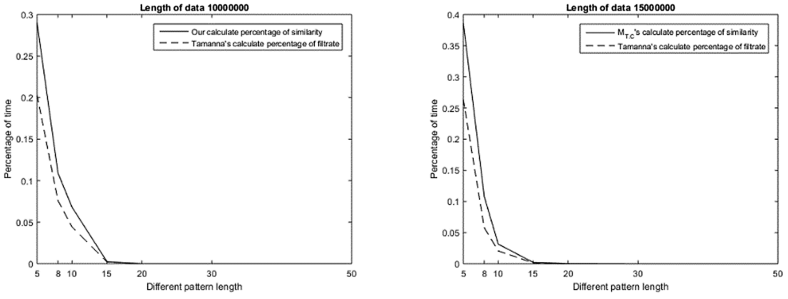

Fig. 8. Comparison of average time consuming

4 Conclusion

Fast and accurate data matching and retrieval is a key problem in big data applications such as video retrieval, stock analysis and bioinformatics etc. A similarity computing based string or number order preserving matching method is presented. Theory analysis indicates our method is sublinear. Furthermore, the experimental results show that our method can improve effectively the precision ratio and the recall ratio compared with the newest method at present under the same conditions. Compared with former research works for order-preserving matching problem, our solution liberalized the verification rules on certain extent and so more qualified matching results can be detected and found out.

Acknowledgements. This work is supported by Guangdong Province Natural Science Foundation (2016A030313703) and Guangdong Province science and technology program (2015B010109001, 2015B010131001, 2016B030305002, 2016YFB10005000, 2017B090901005, 2017A070712016).

References

1. Chhabra, T., Tarhio, J.: A filtration method for order-preserving matching. Inf. Process. Lett. **116**(2), 71–74 (2016)
2. Chhabra, T., Külekci, M.O., Tarhio, J.: Alternative algorithms for order-preserving matching. In: Prague Stringology Conference (2015)
3. Chhabra, T., Giaquinta, E., Tarhio, J.: Filtration algorithms for approximate order-preserving matching. In: Iliopoulos, C., Puglisi, S., Yilmaz, E. (eds.) SPIRE 2015. LNCS, vol. 9309, pp. 177–187. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23826-5_18
4. Cantone, D., Faro, S., Külekci, M.O.: An efficient skip-search approach to the order-preserving pattern matching problem. In: Prague Stringology Conference (2015)
5. Knuth, D.E., Morris, J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM J. Comput. **6**(2), 323–350 (1977)
6. Crochemore, M., Iliopoulos, C.S., Kociumaka, T., Kubica, M., Langiu, A., Pissis, Solon P., Radoszewski, J., Rytter, W., Waleń, T.: Order-preserving incomplete suffix trees and order-preserving indexes. In: Kurland, O., Lewenstein, M., Porat, E. (eds.) SPIRE 2013. LNCS, vol. 8214, pp. 84–95. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-02432-5_13
7. Kim, J., Eades, P., Fleischer, R., et al.: Order-preserving matching. Theoret. Comput. Sci. **525**(4), 68–79 (2013)
8. Boyer, R.S.: A fast string searching algorithm. Commun. ACM **20**(10), 762–772 (1977)
9. Cho, S., Na, J.C., Park, K., Sim, J.S.: Fast order-preserving pattern matching. In: Widmayer, P., Xu, Y., Zhu, B. (eds.) COCOA 2013. LNCS, vol. 8287, pp. 295–305. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-03780-6_26
10. Cho, S., Na, J.C., Park, K., et al.: A fast algorithm for order-preserving pattern matching. Inf. Process. Lett. **115**(2), 397–402 (2015)
11. Belazzougui, D., Pierrot, A., Raffinot, M., Vialette, S.: Single and multiple consecutive permutation motif search. In: Cai, L., Cheng, S.-W., Lam, T.-W. (eds.) ISAAC 2013. LNCS, vol. 8283, pp. 66–77. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45030-3_7