Order-Preserving Pattern Matching with Partition

Youngjoon Kim[a], Joong Chae Na[b,*], Jeong Seop Sim[a,*]

[a]*Department of Computer Engineering, Inha University, Incheon 22212, South Korea*
[b]*Department of Computer Science and Engineering, Sejong University, Seoul 05006, South Korea*

Abstract

Two strings of the same length are order-isomorphic when they have the same relative orders. By utilizing the concept of order-isomorphism, various types of time series data, such as stock prices, energy consumption, annual temperature variance, and music melody, can be effectively analyzed. To perform a more meaningful analysis of time series data, approximate criteria for the order-isomorphism are necessary, considering diverse types of errors. In this paper, we introduce a novel approximation criterion for the order-isomorphism, called the partitioned order-isomorphism. We then propose an efficient $O(n + sort(m))$-time algorithm for the order-preserving pattern matching problem considering the criterion of partition.

Keywords:
order-preserving pattern matching, approximate order-preserving pattern matching, pattern matching with partition, string matching

1. Introduction

Two strings of the same length are *order-isomorphic* when they have the same relative orders. For example, strings $X = (5, 9, 7)$ and $Y = (23, 51, 47)$ are order-isomorphic because the relative order of individual characters in both strings is the same as $(1, 3, 2)$. Given a text T of length n and a pattern P of length m, the order-preserving pattern matching (OPPM) problem is finding all substrings of T which are order-isomorphic to P. The OPPM problem can be solved in $O(n + sort(m))$ time [1, 2, 3] based on the KMP algorithm [4], where $sort(m)$ is time for sorting P. Practical algorithms for the OPPM problem using the Horspool approach and a filtration method

*Corresponding authors
Email addresses: yjkim12130@gmail.com (Youngjoon Kim), jcna@sejong.ac.kr (Joong Chae Na), jssim@inha.ac.kr (Jeong Seop Sim)

have been studied [5, 6]. Also, the order-preserving string regularities [7] and the order-preserving suffix trees [8] have been studied.

There has been vigorous study on approximate OPPM. To effectively analyze various types of time series data, such as stock prices, energy consumption, annual temperature variance, and music melody, OPPM can be applied. However, it is not always easy to apply OPPM algorithms directly due to various factors such as differences in the measurement intervals of time series data, noise occurring in the measurement process, errors in the data transformation process, etc. Therefore, to perform a more meaningful pattern search, prediction, and analysis of time series data, approximate OPPM is necessary, considering diverse types of errors. Gawrychowski and Uznański [9] and Chhabra et al. [10] studied the OPPM problem with k-mismatches. Recently, Kim et al. [11] defined a scaling approximation and presented algorithms to solve the scaling approximation pattern matching problem.

In some cases in OPPM, it might be more effective to compare two strings part by part rather than as whole strings. Consider two strings $x = (1, 8, 3, 7, 5, 6, 4, 2)$ and $y = (3, 24, 8, 21, 15, 27, 12, 6)$ for example. The orders of these two strings are quite different, with $(1, 8, 3, 7, 5, 6, 4, 2)$ and $(1, 7, 3, 6, 5, 8, 4, 2)$, respectively. Despite their overall differences in order, however, when x and y are divided at the fifth position, the orders of the two divided parts of x and y are the same with $(1, 5, 2, 4, 3)$ and $(3, 2, 1)$, respectively, i.e., the two respective parts match each other. This approach can be applied to plagiarism detection in music melodies.

In this paper, we introduce a novel approximation criterion for the order-isomorphism, called the partitioned order-isomorphism. Then we define the OPPM problem with partition and propose an efficient algorithm for the problem using the Z-function [7, 12]. Our algorithm runs in $O(n + sort(m))$ time which are identical to the algorithms [1, 2, 3] for the exact OPPM problem.

This paper is organized as follows. In Section 2, we give some preliminary works. In Section 3, we first define the partitioned order-isomorphism and the OPPM problem with partition formally. And then we give an efficient algorithm for the OPPM problem with partition. In Section 4, we conclude.

Table 1: Location tables and Z for $X = (5, 11, 18, 7, 3, 9)$

i	1	2	3	4	5	6
$X[i]$	5	11	18	7	3	9
$LMax_X[i]$	-1	1	2	1	-1	4
$LMin_X[i]$	-1	-1	-1	2	1	2
$Z(X)$	6	2	1	1	2	1

2. Preliminaries

Let Σ denote the set of characters where two characters can be compared in constant time. A string S is a sequence of characters drawn from Σ. We denote by $|S|$ the length of S and by $S[i]$ the ith character of S $(1 \leq i \leq |S|)$. The substring $S[i] \cdots S[j]$ $(1 \leq i, j \leq |S|)$ is denoted by $S[i..j]$ and $S[i..j]$ is an empty string when $i > j$. Substrings $S[1..i]$ and $S[i..|S|]$ $(1 \leq i \leq |S|)$ are called a prefix and a suffix of S, respectively. The concatenation of two strings X and Y is denoted by XY. For convenience, we assume that characters in a string are all distinct as in [1].

Two strings X and Y of the same length are *order-isomorphic*, denoted by $X \approx Y$, when $X[i] < X[j] \Leftrightarrow Y[i] < Y[j]$ for all $1 \leq i, j \leq |X|$. Obviously, the following lemma and corollary hold by the definition of the order-isomorphism.

Lemma 1. *For two strings X and Y of the same length, if X and Y are order-isomorphic, substrings $X[i..j]$ and $Y[i..j]$ (for all $1 \leq i \leq j \leq |X|$) are also order-isomorphic.*

Corollary 2. *For two strings X and Y of the same length, if there exist substrings $X[i..j]$ and $Y[i..j]$ $(1 \leq i \leq j \leq |X|)$ which are not order-isomorphic, X and Y are not order-isomorphic.*

The order-isomorphism can be efficiently determined using the nearest neighbor representation [5, 1, 3, 2], which is represented by the location tables $LMax_X$ and $LMin_X$.

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

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

That is, $LMax_X[i]$ and $LMin_X[i]$ are the nearest neighbors of $X[i]$ in the sorted sequence of characters in $X[1..i]$. Table 1 shows $LMax_X$ and $LMin_X$ for $X = (5, 11, 18, 7, 3, 9)$. The locations tables can be computed in $O(|X| \log |X|)$ time using a sorting algorithm or an order-statistic tree. Using the location tables for X, we can determine the order-isomorphism of X and Y in $O(|X|)$ time by checking the following inequality for every position $1 \leq i \leq |X|$:

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

(If $LMax_X[i]$ or $LMin_X[i]$ is equal to -1, we assume the respective inequality is true.) Also, we can find the longest prefix β of Y which is order-isomorphic to a prefix of X in $O(|\beta|)$ time by checking the above inequality in increasing order of i until we reach a position where the inequality is false.

For a string S, the Z-function $Z_i(S)$ $(1 \leq i \leq |S|)$ represents the length of the longest prefix of $S[i..|S|]$ which is order-isomorphic to a prefix of S [7]. It is a modified version for order-isomorphism of the Z-function introduced in [12] to solve classical string matching problems. With the nearest neighbor representation of S, $Z(S)$ can be computed in $O(|S|)$ time.

3. Order-preserving pattern matching with partition

In this section, we define order-preserving pattern matching problem with partition and present an efficient algorithm for solving the problem.

3.1. Problem Definition

We define the partitioned order-isomorphism. Let us consider two strings $X = (5, 11, 18, 7, 3, 9)$ and $Y = (1, 2, 3, 5, 4, 6)$, which are not order-isomorphic. When we partition X (resp. Y) into two substrings $X[1..3]$ and $X[4..6]$ (resp. $Y[1..3]$ and $Y[4..6]$), the partitioned substrings of X are order-isomorphic to those of Y, i.e., $X[1..3] \approx Y[1..3]$ and $X[3..6] \approx Y[3..6]$. Then, we say that X is partitioned order-isomorphic to Y. That is, the *partitioned order-isomorphism* is formally defined as follows.

Definition 1. *For two strings X and Y of length m and a position t $(1 \leq t \leq m)$, if $X[1..t] \approx Y[1..t]$ and $X[t+1..m] \approx Y[t+1..m]$, then X and Y are partitioned order-isomorphic at t, denoted by $X \approx_{\vdash}^{t} Y$. If there exists any position t such that $X \approx_{\vdash}^{t} Y$, then X and Y are partitioned order-isomorphic.*

In the example above, $X \approx^3_{\vdash} Y$ since $X[1..3] \approx Y[1..3]$ and $X[3..6] \approx Y[3..6]$. However, X is not partitioned order-isomorphic to $Z = (1, 2, 3, 4, 5, 6)$ because there exists no t such that $X \approx^t_{\vdash} Z$. We call position t a *partition position*. (Although we consider the cases when a string is partitioned into two substrings, Definition 1 can be easily extended to the cases of partitioning into more than two substrings.)

The following lemma shows that partition positions are consecutive.

Lemma 3. *For two strings X and Y of length m, let l_{max} be the largest l such that $X[1..l] \approx Y[1..l]$ $(1 \leq l \leq m)$. Similarly, let r_{min} be the smallest r such that $X[r..m] \approx Y[r..m]$ $(1 \leq r \leq m)$. Then, $X \approx^t_{\vdash} Y$ if and only if $r_{min} - 1 \leq t \leq l_{max}$.*

Proof. Note that l_{max} and r_{min} always exist since two strings of length 1 are order-isomorphic. ($\Rightarrow$) We show by contradiction that if $X \approx^t_{\vdash} Y$, $r_{min} - 1 \leq t \leq l_{max}$. Suppose that $X \approx^t_{\vdash} Y$ for some t such that $t < r_{min} - 1$ or $t > l_{max}$. Consider the case when $t > l_{max}$. Since $X \approx^t_{\vdash} Y$, $X[1..t] \approx Y[1..t]$, which contradicts the definition of l_{max}. Similarly, a contradiction occurs when $t < r_{min} - 1$.
($\Leftarrow$) We show that $X \approx^t_{\vdash} Y$ if $r_{min} - 1 \leq t \leq l_{max}$. Since $X[1..l_{max}] \approx Y[1..l_{max}]$ and $t \leq l_{max}$, $X[1..t] \approx Y[1..t]$ by Lemma 1. Similarly, $X[t + 1..m] \approx Y[t + 1..m]$. Therefore, $X \approx^t_{\vdash} Y$.

Corollary 4. *If $l_{max} < r_{min} - 1$, X and Y are not partitioned order-isomorphic.*

The *order-preserving pattern matching problem with partition* is to find all substrings of a text T that is partitioned order-isomorphic to a pattern P, formally defined as follows.

Problem 1. *The order-preserving pattern matching (OPPM) problem with partition.*
Input*: A text T of length n and a pattern P of length m $(n \geq m)$.*
Output*: Every pair $(i, [a : b])$ where i is a position in T $(1 \leq i \leq n - m + 1)$ and $[a : b]$ is a position range in P $(1 \leq a \leq b \leq m)$ such that $T[i..i + m - 1] \approx^t_{\vdash} P$ for all $a \leq t \leq b$.*

For example, given $T = (13, 92, 34, 88, 77, 63, 37, 40, 70, 54, 35, 24)$ and $P = (54, 12, 38, 69, 45, 22)$, the output is $(2, [3 : 3])$ and $(6, [2 : 5])$.

3.2. An algorithm for the OPPM problem with partition

Let T_i denote the substring of T of length m starting at position i, i.e., $T_i = T[i..i + m - 1]$ $(1 \leq i \leq n - m + 1)$. Let L_i and R_i be the length of the longest prefix and the longest suffix of T_i

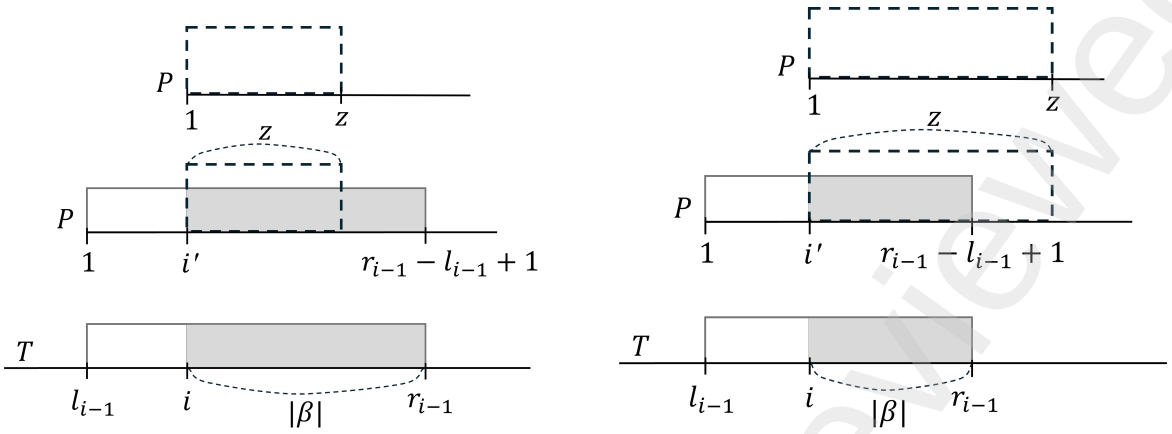
which are order-isomorphic to a prefix and a suffix of P, respectively. When we are given L_i's and R_i's for all $1 \leq i \leq n-m+1$, we can solve Problem 1 in $O(n)$ time by Lemma 3 and Corollary 4. (Note that L_i is l_{max} and R_i is $m - r_{min} + 1$ for T_i and P.)

Now we focus on the problem of computing all L_i's. (We omit the details of computing R_i's since it can be computed using the same way by considering reversed strings of T and P.) Using the nearest neighbor representation of P, we can compute each L_i in $O(m)$ time, resulting in a total time complexity of $O(nm+m\log m)$, where $O(m\log m)$ represents the time required for computing the nearest neighbor representation $LMax_P$ and $LMin_P$ of P. More efficiently, we can compute all L_i's in $O((n+m)\log(n+m))$ time by concatenating P and T (say $S = PT$) and computing $Z(S)$.

However, we can improve the time complexity to $O(n + m\log m)$ time by computing $Z(P)$ and all L_i's separately. We first preprocess P for computing $LMax_P$, $LMin_P$, and $Z(P)$. Then, we compute L_i in increasing order of i using $Z(P)$ and L_j $(1 \leq j < i)$, which is similar to the Z-algorithm in [12]. For any position k $(1 \leq k \leq n)$, the L-box starting at k is defined as the range $(k, k + L_k - 1)$ of length L_k. (Note that $L_k > 0$ because the first characters of two strings are trivially order-isomorphic). For each i $(1 \leq i \leq n)$, let (l_i, r_i) be the L-box whose end position is the rightmost among the L-boxes whose start position is less than or equal to i. (If more than one L-box has the rightmost end position, we may choose any L-box.)

At each iteration i $(1 \leq i \leq n-m+1)$, we compute L_i and (l_i, r_i) using (l_{i-1}, r_{i-1}). Initially, we set $(l_0, r_0) = (0, 0)$. The details of each iteration i is as follows.

1. If $i > r_{i-1}$, then compute L_i by explicitly checking the inequality (1) for T_i $(= T[i..i+m-1])$ and P character by character, starting from the first position and continuing until either the inequality becomes false or the check reaches the last position. Since $i > r_{i-1}$, we set (l_i, r_i) to $(i, i + L_i - 1)$, i.e., the new L-box starting at position i.
2. If $i \leq r_{i-1}$, then $T[i]$ is contained in substring $T[l_{i-1}..r_{i-1}]$ such that $T[l_{i-1}..r_{i-1}] \approx P[1..r_{i-1}-l_{i-1}+1]$, which means $T[i..r_{i-1}] \approx P[i-l_{i-1}+1..r_{i-1}-l_{i-1}+1]$ by Lemma 1. Let $\beta = T[i..r_{i-1}]$, $i' = i - l_{i-1} + 1$ and $z = Z_{i'}(P)$. Then, $\beta \approx P[i'..i' + |\beta| - 1]$ and $P[i'..i' + z - 1] \approx P[1..z]$ (by the definition of z). Thus, $\beta[1..\ell] \approx P[1..\ell]$ where $\ell = \min(|\beta|, z)$, i.e., it is guaranteed that L_i is at least $\min(|\beta|, z)$. See Figure 1.
   (a) If $z < |\beta|$, then $T_i[z+1]$ is contained in β and $T_i[1..z+1] \approx P[i'..i'+z]$. Since



(i) Case 2(a) (ii) Case 2(b)

Figure 1: Illustration of several cases under the condition $i \leq r_{i-1}$.

$P[i'..i'+z] \not\approx P[1..z+1]$ by the definition of $z = Z_{i'}(P)$, $T_i[1..z+1] \not\approx P[1..z+1]$ and thus $L_i = z$. Also, since the end position $i+z-1$ of the new L-box starting at position i is to the left of r_{i-1} $(= i + |\beta| - 1)$, we set (l_i, r_i) as (l_{i-1}, r_{i-1}).

(b) If $z > |\beta|$, then $P[|\beta|+1]$ is contained in $P[1..z]$ and $P[1..|\beta|+1] \approx P[i'..i'+|\beta|]$. Since $P[i'..i'+|\beta|] \not\approx T_i[1..|\beta|+1]$ by the definition of β, $T_i[1..|\beta|+1] \not\approx P[1..|\beta|+1]$ and thus $L_i = |\beta|$. Also, since the end position $i+|\beta|-1$ of the new L-box starting at position i is equal to r_{i-1}, we set (l_i, r_i) as (l_{i-1}, r_{i-1}) or $(i, i+L_i-1)$.

(c) If $z = |\beta|$, then L_i can be larger than $z = r_{i-1} - i + 1$. Thus, we compute L_i by explicitly checking the inequality (1) for T_i and P character by character, starting from the position $z+1$ and continuing until either the inequality becomes false or the check reaches the last position. Also, since the end position $i + L_i - 1$ of the new L-box starting at position i is to the right of or equal to r_{i-1} $(= i + |\beta| - 1)$, we set (l_i, r_i) as $(i, i+L_i-1)$.

We analyze the running time for computing all L_i's. Each iteration takes constant time except for checking the inequality (1). The check ends when the inequality becomes false or the check reaches the last position. Consider the number x_i of times when the inequality is true at iteration i. If $x_i > 0$, r_i increases by x_i (and never decreases) from r_{i-1} (Cases 1 and 2(c)). Since $r_i \leq n$, the total sum of x_i's for all iterations is at most n. Therefore, given $LMax_P$, $LMin_P$, and $Z(P)$, all L_i's can be computed in $O(n)$ time.

Now we analyze the time complexity of our algorithm. We can compute $LMax_P$ and $LMin_P$ in $O(m \log m)$ time. (If we can sort the characters of P in linear time, for example, the characters are drawn from an integer alphabet, we can compute $LMax_P$ and $LMin_P$ in $O(m)$ time.) Then, $Z(P)$ and L_i's can be computed in $O(m)$ time and $O(n)$ time, respectively. Also, R_i's can be computed similarly using the reversed strings of P and T. For each position i $(1 \leq i \leq n-m+1)$, given L_i and R_i, we can determine whether T_i is partitioned isomorphic to P and compute the range of partition positions in constant time. Therefore, we obtain the following theorem.

Theorem 5. *Given T of length n and P of length m, the order-preserving pattern matching problem with partition can be solved in $O(n + m \log m)$ time. If the characters of P can be sorted in linear time, the problem can be solved in $O(n + m)$ time.*

4. Conclusion

We have introduced the concept of partitioned order-isomorphism as a novel criterion for approximate order-isomorphism. This approximation criterion is significant not only because it can be utilized in analyzing time series data where errors may occur, but also because it provides a criterion for approximation that is difficult to consider in classical pattern matching based solely on comparing character values. Using this criterion, we have defined the OPPM problem with partition and presented an efficient algorithm for solving it. Although we currently focus on the case where a string is partitioned into two substrings, our definition can readily be extended to cases where the string is partitioned into three or more substrings.

Acknowledgements

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean Government (MSIT) (No.RS-2022-00155915, Artificial Intelligence Convergence Innovation Human Resources Development (Inha Univer-sity)), by INHA UNIVERSITY Research Grant, and by the faculty research fund of Sejong University in 2024.

References

[1] 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. doi:10.1016/j.tcs.2013.10.006.
URL https://doi.org/10.1016/j.tcs.2013.10.006

[2] 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. doi:10.1016/j.ipl.2013.03.015.
URL https://doi.org/10.1016/j.ipl.2013.03.015

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

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

[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. doi:10.1016/j.ipl.2014.10.018.
URL https://doi.org/10.1016/j.ipl.2014.10.018

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

[7] M. M. Hasan, A. S. M. S. Islam, M. S. Rahman, M. S. Rahman, Order preserving pattern matching revisited, Pattern Recognit. Lett. 55 (2015) 15–21. doi:10.1016/j.patrec.2014.11.013.
URL https://doi.org/10.1016/j.patrec.2014.11.013

[8] 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. doi:10.1016/j.tcs.2015.06.050.
URL https://doi.org/10.1016/j.tcs.2015.06.050

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

[10] 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, Vol. 9309 of Lecture Notes in Computer Science, Springer, 2015, pp. 177–187. doi:10.1007/978-3-319-23826-5_18.
URL https://doi.org/10.1007/978-3-319-23826-5_18

[11] Y. Kim, M. Kang, J. C. Na, J. S. Sim, Order-preserving pattern matching with scaling, Information Processing Letters 180 (2023) 106333. doi:https://doi.org/10.1016/j.ipl.2022.106333.
URL https://www.sciencedirect.com/science/article/pii/S0020019022000904

[12] D. Gusfield, Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology,

Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.
URL https://doi.org/10.1017/cbo9780511574931