

Almost Order Preserving Matching

Wen-Wen Liao

Dept. Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan

Chang-Biau Yang

Dept. Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan
correspondence: cbyang@cse.nsysu.edu.tw

Abstract

In this paper, we define a new problem, called the almost order-preserving matching (aOPM) problem, which is a combined variant of order-preserving matching (OPM) and the longest almost increasing subsequence (LaIS). Given a numeric pattern P , a numeric text string T , and a tolerance constant c , the aOPM problem seeks to find the substring of T that each element has the almost same rank as the corresponding element in P . In an almost increasing sequence, small drops within c are permitted while still maintaining a generally increasing trend. Our algorithm for solving the aOPM problem consists of three stages: fingerprint, almost rank, and bipartite matching. The time complexity of the proposed aOPM algorithm is $O(n + k_1 m \log m + R + k_2 m^3)$, where $n = |T|$, $m = |P|$, R is the number of separation rank combinations, k_1 and k_2 denote the number of candidate substrings identified in the fingerprint and almost rank stages, respectively.

Keywords: almost increasing subsequence, order-preserving matching, fingerprint, string matching, bipartite matching

I. INTRODUCTION

The longest increasing subsequence (LIS) problem, introduced by Schensted [10], stands as a classic challenge in computer science. This problem aims to determine the longest subsequence with increasing order. For instance, consider the sequence $A = \langle 1, 12, 6, 9, 5, 7, 4 \rangle$. The possible LIS answers include $\langle 1, 6, 9 \rangle$, $\langle 1, 6, 7 \rangle$, or $\langle 1, 5, 7 \rangle$, each having a length of 3. Note that the LIS answer may not be unique.

The longest increasing subsequence (LIS) problem [10], [8], [2], [6], [1] is well-suited for cases with monotonically incremental or nondecremental sequences. However, real-world scenarios often involve situations that lack strict monotonicity. Take, for instance, the stock market where prices may alternate between rising and falling during a period, introducing non-monotonicity. Another example is temperature changes in real life. Although the temperature may exhibit an overall rising trend in recent years, it might not ascend continuously, potentially interspersed with fluctuations and noise. Thus, the

longest almost increasing subsequence (LaIS) problem was first introduced in 2010 by Elmastas [7].

In an almost increasing subsequence, a small drop (controlled by a predefined tolerance constant) is allowed for comparing to the maximal element that has appeared so far. The LaIS problem is a generalized variant of the LIS problem. Consider a sequence $A = \langle 1, 12, 6, 9, 5, 7, 4 \rangle$ and a tolerance constant $c = 3$, where c represents the acceptable drop being less than 3 in an almost increasing subsequences. Thus, the possible LaIS answers are $\langle 1, 6, 9, 7 \rangle$, the possible LaIS answers are $\langle 1, 6, 9, 7 \rangle$, $\langle 1, 6, 5, 7 \rangle$, or $\langle 1, 6, 5, 4 \rangle$, with length 4. In $\langle 1, 6, 9, 7 \rangle$, the overall trend is increasing, but there is a small drop between 9 and 7 with a drop of $9 - 7 = 2 < c = 3$. In $\langle 1, 6, 5, 7 \rangle$, we cannot append 4 after 7, since the drop $7 - 4 = 3$, which is not less than $c = 3$. The LaIS problem can be solved in $O(n \log l)$ time [7], where l is the length of the LaIS answer.

The order preserving matching (OPM) problem was first proposed by Kim *et al.* [9] in 2014. Given a numeric text T and a numeric pattern P , the OPM problem focuses on finding a substring of T such that the rank of each element is the same as that of each corresponding element in P . For example, suppose $T = \langle 23, 12, 9, 28, 17, 14, 15 \rangle$ and $P = \langle 7, 6, 11, 10, 9 \rangle$. The solution of OPM is $\langle 12, 9, 28, 17, 14 \rangle$, where each element has the identical relative order as the corresponding element of P . Order preserving matching has applications in various fields, including data analysis, pattern recognition, and bioinformatics, where identifying substrings with shared rank is crucial.

The related studies on the OPM problems are summarized in Table I.

In daily life, the precise knowledge of sequence order may not always be necessary and only an approximate trend is required. For instance, consider the stock price series shown in Figure 1. Instead of needing exact sequences of price movements, we may only need to identify approximate trends, such as whether the market is generally bullish or bearish during a certain period. Thus, we propose the concept of almost order preserving matching (aOPM). Besides finance, aOPM finds applications in weather forecasting, where it emphasizes approximate trends like temperature fluctuations over days or weeks, rather than minute-by-minute predictions.

In this paper, we propose an efficient algorithm for solving the aOPM problem, which consists of three stages: fingerprint, almost rank, and bipartite matching. The time complexity is

TABLE I
THE PREVIOUS ALGORITHMS OF THE OPM PROBLEM, WHERE $m = |P|$ AND $n = T$.

The order preserving matching (OPM) problem			
Year	Author(s)	Time complexity	Note
2014	Kim <i>et al.</i> [9]	$O(n + m \log m)$	KMP
2015	Cho <i>et al.</i> [5]	$O(n + m)$	Boyer–Moore, fingerprint
2016	Chhabra and Tarhio [4]	average case: sublinear, worst case: $O(mn)$	fingerprint

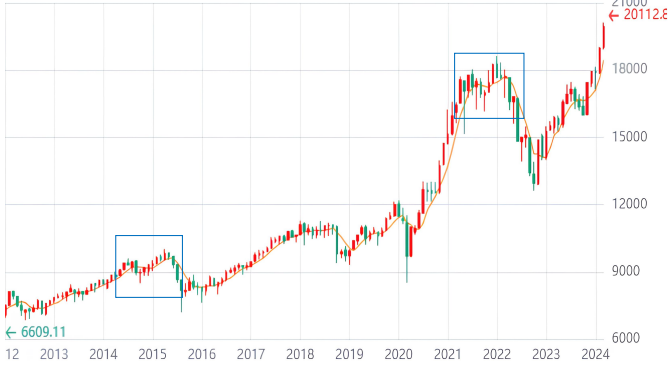


Fig. 1. An example of stock prices in Taiwan from 2012 to 2024. In the analysis of stock prices, employing aOPM would result in an approximate matching. This approach allows for slight increases or decreases in stock prices, rather than focusing solely on exact sequences of price movements.

$O(n + k_1 m \log m + R + k_2 m^3)$, where $n = |T|$, $m = |P|$, R is the number of separation rank combinations, k_1 and k_2 denote the number of candidate substrings identified in the fingerprint and almost rank stages, respectively.

The rest of the paper is organized as follows. Section II first introduces the LIS problem and its variants, then formally defines the aOPM problem. Section III presents our algorithms for solving the aOPM problem. Finally, we summarize our conclusion in Section IV.

II. PRELIMINARIES

A. Longest Increasing Subsequence (LIS)

The LIS problem, proposed by Schensted in 1961 [10], aims to find the longest subsequence in a given sequence that is strictly increasing. The time complexity of his algorithm is $O(n \log n)$, where n denotes the length of the input sequence. If the input is a permutation of $\{1, 2, \dots, n\}$, with subsequent improvements using the data structure of van Emde Boas tree [12], the time complexity was reduced to $O(n \log \log n)$ or $O(n \log \log l)$ [8], [2], [6], where l represents the answer length.

For example, given a sequence $A = \langle 1, 12, 6, 9, 5, 7, 4 \rangle$, the possible LIS answers are $\langle 1, 6, 9 \rangle$, $\langle 1, 6, 7 \rangle$, or $\langle 1, 5, 7 \rangle$, with length 3.

B. Longest Almost Increasing Subsequence (LaIS)

The *longest almost increasing subsequence* (LaIS) problem, introduced by Elmasry in 2010 [7], relaxes the strictly

increasing property of the LIS problem. LaIS allows for a small drop for each element in the answer, where the drop is determined by a predefined tolerance constant c .

Given a sequence $A = \langle a_1, a_2, \dots, a_n \rangle$, along with a tolerance constant c , we say that A is an *almost increasing sequence* if $a_i + c > \max\{a_{i'} | 1 \leq i' \leq i - 1\}$, for $2 \leq i \leq n$. The LaIS problem seeks to identify an almost increasing subsequence of a given sequence with the maximum length.

For instance, given a sequence $A = \langle 1, 12, 6, 9, 5, 7, 4 \rangle$ and $c = 3$, the possible LaIS answers are $\langle 1, 6, 9, 7 \rangle$, $\langle 1, 6, 5, 7 \rangle$, or $\langle 1, 6, 5, 4 \rangle$, with length 4.

C. Order Preserving Matching (OPM)

Kim *et al.* in 2014 [9] proposed the *order-preserving matching* (OPM) problem, defined as follows.

Definition 1. (order preserving matching, OPM)[9] *Given a numeric text $T = \langle t_1, t_2, \dots, t_n \rangle$ and a numeric pattern $P = \langle p_1, p_2, \dots, p_m \rangle$, the order preserving matching (OPM) problem is to determine whether there exists a substring $T_{i..i+m-1}$ such that $\text{rank}(t_{i+j-1}, T_{i..i+m-1}) = \text{rank}(p_j, P)$, for $1 \leq j \leq m$. Here, $\text{rank}(e, E)$ means the rank of element e within sequence E . That is, $\text{rank}(e, E) = |\{e' | e' \leq e, e' \in E\}|$.*

For example, for a given numeric text $T = \langle 23, 12, 9, 28, 17, 14, 15 \rangle$ and a numeric pattern $P = \langle 7, 6, 11, 10, 9 \rangle$, the OPM answer is $\langle 12, 9, 28, 17, 14 \rangle$.

D. Almost Order Preserving Matching (aOPM)

In this paper, we aim to extend the concept of OPM to a more generalized version, termed as *almost order preserving matching* (aOPM). The aOPM problem combines aspects of both LaIS and OPM. aOPM relaxes the exact rank to the almost rank by introducing a *tolerance constant* c . An example for illustrating the aOPM problem is shown in Figure 2.

Definition 2. (permutation)[3] *Given a numeric sequence $A = \langle a_1, a_2, \dots, a_n \rangle$, a permutation $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ operating on A is denoted as $\pi(A) = A' = \langle a'_1, a'_2, \dots, a'_n \rangle$, where $a'_i = a_{\pi_i}$.*

For example, consider a numeric sequence $A = \langle 7, 4, 11, 10, 5 \rangle$ and a permutation $\pi = (2, 1, 5, 3, 4)$. Then, the permuted sequence is $\pi(A) = A' = \langle 4, 7, 5, 11, 10 \rangle$.

Definition 3. (almost order preserving match, aOPM) *Given two numeric sequences $A = \langle a_1, a_2, \dots, a_n \rangle$ and $B =$*

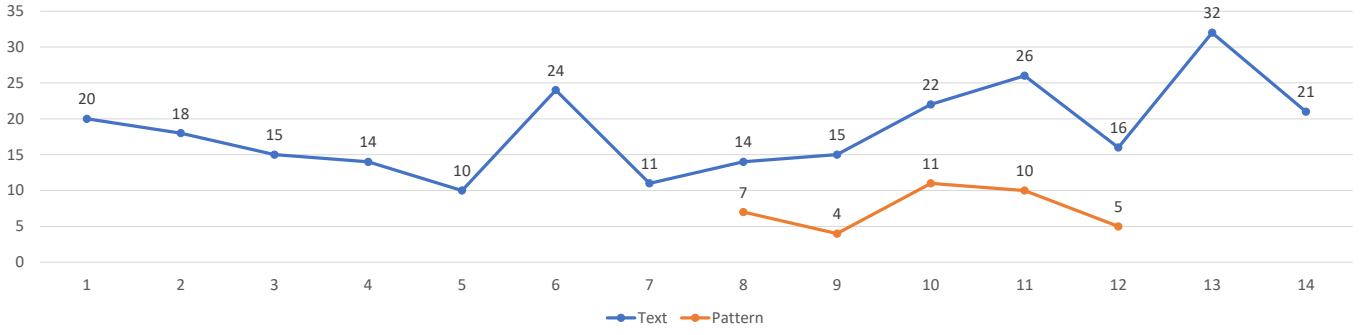


Fig. 2. An example of aOPM with a numeric text $T = \langle 20, 18, 15, 14, 10, 24, 11, 14, 15, 22, 26, 16, 32, 21 \rangle$, a numeric pattern $P = \langle 7, 4, 11, 10, 5 \rangle$ and tolerance constant $c = 3$. Here, 14 and 15 are in T , 7 and 4 are in P . $T_{8..12}$ is not a valid OPM answer, because $14 < 15$ and $7 > 4$. However, $T_{8..12}$ is an aOPM answer since the tolerance $c = 3$ is allowed (14 is almost greater than 16).

$\langle b_1, b_2, \dots, b_n \rangle$, along with a tolerance constant c , if there exists a permutation π of $1, 2, \dots, n$, such that $\pi(A)$ and $\pi(B)$ are both almost increasing, then it is an almost order preserving match (aOPM) between A and B .

For example, consider two sequences $A = \langle 7, 4, 11, 10, 5 \rangle$ and $B = \langle 14, 15, 22, 26, 16 \rangle$, along with a tolerance constant $c = 3$. There exists a permutation $\pi = (2, 1, 5, 3, 4)$ that $\pi(A) = \langle 4, 7, 5, 11, 10 \rangle$ and $\pi(B) = \langle 15, 14, 16, 22, 26 \rangle$ are both almost increasing. Therefore, A and B is an aOPM. There are two other permutations π , $(2, 5, 1, 3, 4)$ and $(5, 2, 1, 3, 4)$ that make $\pi(A)$ and $\pi(B)$ are both almost increasing.

Definition 4. (aOPM problem) Given a numeric text $T = \langle t_1, t_2, \dots, t_n \rangle$ and a numeric pattern $P = \langle p_1, p_2, \dots, p_m \rangle$, where $t_i, p_j \in \Sigma$ and Σ is an alphabet, the almost order preserving matching (aOPM) problem is to determine whether there exists a substring $T_{i..i+m-1}$, $1 \leq i \leq n - m + 1$, such that there is an aOPM between $T_{i..i+m-1}$ and P .

For example, in Figure 2, there is an aOPM between $T_{8..12}$ and P .

III. OUR ALGORITHM

For solving the aOPM problem, we need some definitions as follows.

Definition 5. (almost full-rank) Given a numeric sequence $A = \langle a_1, a_2, \dots, a_n \rangle$ and a tolerance c , the almost full-rank (aFR) of an element a_i , $1 \leq i \leq n$, is an interval $fr(A, i) = [fr_{min}(A, i), fr_{max}(A, i)]$, where $fr_{min}(A, i) = \{i\} \cup \{j | a_j < a_i - c + 1, 1 \leq j \leq n\}$ means the minimal almost full-rank set and $fr_{max}(A, i) = \{i\} \cup \{j | a_j < a_i + c, 1 \leq j \leq n\}$ means the maximal almost full-rank set.

For example, given a sequence $A = \langle 7, 4, 11, 10, 5 \rangle$ and a tolerance constant $c = 3$, Table II shows an example for the aFR. $fr_{min}(A, 4) = \{1, 2, 4, 5\}$ corresponds to $\langle 7, 4, 10, 5 \rangle$, and $fr_{max}(A, 4) = \{1, 2, 3, 4, 5\}$ corresponds to $\langle 7, 4, 11, 10, 5 \rangle$.

We have to consider scenario where the two almost full-rank sets of two neighboring elements overlap, but they cannot be reversed. Table III shows an example of such a scenario.

TABLE II
AN EXAMPLE OF THE AFR WITH $A = \langle 7, 4, 11, 10, 5 \rangle$ AND $c = 3$.

$A \backslash \text{rank}$		fr_{min}	fr_{max}	$fr(A, i)$
a_1	7	$\{1, 2\}$	$\{1, 2, 5\}$	$[2, 3]$
a_2	4	$\{2\}$	$\{2, 5\}$	$[1, 2]$
a_3	11	$\{1, 2, 3, 5\}$	$\{1, 2, 3, 4, 5\}$	$[4, 5]$
a_4	10	$\{1, 2, 4, 5\}$	$\{1, 2, 3, 4, 5\}$	$[4, 5]$
a_5	5	$\{5\}$	$\{1, 2, 5\}$	$[1, 3]$

TABLE III
THE OVERLAPPING AFRS OF TWO NEIGHBORING ELEMENTS, WHILE THEY CANNOT BE REVERSED. HERE, $X = \langle 12, 14, 15, 17 \rangle$ AND $c = 4$.

$X \backslash \text{rank}$		$fr(X, i)$	$R_1(X, i)$	$R_2(X, i)$	$R_3(X, i)$
x_1	12	$[1, 3]$	$[1, 1]$	$[1, 2]$	$[1, 3]$
x_2	14	$[1, 4]$	$[1, 4]$	$[1, 4]$	$[1, 4]$
x_3	15	$[1, 4]$	$[1, 4]$	$[1, 4]$	$[1, 4]$
x_4	17	$[2, 4]$	$[2, 4]$	$[3, 4]$	$[4, 4]$

$fr(X, 1) = [1, 3]$ indicates x_1 may be put at position 1, 2, or 3. In addition, x_4 may also be placed at position 2, 3, or 4. If x_1 is placed at position 3, and x_4 at position 2, it would produce the permutation $\langle 14, 17, 12, 15 \rangle$, or $\langle 15, 17, 12, 14 \rangle$. However, neither of these permutations is almost increasing, since 17 and 12 are reversed. So, there should be a definite separation rank between x_1 and x_4 . In $R_1(\cdot)$, $R_2(\cdot)$ or $R_3(\cdot)$, x_1 and x_4 are restricted to not be reversed. In the following, an aFR is modified to have a definite separation rank between any pair of elements who cannot be reversed.

Definition 6. (almost full-rank match) Given two numeric sequences A and B , if $fr(A, i) \cap fr(B, i)$ is not empty for all $1 \leq i \leq n$, then there is an almost full-rank match (aFRM) between A and B .

For example, see Tables II and IV. $fr(A, 3) \cap fr(B, 3) = [4, 5] \cap [4, 4] = [4, 4]$. It can be observed that $fr(A, i) \cap fr(B, i) \neq \emptyset$ for $1 \leq i \leq 5$. Thus, there is an aFRM between A and B .

Furthermore, if there is an aFRM between A and B , we will decide whether there is an aOPM between A and B . In Tables II and IV, we can find a permutation π such that each $\pi_j \in$

TABLE IV
THE AFR OF $B = \langle 14, 15, 22, 26, 16 \rangle$ AND $c = 3$.

$B \backslash \text{rank}$		fr_{min}	fr_{max}	$fr(B, i)$	$fr(A, i) \cap fr(B, i)$
b_1	14	{1}	{1, 2, 5}	{1, 3}	{2, 3}
b_2	15	{2}	{1, 2, 5}	{1, 3}	{1, 2}
b_3	22	{1, 2, 3, 5}	{1, 2, 3, 5}	{4, 4}	{4, 4}
b_4	26	{1, 2, 3, 4, 5}	{1, 2, 3, 4, 5}	{5, 5}	{5, 5}
b_5	16	{5}	{1, 2, 5}	{1, 3}	{1, 3}

TABLE V
NO AOPM BETWEEN X AND Y , WHERE $X = \langle 12, 14, 15, 17 \rangle$ (TABLE III)
 $Y = \langle 27, 24, 25, 22 \rangle$, ALONG WITH $c = 4$.

$Y \backslash \text{rank}$		$fr(Y, i)$	$R_1(Y, i)$	$R_2(Y, i)$	$R_3(Y, i)$
y_1	27	{2, 4}	{2, 4}	{3, 4}	{4, 4}
y_2	24	{1, 4}	{1, 4}	{1, 4}	{1, 4}
y_3	25	{1, 4}	{1, 4}	{1, 4}	{1, 4}
y_4	22	{1, 3}	{1, 1}	{1, 2}	{1, 3}

$fr(A, i) \cap fr(B, i)$ for a distinct value of i . It can be observed that π might be $(2, 5, 1, 3, 4)$, $(2, 1, 5, 3, 4)$ or $(5, 2, 1, 3, 4)$. And, $\pi(A)$ and $\pi(B)$ are both almost increasing for each π . Therefore, there is an aOPM between A and B .

See Tables III and V. When the separation rank is applied, it is not true that $R_k(X, i) \cap R_{k'}(Y, i) \neq \emptyset$, for $1 \leq i \leq 4$ and for $1 \leq k, k' \leq 3$. Thus, there is no aOPM between X and Y .

Table VI shows that there is an aOPM between X and Z when the separation rank is applied. In this example, there exists $k = 2$ such that $R_k(X, i) \cap fr(Z, i) \neq \emptyset$, for $1 \leq i \leq 4$.

Based on the above observation, if there exists a certain i that $fr(A, i) \cap fr(B, i) = \emptyset$, then the permutation for arranging both sequences to be almost increasing cannot be found. Accordingly, there is no aOPM between A and B . This property is formally presented as follows.

Lemma 1. *Given two numeric sequences A and B , if there exists a certain value of i that $fr(A, i) \cap fr(B, i) = \emptyset$, there is no aOPM between A and B .*

By the logically reverse direction of Lemma 1, we can obtain the following theorem.

Theorem 1. *If there is an almost order-preserving match between sequences A and B , then there is an almost full-rank match between A and B .*

The algorithm for solving the aOPM problem can be divided into three stages as follows.

TABLE VI
THE AOPM BETWEEN X AND Z , WHERE $X = \langle 12, 14, 15, 17 \rangle$ (TABLE III)
 $Z = \langle 20, 40, 10, 30 \rangle$, ALONG WITH $c = 4$.

$Z \backslash \text{rank}$		$fr(Z, i)$	$R_2(X, i) \cap fr(Z, i)$
z_1	20	{2, 2}	{2, 2}
z_2	40	{4, 4}	{4, 4}
z_3	10	{1, 1}	{1, 1}
z_4	30	{3, 3}	{3, 3}

TABLE VII
THE NINE RESULTS OF ALMOST MATCHES IN FINGERPRINT COMPARISONS.

$g_P(i)$	$g_T(j)$	$g_P(i) \times g_T(j)$	match or not
1	1	1	match
1	0	0	match
1	-1	-1	not match
0	1	0	match
0	0	0	match
0	-1	0	match
-1	1	-1	not match
-1	0	0	match
-1	-1	1	match

TABLE VIII
THE FINGERPRINT AND JUMP TABLE OF $P = \langle 7, 4, 11, 10, 5 \rangle$ AND $c = 3$.

i	1	2	3	4	5
p_i	7	4	11	10	5
$g_P(i)$	0	-1	1	0	1
$jump[i]$	0	1	1	2	2

Stage 1 (fingerprint): Invoke a fingerprint method to filter out the impossible matching positions.

Stage 2 (almost full-rank): Employ the method for the almost full-rank match.

Stage 3 (bipartite matching): Apply the bipartite graph matching method to find the possible almost order-preserving match.

Definition 7. (fingerprint) *Given a numeric string $S = \langle s_1, s_2, \dots, s_n \rangle$ and the tolerance constant $c = 3$, the fingerprint of s_i , denoted as $g_S(i)$, is defined as follows.*

$$g_S(i) = \begin{cases} 0 & \text{if } i = 1, \\ 1 & \text{if } s_i > s_{i-1} + c - 1, \\ 0 & \text{if } s_{i-1} - c + 1 \leq s_i \leq s_{i-1} + c - 1, \\ -1 & \text{if } s_i < s_{i-1} - c + 1. \end{cases}$$

For example, given a string $S = \langle 7, 4, 11, 10, 5 \rangle$ and a constant $c = 3$, the fingerprint of S is $\langle 0, -1, 1, 0, -1 \rangle$.

Basically, our fingerprint algorithm is based on the KMP algorithm [11], widely used for the string pattern matching problem. The fingerprint value 0 means that it can be either increasing or decreasing. So, it is a match to each of -1 , 0 and 1 in the concept of ‘almost matching’. We present the nine possible matching results in Table VII. As one can see, it is a not match only if the multiplication is equal to -1 .

In stage 1, we build the jump table for the pattern P , as shown in Table VIII. The construction method is similar to the KMP algorithm [11] for the construction of the prefix function, or failure function, in the text string pattern matching algorithm.

Figure 3 shows an example for finding out the possible candidate substrings by fingerprints and the jump table.

After finding out one possible candidate in stage 1, we check whether it is aFRM in stage 2. Suppose the possible candidate is denoted by $T_{i-m+1..i}$. We can sort the elements in P and $T_{i-m+1..i}$, respectively. Then, the aFR of P can be obtained by checking the range of c . The aFR of $T_{i-m+1..i}$ can be

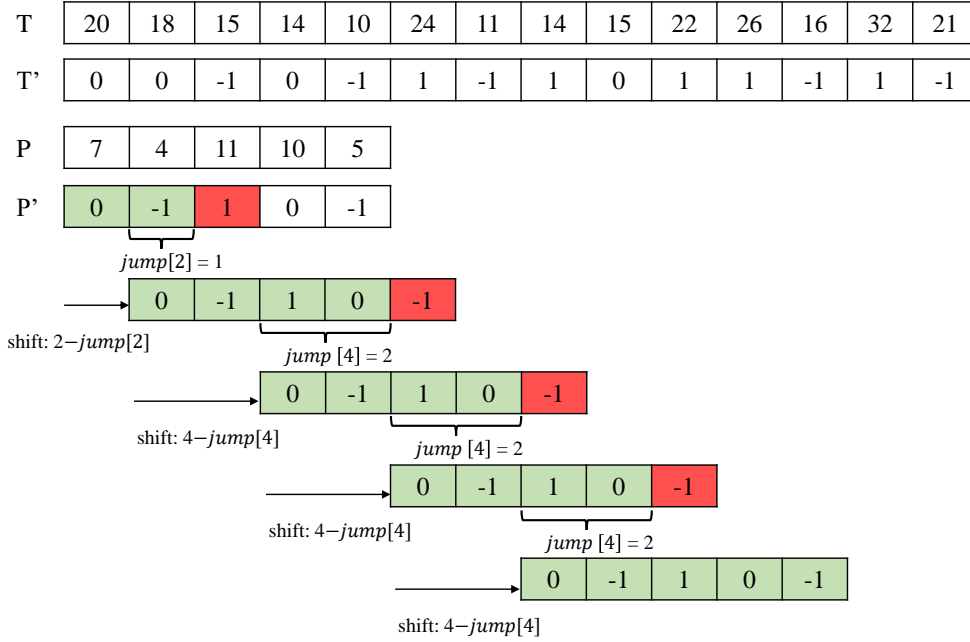


Fig. 3. An example for finding the candidates with fingerprints and the jump table in stage 1.

TABLE IX
NO AOPM BETWEEN $U = \langle 15, 14, 12, 10 \rangle$ AND $V = \langle 22, 25, 32, 28 \rangle$,
WHERE $c = 4$.

U	u_i	V	v_i	$fr(U, i)$	$fr(V, i)$	$fr(V, i) \cap fr(B, i)$
u_1	15	v_1	22	[2, 4]	[1, 2]	[2, 2]
u_2	14	v_2	25	[2, 4]	[1, 3]	[2, 3]
u_3	12	v_3	32	[1, 4]	[4, 4]	[4, 4]
u_4	10	v_4	28	[1, 2]	[2, 3]	[2, 2]

got similarly. Next, the separation rank is obtained if there is an overlap between two neighboring elements who cannot be reversed. Subsequently, we perform the intersection of the separation rank sets for two corresponding elements in P and $T_{i-m+1..i}$. If the intersection of each pair is not empty, then there is an aFRM between P and $T_{i-m+1..i}$.

Note that two strings with an aFRM may not guarantee to have an aOPM. Table IX illustrates an example with an aFRM, but there is no aOPM. Thus, the substring with an aFRM is one possible candidate to be passed to the next stage.

In stage 3, we use *bipartite matching* [3] to check if all positions in the candidate substring meet the condition of the almost matching. Figure 4 shows an example of the bipartite matching in our algorithm.

Let $G = (V_1, V_2, E)$ denote a bipartite graph, where each edge in E connects one vertex in V_1 and one vertex in V_2 . In addition, let $A = T_{i-m+1..i}$ and $B = P$. The transformation of our aFRM instance is given as follows.

$V_1(i)$ is denoted by $fr(A, i) \cap fr(B, i)$, for $1 \leq i \leq m$.
 $V_2(i) = i$, for $1 \leq i \leq m$.

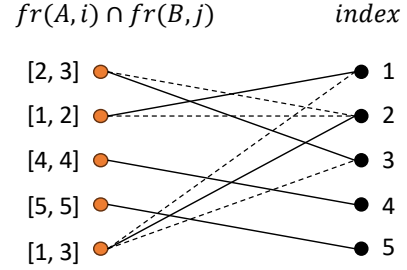


Fig. 4. The transformation from one aFRM instance to one bipartite matching instance in stage 3. Here, $[2, 3]$ means position 1 may be in index 2 or 3.

$e_{ij} \in E$ if $j \in fr(A, i) \cap fr(B, i)$.

If there exists one perfect matching in the bipartite matching instance, this perfect matching represents a permutation π that $\pi(P)$ and $\pi(T_{i-m+1..i})$ are both almost increasing. In other words, we have already found one substring for the answer of aOPM with the pattern P . Otherwise, this substring is not an aOPM answer. And, we will go back to stage 1 for identifying the next possible candidate.

The algorithm for solving the aOPM problem is formally presented in Algorithm 1. The time complexity of the proposed aOPM algorithm is $O(n + k_1 m \log m + R + k_2 m^3)$, where $n = |T|$, $m = |P|$, R is the number of separation rank combinations, k_1 and k_2 denote the number of candidate substrings identified in the fingerprint and almost rank stages, respectively. Note that the time complexity for bipartite matching is $O(m^3)$ for an m -vertex graph [3].

Algorithm 1 An algorithm for solving the aOPM problem

Require: One numeric text string $T = \langle t_1, t_2, t_3, \dots, t_n \rangle$, one numeric pattern string $P = \langle p_1, p_2, p_3, \dots, p_m \rangle$, and a tolerance constant c .

Ensure: The position index of T for the aOPM answer of P .

```
1: Build the fingerprint  $g_T(\cdot)$  of  $T$ 
2: Build the fingerprint  $g_P(\cdot)$  of  $P$ 
3:  $jump[\cdot] = \text{JUMP}(g_P)$ 
4:  $i \leftarrow 1, j \leftarrow 1$ 
5: while  $i \leq n$  do
6:                                      $\triangleright$  stage 1: fingerprint
7:   if  $g_T(i) \times g_P(j) \geq 0$  then    $\triangleright$  almost matching
8:      $i \leftarrow i + 1$ 
9:      $j \leftarrow j + 1$ 
10:  else                              $\triangleright$  not matching
11:    if  $j = 1$  then
12:       $i \leftarrow i + 1$ 
13:    else
14:       $j = jump[j - 1] + 1$ 
15:    if  $j = m + 1$  then              $\triangleright$  a fingerprint match
16:       $\triangleright$  stage 2: almost full-rank matching
17:       $A \leftarrow T_{i-m+1..i}$ , calculate  $fr(A, i)$ ,  $1 \leq i \leq m$ 
18:       $B \leftarrow P$ , calculate  $fr(B, i)$ ,  $1 \leq i \leq m$ 
19:      if there exists  $i$  that  $fr(A, i) \cap fr(B, i) = \emptyset$  then
20:        Break                        $\triangleright$  next candidate
21:       $\triangleright$  stage 3: bipartite matching
22:      Transform  $fr(A, i) \cap fr(B, i)$  to an instance of
      bipartite matching  $G$ 
23:      if there exists a perfect bipartite matching on  $G$ 
      then
24:        return  $i - m + 1$             $\triangleright$  start index of an aOPM
25:      return  $-1$                     $\triangleright$  not exist
```

IV. CONCLUSION

In this paper, we define a new problem, the almost order preserving matching (aOPM) problem, which is a more generalized variant of the order preserving matching (OPM) problem. It is also a generalized variant of the longest almost increasing subsequence (LaIS). Note that in the LaIS problem, a small drop is allowed within a subsequence exhibiting an increasing trend.

Our approach for solving the aOPM problem involves three key stages: fingerprint, almost rank, and bipartite matching. The total time complexity of our algorithm is $O(n + k_1 m \log m + R + k_2 m^3)$, where $n = |T|$, $m = |P|$, R is the number of separation rank combinations, k_1 and k_2 denote the number of candidate substrings identified in the fingerprint and almost rank stages, respectively.

REFERENCES

- [1] M. R. Alam and M. S. Rahman, "A divide and conquer approach and a work-optimal parallel algorithm for the LIS problem," *Information Processing Letters*, vol. 113, no. 13, pp. 470–476, 2013.
- [2] S. Bespamyatnikh and M. Segal, "Enumerating longest increasing subsequences and patience sorting," *Information Processing Letters*, vol. 76, no. 1-2, pp. 7–11, 2000.

- [3] K. P. Bogart, *Introductory Combinatorics*, 2nd ed. Harcourt Brace Jovanovich, 1990.
- [4] T. Chhabra and J. Tarhio, "A filtration method for order-preserving matching," *Information Processing Letters*, vol. 116, no. 2, pp. 71–74, 2016.
- [5] S. Cho, J. C. Na, K. Park, and J. S. Sim, "A fast algorithm for order-preserving pattern matching," *Information Processing Letters*, vol. 115, no. 2, pp. 397–402, 2015.
- [6] M. Crochemore and E. Porat, "Fast computation of a longest increasing subsequence and application," *Information and Computation*, vol. 208, no. 9, pp. 1054–1059, 2010.
- [7] A. Elmasry, "The longest almost-increasing subsequence," *Information Processing Letters*, vol. 110, no. 16, pp. 655–658, 2010.
- [8] J. W. Hunt and T. G. Szymanski, "A fast algorithm for computing longest common subsequences," *Communications of the ACM*, vol. 20, pp. 350–353, 1977.
- [9] J. Kim, P. Eades, R. Fleischer, S.-H. Hong, C. S. Iliopoulos, K. Park, S. J. Puglisi, and T. Tokuyama, "Order-preserving matching," *Theoretical Computer Science*, vol. 525, pp. 68–79, 2014.
- [10] C. Schensted, "Longest increasing and decreasing subsequences," *Canadian Journal of Mathematics*, vol. 13, pp. 179–191, 1961.
- [11] C. S. Thomas H. Cormen; Charles E. Leiserson, Ronald L. Rivest, *The Knuth-Morris-Pratt algorithm*, 2nd ed. The MIT Press, 2001.
- [12] P. van Emde Boas, R. Kaas, and E. Zijlstra, "Design and implementation of an efficient priority queue," *Mathematical Systems Theory*, vol. 10, no. 1, pp. 99–127, 1976.