Information Processing Letters 180 (2023) 106333

Contents lists available at ScienceDirect

Information Processing Letters

journal homepage: www.elsevier.com/locate/ipl

Order-preserving pattern matching with scaling

Youngho Kim [a], Munseong Kang [b], Joong Chae Na [c,*], Jeong Seop Sim [a,*]

[a] Department of Computer Engineering, Inha University, Incheon 22212, South Korea
[b] Samsung Electronics, Suwon 16677, South Korea
[c] Department of Computer Science and Engineering, Sejong University, Seoul 05006, South Korea

ARTICLE INFO

Article history:
Received 10 September 2021
Received in revised form 3 April 2022
Accepted 2 October 2022
Available online 5 October 2022
Communicated by Elena Grigorescu

Keywords:
Analysis of algorithms
String matching
Approximate string matching
Order-preserving pattern matching

ABSTRACT

Given a text T and a pattern P, the order-preserving pattern matching (OPPM for short) problem is to find all substrings of T which have the same relative orders as P. Recently, approximate OPPM that allows errors have been studied such as OPPM with k-mismatches. In this paper we define the OPPM *with scaling* which is a novel criteria for approximate OPPM by considering the relative orders of cusps and the scale of lengths of strings between them. Also we present an algorithm to solve the OPPM problem with scaling in $O(n + m \log m)$ time, which is the same time bound as the best known exact OPPM algorithm.

1. Introduction

The order-preserving pattern matching (OPPM for short) compares the relative orders of characters instead of their exact values. When two strings X and Y of the same length have the same relative orders, they are called *order-isomorphic*. Given a text T of length n and a pattern P of length m, the OPPM problem is to find all substrings of T which are order-isomorphic to P. For example, when $T = (4, 3, 8, 1, 2)$ and $P = (4, 6, 1)$ are given, the order of individual characters in the substring $T[1..3] = (3, 8, 1)$ of T is $(2, 3, 1)$, which is the same as the order of characters in P. The OPPM problem can be solved in $O(n + m \log m)$ time for general alphabets and in $O(n+m)$ time for linear-time sortable alphabets [1–3]. Fast algorithms on average have been presented in [4,5]. Also, the order-preserving suffix tree has been proposed in [6].

Recently, approximate OPPM problems that allow errors in matching have been studied. Gawrychowski and Uznański [7] defined the OPPM problem with k-mismatches and presented an $O(n(\log \log m + k \log \log k))$-time algorithm. Chhabra et al. [8] presented an $O(nm(\lceil m/w \rceil + \log m))$-time algorithm and an $O(n(\lceil m/w \rceil \log \log w + m \log m))$-time algorithm for the OPPM problem with k-mismatches, where w is the computer word size.

In classical pattern matching, scaling approximation of strings has been studied. Given a text T and a pattern P, the pattern matching problem *with scaling* is to find all the positions where scaled P occurs in T [9]. For example, given $T = aaabbbbcccd$ and $P = abbc$, $T[1..8] = aabbbbcc$ matches with P scaled to 2. Amir et al. [9] proposed algorithms to solve one-dimensional and two-dimensional pattern matching problems with scaling in linear time for a pattern scaled by natural numbers. For the pattern matching problem scaled by real numbers, an $O(n)$-time algorithm using round down was presented in [10] and an $O(n \log m + m^{3/2}\sqrt{n \log m})$-time algorithm using round off was presented in [11]. For the two dimensional pattern matching problem scaled by real numbers, an $O(n^2 m)$-

* Corresponding authors.
E-mail addresses: yhkim85@inha.ac.kr (Y. Kim), kmsung0102@gmail.com (M. Kang), jcna@sejong.ac.kr (J.C. Na), jssim@inha.ac.kr (J.S. Sim).

https://doi.org/10.1016/j.ipl.2022.106333
0020-0190/

Table 1
Location tables for $X = (33, 24, 43, 30, 93, 28, 13, 81, 58)$.

i	0	1	2	3	4	5	6	7	8
$X[i]$	33	24	43	30	93	28	13	81	58
$LMax_X[i]$	−1	−1	0	1	2	1	−1	2	2
$LMin_X[i]$	−1	0	−1	0	−1	3	1	4	7

time algorithm using round off was presented in [12]. However, no study has been conducted yet for OPPM considering scaled patterns.

In this paper we first propose *order-preserving pattern matching with scaling* for natural numbers which is a novel criteria for approximate order-preserving pattern matching. Our contributions are as follows.

- We define the scaled order-isomorphism of two strings for the first time. In classical pattern matching, the scaling of a string is defined by duplicating each character by the scaled size. However, it is not straightforward to define the scaling of a string in OPPM for the following reasons. First, simply duplicating each character is not natural in order relations of a string. Second, if we add new characters between the two original characters, the ranks of some characters may change because of the added characters.
- We present an efficient algorithm for the OPPM problem with scaling. Our algorithm runs in $O(n+m\log m)$ time, which is the same time bound as the best known exact OPPM algorithm.

This paper is organized as follows. In Section 2, we give some basic terms and related works. In Section 3, we define the scaled order-isomorphism and the OPPM problem with scaling. We present an algorithm for the OPPM problem with scaling in Section 4 and we conclude in Section 5.

2. Preliminaries

Let Σ denote the set of characters where two characters can be compared in constant time. For a string S, let $|S|$ denote its length and $S[i]$ denote the ith $(0 \le i < |S|)$ character of S. The substring $S[i]\cdots S[j]$ $(0 \le i, j < |S|)$ is denoted by $S[i..j]$, and $S[i..j]$ is an empty string when $i > j$. The concatenation of two strings X and Y is denoted by $X \circ Y$. For convenience, we assume that characters in a string are all distinct as in [2].

We formally define the order-isomorphism and the nearest neighbor representation. If two strings X and Y of length m satisfy $X[i] < X[j] \Leftrightarrow Y[i] < Y[j]$ $(0 \le i, j < m)$, X and Y are *order-isomorphic*, which is denoted by $X \approx Y$. The order-isomorphism can be efficiently determined using the nearest neighbor representation [5,1–3]. The nearest neighbor representation of X is defined using the location tables $LMax_X$ and $LMin_X$, as follows.

$$LMax_X[i] = \begin{cases} j & \text{if } X[j] = \max\{X[k] : X[k] < X[i], \\ & \quad 0 \le k \le i-1\} \\ -1 & \text{if there is no such } j \end{cases}$$

$$LMin_X[i] = \begin{cases} j & \text{if } X[j] = \min\{X[k] : X[k] > X[i], \\ & \quad 0 \le k \le i-1\} \\ -1 & \text{if there is no such } j \end{cases}$$

That is, $LMax_X[i]$ stores the index j of the largest among the characters smaller than $X[i]$ in $X[0..i-1]$, and $LMin_X[i]$ stores the index j of the smallest among the characters larger than $X[i]$ in $X[0..i-1]$. Table 1 shows $LMax_X$ and $LMin_X$ for $X = (33, 24, 43, 30, 93, 28, 13, 81, 58)$. $LMax_X$ and $LMin_X$ can be computed in $O(m\log m)$ time using a sort algorithm or an order-statistic tree. When the location tables for X are given, the order-isomorphism of X and Y can be determined in $O(m)$ time by checking the following inequality for every $Y[i]$ $(0 \le i < m)$ [2,3]:

$$Y[LMax_X[i]] < Y[i] < Y[LMin_X[i]]. \tag{1}$$

The following lemma and corollary show some properties on determining the order-isomorphism of two strings.

Lemma 1. *For two strings X and Y of length m, let X' and Y' be the strings obtained from X and Y, respectively, by the same permutation. Then $X \approx Y \Leftrightarrow X' \approx Y'$.*

Proof. It is obvious by the definition of the order-isomorphism. □

Then, we can get the following corollary from Lemma 1.

Corollary 2. *For two strings X and Y of length m, $X[1..m-1] \circ X[0] \approx Y[1..m-1] \circ Y[0] \Leftrightarrow X \approx Y$.*

By Corollary 2 we can determine the order isomorphism of X and Y using the location tables of $X[1..m-1] \circ X[0]$ instead of X.

3. Scaled order-isomorphism

We first introduce some terminologies. For a string X of length ℓ, $X[i]$ $(1 \le i \le \ell - 2)$ is called a *cusp* if either $(X[i-1] < X[i]) \wedge (X[i] > X[i+1])$ or $(X[i-1] > X[i]) \wedge (X[i] < X[i+1])$. For simplicity of description, we regard the characters $X[0]$ and $X[\ell-1]$ as cusps. When $X = (1, 10, 6, 2, 7)$ in Fig. 1, cusps are 1, 10, 2, and 7. In addition, the interval between adjacent cusps is called a *run* whose length is called a *run-length*. For example, X consists of three runs whose lengths are 1, 2, and 1, respectively. We define the *cusp string* of X, denoted by X^C, as the concatenation of all the cusps of X in order. Also, we define the *run-length string* of X, denoted by X^R, as the sequence of all the run-lengths of X in order. That is, for $X = (1, 10, 6, 2, 7)$, $X^C = (1, 10, 2, 7)$ and $X^R = (1, 2, 1)$.

Definition 1. Two strings X and Y of the same length are cusped order-isomorphic if $X^C \approx Y^C$ and $X^R = Y^R$.

Observation 1. $\sum_{i=0}^{|X^R|-1} X^R[i] = |X| - 1$.

For example, X and Y_1 in Fig. 1 are cusped order-isomorphic. Note that we ignore $X[2]$ and $Y_1[2]$ because they are not cusps.

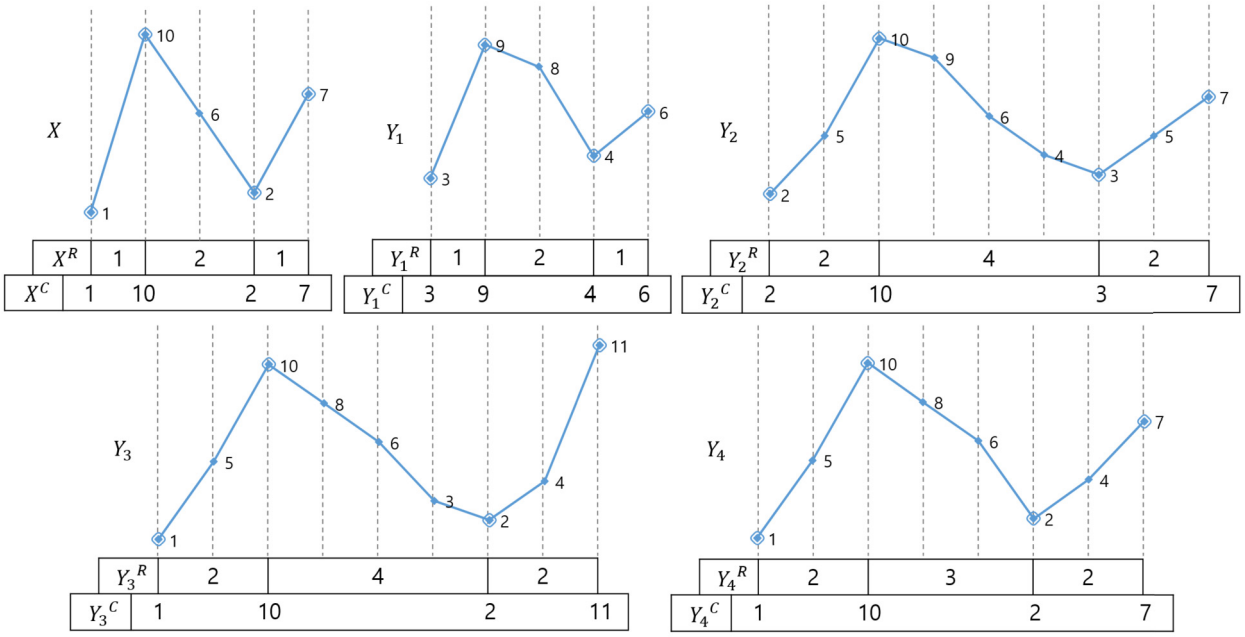

Fig. 1. X^R, X^C, Y_1^R, Y_1^C, Y_2^R, Y_2^C, Y_3^R, Y_3^C, Y_4^R, Y_4^C when the strings $X = (1, 10, 6, 2, 7)$, $Y_1 = (3, 9, 8, 4, 6)$, $Y_2 = (2, 5, 10, 9, 6, 4, 3, 5, 7)$, $Y_3 = (1, 5, 10, 8, 6, 3, 2, 4, 11)$, and $Y_4 = (1, 5, 10, 8, 6, 2, 4, 7)$.

Now we give an intuition on the scaled order-isomorphism. We mainly focus on cusps ignoring the other characters, more specifically, the relative orders of cusps and the run-lengths between them. For example, X in Fig. 1 is scaled order-isomorphic to Y_1. The cusps of X and those of Y_1 are order-isomorphic and the run-lengths of the two strings are the same. The string X in Fig. 1 is also scaled order-isomorphic to Y_2. The cusps of X and those of Y_2 are order-isomorphic, and each run-length of Y_2 is twice the corresponding run-length of X. Meanwhile, X is scaled order-isomorphic to neither Y_3 nor Y_4. The cusps of Y_3 are not order-isomorphic to those of X, and the ratios of the run-lengths of Y_4 to the corresponding run-lengths of X are not constant.

The formal definition of the scaled order-isomorphism is as follows.

Definition 2. Given two strings X of length m and Y of length n $(n \geq m)$, if the following conditions are satisfied for a positive integer $k \leq \lfloor \frac{n-1}{m-1} \rfloor$, X is k-scaled order-isomorphic to Y, denoted by $X \cong^k Y$.
Condition 1: $|X^R| = |Y^R|$ and $k \times X^R[i] = Y^R[i]$ for all $0 \leq i < |X^R|$. (By Observation 1, $n = k(m-1)+1$.)
Condition 2: $X^C \approx Y^C$.

For example, $X \cong^1 Y_1$ and $X \cong^2 Y_2$ in Fig. 1. Meanwhile, X is not scaled order-isomorphic to Y_3 since Condition 2 is not satisfied ($X^C \not\approx Y_3^C$). Also, X is not scaled order-isomorphic to Y_4 since Condition 1 is not satisfied ($X^R = (1, 2, 1)$ and $Y_4^R = (2, 3, 2)$).

Problem 1. The order-preserving pattern matching (OPPM) problem with scaling.

Input: Two strings T of length n and P of length m $(n \geq m)$.
Output: Every position i in T such that $P \cong^k T[i..i+k(m-1)]$ $(0 \leq i \leq n-m)$ with some integer k $(1 \leq k \leq \lfloor \frac{n-1}{m-1} \rfloor)$.

4. Algorithm for the OPPM problem with scaling

In this section we give an algorithm to solve the OPPM problem with scaling in $O(n+m\log m)$ time. Let $|T^R| = n'$ and $|P^R| = m'$. Then, $|T^C| = n'+1$ and $|P^C| = m'+1$. Note that the j-th run in T $(0 \leq j \leq n'-1)$ starts with the character $T^C[j]$. Also, $T^C[j] = T[j']$ where $j' = \sum_{q=0}^{j-1} T^R[q]$. Since the cases when $m' \leq 3$ are trivial, we assume that $m' \geq 4$.

We define the quotient strings of P and T. The *quotient string* P^Q of P is the string of length $m'-1$ indicating the ratios between adjacent run-lengths of P, i.e., the ratios between adjacent characters in P^R as follows:

$$P^Q = \left(\frac{P^R[0]}{P^R[1]}, \frac{P^R[1]}{P^R[2]}, \ldots, \frac{P^R[m'-2]}{P^R[m'-1]} \right)$$

The quotient string of T is defined similarly. Note that if a string is scaled by some integer k $(k \geq 1)$, the ratios between adjacent run-lengths of the string are conserved regardless of k. For instance, assume $X \cong^2 Y$ for two strings X and Y such that $X^R = (1, 2, 1, 2)$ and $Y^R = (2, 4, 2, 4)$. Then the quotient strings of X and Y are $(1/2, 2/1, 1/2)$ and $(2/4, 4/2, 2/4)$ respectively, which are the same.

Our algorithm uses T^R, P^R, T^C, P^C, T^Q, and P^Q. We can construct T^R, P^R, T^C, and P^C by scanning T and P, and can construct T^Q and P^Q by scanning T^R and P^R, respectively. All these strings can be constructed in $O(n+m)$ time using $O(n+m)$ space.

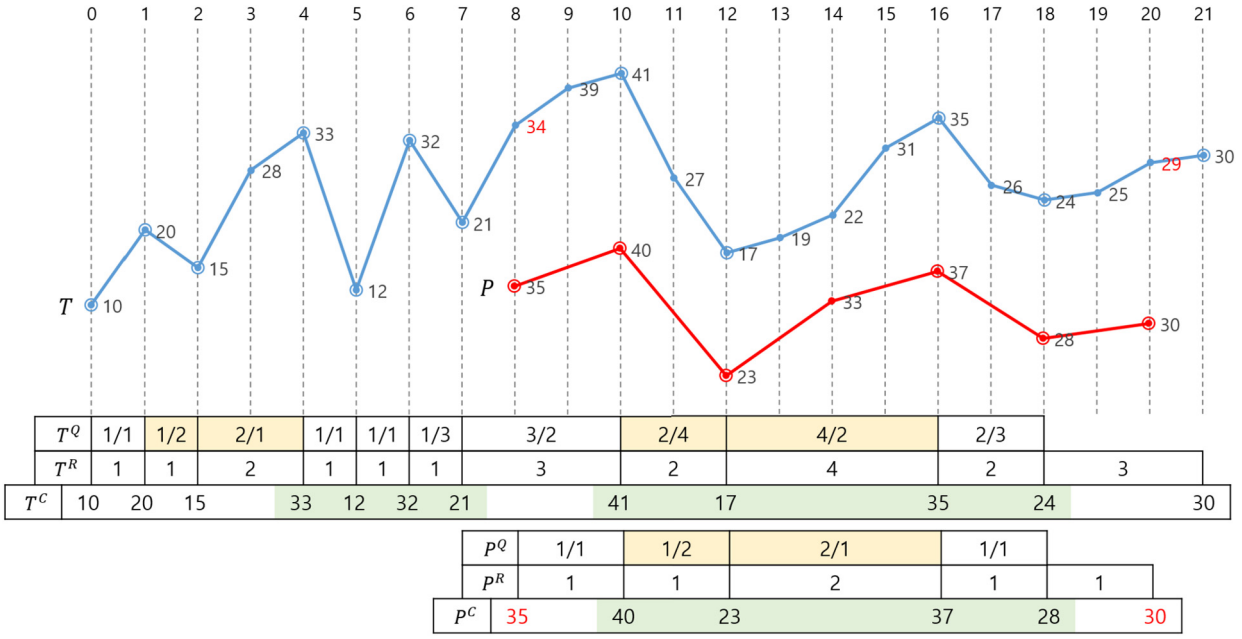

Fig. 2. An example of order-preserving pattern matching with scaling.

Our algorithm for the OPPM problem with scaling consists of two steps. Let the *center string* of P be the substring of P excluding the first $P^R[0]$ characters and the last $P^R[m'-1]$ characters from P. In Step 1, we search for the candidate substrings w of T to which P can be scaled order-isomorphic by considering only the center string of P. In Step 2, we check if P is scaled order-isomorphic to each w by considering the characters excluding the center string of P. The reason why we handle the center string and the other characters separately is that while the cusps of the center string of P correspond to the cusps of T, the first and last characters of P may not correspond to any cusp of T. For instance, consider $T[8..20]$ in Fig. 2, to which P is scaled order-isomorphic. The first character $P[0]$ and the last character $P[6]$ correspond to $T[8]$ and $T[20]$, respectively, both of which are not cusps of T.

The detailed algorithm is as follows.

- **Step 1**: Find every candidate substring w of T to which P can be scaled order-isomorphic considering only the center string of P. For each condition of Definition 2, we search every position that satisfies the condition.
  - **Finding the set J_1 of every position in T^R that satisfies Condition 1**: We find every position $j_1 \in J_1$ such that $P^Q[1..m'-3]$ (the quotient string of the center string of P) occurs exactly at j_1 in T^Q and the scale ratio $k = T^R[j_1]/P^R[1]$ is an integer.
    Note that we do not know the scale ratio k which may be different at each occurrence. Thus, instead of using T^R and P^R to search j_1, we use the quotient strings because as mentioned before, if a string is scaled by an integer k, the ratios between adjacent run-lengths are conserved regardless of k. That is, if $T^R[j_1+\ell-1] = kP^R[\ell]$ $(1 \le \ell \le m'-2)$ for an integer k, $T^Q[j_1..j_1+m'-4] = P^Q[1..m'-3]$.
    In the example of Fig. 2 when $m'=5$, the occurrences of $P^Q[1..2] = (1/2, 2/1)$ are $T^Q[1..2]$ $(k=1)$ and $T^Q[7..8]$ $(k=2)$, i.e., $J_1 = \{1, 7\}$.
  - **Finding the set J_2 of every position in T^C that satisfies Condition 2**: We find every position $j_2 \in J_2$ such that $T^C[j_2..j_2+m'-2]$ is order-isomorphic to $P^C[1..m'-1]$ (the cusp string of the center string of P).
    In the example of Fig. 2, $T^C[3..6]$ and $T^C[7..10]$ are order-isomorphic to $P^C[1..4]$, i.e., $J_2 = \{3, 7\}$.

  For each $j \in J_1 \cap J_2$, we determine the starting position s and ending position e of the candidate substring $w = T[s..e]$ of T. Obviously, $e = s + k(m-1)$ where k is the scale ratio $T^R[j]/P^R[1]$. Let j' be the position in T corresponding to $T^C[j]$. Then $j' = \sum_{q=0}^{j-1} T^R[q]$ as explained before. Since the scale ratio of the first run of w to that of P also should be k, $s = j' - kP^R[0]$. In the example of Fig. 2, when $j=7$, $k = T^R[7]/P^R[1] = 2$, $s = \sum_{q=0}^{6} T^R[q] - 2P^R[0] = 10 - 2 = 8$, and $e = 8 + 2 \times 6 = 20$. That is, $T[8..20]$ is a candidate.
- **Step 2**: Determine whether P is scaled order-isomorphic to each candidate $w = T[s..e]$ found in Step 1 or not, by also considering the first and the last runs of P.
  - **Verifying if Condition 1 is satisfied for w^R and P^R**: Check whether $T^R[j-1] \ge kP^R[0]$ and $T^R[j+m'-2] \ge kP^R[m'-1]$.
    To verify Condition 1, we should check if $|w^R| = |P^R|$ and $w^R[i] = kP^R[i]$ $(0 \le i < m')$. Since we already know that Condition 1 satisfies for $P^R[1..m'-2]$ by Step 1, the remaining parts to check are $P^R[0]$ and $P^R[m'-1]$. The substring of w corresponding to $P^R[0]$ is $w[0..r] = T[s..s+r]$, where $r = kP^R[0]$. Since the length of the run ending at $T[s+r]$ is $T^R[j-1]$, if $T^R[j-1] \ge r$, only one run exists in $T[s..s+r]$ and $w^R[0] = kP^R[0]$. By checking simi-

larly for the last run, we can check if Condition 1 is satisfied.

In the example of Fig. 2, a candidate $T[8..20]$ ($j = 7$, $k = 2$, and $m' = 5$) satisfies Condition 1, since $T^R[6]$ $(= 3) \geq kP^R[0]$ $(= 2)$ and $T^R[10]$ $(= 3) \geq kP^R[4]$ $(= 2)$,

– **Verifying if Condition 2 is satisfied for w^C and P^C:** Check the inequality (1) for the last two characters in $T^C[j..j+m'-2] \circ T[e] \circ T[s]$ using the location tables of $P^C[1..m'-1] \circ P^C[m'] \circ P^C[0]$.

Since Condition 1 is satisfied, $w^C = T[s] \circ T^C[j..j+m'-2] \circ T[e]$. We can verify Condition 2 ($w^C \approx P^C$) by checking if $T^C[j..j+m'-2] \circ T[e] \circ T[s] \approx P^C[1..m'-1] \circ P^C[m'] \circ P^C[0]$ by Corollary 2. Since $T^C[j..j+m'-2] \approx P^C[1..m'-1]$ by Step 1, we need to check the inequality (1) for the remaining characters $T[e]$ and $T[s]$ using the location tables of $P^C[1..m'-1] \circ P^C[m'] \circ P^C[0]$.

In the example of Fig. 2, Condition 2 is satisfied for candidate $T[8..20]$ and P, since the inequality (1) is satisfied for the last two characters of $T^C[j..j+m'-2] \circ T[e] \circ T[s] = (41, 17, 35, 24, \underline{29, 34})$ and $P^C[1..m'-1] \circ P^C[m'] \circ P^C[0] = (40, 23, \underline{37, 28, 30, 35})$.

In the example in Fig. 2, since $T[8..20]$ and P satisfy both conditions for $k = 2$, $P \cong^2 T[8..20]$.

Now we analyze the running time of our algorithm. First, let us consider Step 1. The set J_1 can be computed in $O(n+m)$ time using a pattern matching algorithm [13]. Also, the set J_2 can be computed in $O(n + m \log m)$ time using an OPPM algorithm [2,3]. In calculating the starting position s of every candidate w, computing $j' = \sum_{q=0}^{j-1} T^R[q]$ takes $O(n)$ time in total by processing every position j in increasing order. Therefore, Step 1 requires $O(n+m \log m)$ time in total. In Step 2, after computing the location tables of $P^C[1..m'] \circ P^C[0]$ in $O(m \log m)$ time, for each candidate w we can check Conditions 1 and 2 in $O(1)$ time. Since there are at most n candidates Step 2 requires $O(n + m \log m)$ time in total. Therefore, our algorithm takes $O(n + m \log m)$ time. Moreover, if we can sort the characters of P^C in linear time, we can compute the location tables used for checking Condition 2 in Steps 1 and 2 in $O(m)$ time [3]. Therefore, we obtain the following Theorem 3.

Theorem 3. *Given T ($|T| = n$) and P ($|P| = m$), the order-preserving pattern matching problem with scaling can be solved in $O(n + m \log m)$ time. If the characters of P^C can be sorted in linear time, the problem can be solved in $O(n + m)$ time.*

5. Conclusion

In this paper we defined the scaled order-isomorphism for the first time considering only cusps of strings while ignoring characters between them. Also we presented an efficient $O(n+m \log m)$-time algorithm for the OPPM problem with scaling. To apply OPPM to time series data analysis more effectively, it is necessary to study on diverse approximation in OPPM, for example, considering insertions, deletions, overlaps, etc.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (2020R1F1A1068873 & 2022R1G1A1012473), and by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean Government (MSIT) (2020-0-01389, Artificial Intelligence Convergence Research Center (Inha University) & No.RS-2022-00155915, Artificial Intelligence Convergence Innovation Human Resources Development (Inha University)), and by Inha University Research Grant.

References

[1] J. Kim, A. Amir, J.C. Na, K. Park, J.S. Sim, On representations of ternary order relations in numeric strings, Math. Comput. Sci. 11 (2) (2017) 127–136, https://doi.org/10.1007/s11786-016-0282-0.

[2] J. Kim, P. Eades, R. Fleischer, S. Hong, C.S. Iliopoulos, K. Park, S.J. Puglisi, T. Tokuyama, Order-preserving matching, Theor. Comput. Sci. 525 (2014) 68–79, https://doi.org/10.1016/j.tcs.2013.10.006.

[3] M. Kubica, T. Kulczyński, J. Radoszewski, W. Rytter, T. Waleń, A linear time algorithm for consecutive permutation pattern matching, Inf. Process. Lett. 113 (12) (2013) 430–433, https://doi.org/10.1016/j.ipl.2013.03.015.

[4] T. Chhabra, J. Tarhio, A filtration method for order-preserving matching, Inf. Process. Lett. 116 (2) (2016) 71–74, https://doi.org/10.1016/j.ipl.2015.10.005.

[5] S. Cho, J.C. Na, K. Park, J.S. Sim, A fast algorithm for order-preserving pattern matching, Inf. Process. Lett. 115 (2) (2015) 397–402, https://doi.org/10.1016/j.ipl.2014.10.018.

[6] M. Crochemore, C.S. Iliopoulos, T. Kociumaka, M. Kubica, A. Langiu, S.P. Pissis, J. Radoszewski, W. Rytter, T. Waleń, Order-preserving indexing, Theor. Comput. Sci. 638 (2016) 122–135, https://doi.org/10.1016/j.tcs.2015.06.050.

[7] P. Gawrychowski, P. Uznański, Order-preserving pattern matching with k mismatches, Theor. Comput. Sci. 638 (2016) 136–144, https://doi.org/10.1016/j.tcs.2015.08.022.

[8] T. Chhabra, E. Giaquinta, J. Tarhio, Filtration algorithms for approximate order-preserving matching, in: C.S. Iliopoulos, S.J. Puglisi, E. Yilmaz (Eds.), String Processing and Information Retrieval - 22nd International Symposium, SPIRE 2015, London, UK, September 1-4, 2015, Proceedings, in: Lecture Notes in Computer Science, vol. 9309, Springer, 2015, pp. 177–187.

[9] A. Amir, G.M. Landau, U. Vishkin, Efficient pattern matching with scaling, J. Algorithms 13 (1) (1992) 2–32, https://doi.org/10.1016/0196-6774(92)90003-U.

[10] A. Amir, A. Butman, M. Lewenstein, Real scaled matching, Inf. Process. Lett. 70 (4) (1999) 185–190, https://doi.org/10.1016/S0020-0190(99)00060-5.

[11] A. Amir, A. Butman, M. Lewenstein, E. Porat, D. Tsur, Efficient one-dimensional real scaled matching, J. Discret. Algorithms 5 (2) (2007) 205–211, https://doi.org/10.1016/j.jda.2006.03.017.

[12] A. Amir, E. Chencinski, Faster two dimensional scaled matching, in: M. Lewenstein, G. Valiente (Eds.), Combinatorial Pattern Matching, 17th Annual Symposium, CPM 2006, Barcelona, Spain, July 5-7, 2006, Proceedings, in: Lecture Notes in Computer Science, vol. 4009, Springer, 2006, pp. 200–210.

[13] D.E. Knuth, J.H. Morris Jr., V.R. Pratt, Fast pattern matching in strings, SIAM J. Comput. 6 (2) (1977) 323–350, https://doi.org/10.1137/0206024.