Duel and Sweep Algorithm for Order-Preserving Pattern Matching

Davaajav Jargalsaikhan(✉), Diptarama, Yohei Ueki, Ryo Yoshinaka, and Ayumi Shinohara

Graduate School of Information Sciences, Tohoku University,
6-6-05 Aramaki Aza Aoba, Aoba-ku, Sendai, Japan
{davaajav,ry,ayumi}@ecei.tohoku.ac.jp,
{diptarama,yohei_ueki}@shino.ecei.tohoku.ac.jp

Abstract. Given a text and a pattern over an alphabet, the classic exact matching problem searches for all occurrences of the pattern in the text. Unlike exact matching, *order-preserving pattern matching* (OPPM) considers the relative order of elements, rather t han their real values. In this paper, we propose an efficient algorithm for the OPPM problem using the "duel-and-sweep" paradigm. For a pattern of length m and a text of length n, our algorithm runs in $O(n + m \log m)$ time in general, and in $O(n + m)$ time under an assumption that the characters in a string can be sorted in linear time with respect to the string size. We also perform experiments and show that our algorithm is faster than the KMP-based algorithm.

Keywords: Order-preserving pattern matching · Duel-and-sweep

1 Introduction

The exact string matching problem is one of the most widely studied problems. Given a text and a pattern, the exact matching problem searches for all occurrence positions of the pattern in the text. Many pattern matching algorithms have been proposed such as the well-known Knuth-Morris-Pratt algorithm [15], Boyer-Moore algorithm [2], and Horspool algorithm [13]. These algorithms preprocess the pattern first and then match the pattern from its prefix or suffix when comparing it with the text. Vishkin proposed two algorithms for pattern matching, pattern matching by duel-and-sweep [18] and pattern matching by sampling [19]. Both algorithms match the pattern to a substring of the text from some positions which are determined by the property of the pattern, instead of its prefix or suffix. These algorithms are developed also for parallel processing.

Furthermore, variants of Vishkin's duel-and-sweep algorithm have been developed for other types of pattern matching. Amir et al. [1] proposed a duel-and-sweep algorithm for the two-dimensional pattern matching problem. Cole et al. [7] generalized it for two-dimensional parameterized pattern matching. The aim of this paper is to show that the duel-and-sweep paradigm is also useful for

© Springer International Publishing AG 2018
A M. Tjoa et al. (Eds.): SOFSEM 2018, LNCS 10706, pp. 624–635, 2018.
https://doi.org/10.1007/978-3-319-73117-9_44

another variant of pattern matching, namely, *order-preserving pattern matching* (OPPM).

Unlike the exact matching problem, OPPM considers the relative order of elements, rather than their real values. Order-preserving matching has gained much interest in recent years, due to its applicability in problems where the relative order matters, such as share prices in stock markets, weather data or musical notes. The difficulty of OPPM mainly comes from the fact that we cannot determine the isomorphism by comparing the symbols in the text and the pattern on each position independently; instead, we have to consider their respective relative orders in the pattern and in the text.

Kubica et al. [16] and Kim et al. [14] independently proposed the same solution for OPPM based on the KMP algorithm. Their KMP-based algorithm runs in $O(n + m \log m)$ time. Cho et al. [6] brought forward another algorithm based on the Horspool algorithm that uses q-grams, which was proven to be experimentally fast. Crochemore et al. [8] proposed useful data structures for OPPM. On the other hand, Chhabra and Tarhio [5], Faro and Külekci [10] proposed filtration methods which are practically fast. Moreover, faster filtration algorithms using SIMD (Single Instruction Multiple Data) instructions were proposed by Cantone et al. [3], Chhabra et al. [4] and Ueki et al. [17]. They showed that SIMD instructions are effective in speeding up their algorithms.

In this paper, we propose a new algorithm for OPPM based on the duel-and-sweep technique. Our algorithm runs in $O(n + m \log m)$ time which is as fast as the KMP based algorithm. Moreover, we perform experiments to compare those algorithms, which show that our algorithm is faster than the KMP-based algorithm.

The rest of the paper is organized as follows. In Sect. 2, we give preliminaries on the problem. We describe our algorithm for the OPPM problem in Sect. 3. Section 4 shows some experimental results that compare the performance of our algorithm with the KMP-based algorithm. In Sect. 5, we conclude our work and discuss future directions.

2 Preliminaries

We use Σ to denote an alphabet of integer symbols such that the comparison of any two symbols can be done in constant time. Σ^* denotes the set of strings over the alphabet Σ. For a string $S \in \Sigma^*$, we will denote the i-th element of S by $S[i]$ and the substring of S that starts at the location i and ends at j as $S[i:j]$. We say that two strings S and T of equal length n are *order-isomorphic*, written $S \approx T$, if

$$S[i] \leq S[j] \iff T[i] \leq T[j] \text{ for all } 1 \leq i, j \leq n.$$

For instance, $(12, 35, 5) \approx (25, 30, 21) \not\approx (11, 13, 20)$.

In order to check the order-isomorphism of two strings, Kubica et al. [16] defined useful arrays[1] $Lmax_S$ and $Lmin_S$ by

$$Lmax_S[i] = j \ (j < i) \ \text{ if } \ S[j] = \max_{k<i}\{S[k] \mid S[k] \leq S[i]\}, \tag{1}$$

$$Lmin_S[i] = j \ (j < i) \ \text{ if } \ S[j] = \min_{k<i}\{S[k] \mid S[k] \geq S[i]\}. \tag{2}$$

We use the rightmost (largest) j if there exist more than one such j. If there is no such j then we define $Lmin_S[i] = 0$ and $Lmax_S[i] = 0$. From the definition, we can easily observe the following properties. Unless $Lmax_S[i] = 0$ or $Lmin_S[i] = 0$,

$$S[Lmax_S[i]] = S[i] \iff S[i] = S[Lmin_S[i]], \tag{3}$$

$$S[Lmax_S[i]] < S[i] \iff S[i] < S[Lmin_S[i]]. \tag{4}$$

Lemma 1 [16]**.** *For a string S, let $sort(S)$ be the time required to sort the elements of S. $Lmax_S$ and $Lmin_S$ can be computed in $O(sort(S) + |S|)$ time.*

Thus, $Lmax_S$ and $Lmin_S$ can be computed in $O(|S| \log |S|)$ time in general. Moreover, the computation can be done in $O(|S|)$ time under a natural assumption [16] that the characters of S are elements of the set $\{1, \ldots, |S|^{O(1)}\}$. By using $Lmax_S$ and $Lmin_S$, the order-isomorphism of two strings can be decided as follows.

Lemma 2 [6]**.** *For two strings S and T of length n, assume that $S[1\!:\!j] \approx T[1\!:\!j]$ for some $j < n$. Moreover assume that $Lmax_S[j+1] \neq 0$ and $Lmin_S[j+1] \neq 0$. Let $i_{max} = Lmax_S[j+1]$ and $i_{min} = Lmin_S[j+1]$. Then $S[1\!:\!j+1] \approx T[1\!:\!j+1]$ if and only if either of the following two conditions holds.*

$$S[i_{max}] = S[j+1] = S[i_{min}] \ \wedge \ T[i_{max}] = T[j+1] = T[i_{min}], \tag{5}$$

$$S[i_{max}] < S[j+1] < S[i_{min}] \ \wedge \ T[i_{max}] < T[j+1] < T[i_{min}]. \tag{6}$$

Corollary 1. *Suppose that $P[1:j-1] \approx Q[1:j-1]$ and $P[1:j] \not\approx Q[1:j]$ for two strings P and Q of length at least j. For $i_{max} = Lmax_P[j]$ and $i_{min} = Lmin_P[j]$, if $i_{max}, i_{min} \neq 0$, we have*

$$\begin{aligned} & P[j] = P[i_{max}] \wedge \ Q[j] \neq Q[i_{max}] \\ \vee \ & P[j] = P[i_{min}] \wedge \ Q[j] \neq Q[i_{min}] \\ \vee \ & P[j] > P[i_{max}] \wedge \ Q[j] \leq Q[i_{max}] \\ \vee \ & P[j] < P[i_{min}] \wedge \ Q[j] \geq Q[i_{min}]. \end{aligned}$$

The order preserving-pattern matching problem is defined as follows.

[1] Similar arrays $Prev_S$ and $Next_S$ are introduced in [12].

Definition 1 (OPPM problem).

Input: A text $T \in \Sigma^*$ of length n and a pattern $P \in \Sigma^*$ of length $m \leq n$.
Output: All occurrence positions of substrings of T that are order-isomorphic to P.

Hasan et al. [12] proposed a modification to Z-function, which Gusfield [11] defined for ordinary pattern matching, to make it useful from the order-preserving point of view. For a string S, the *(modified) Z-array* of S is defined by

$$Z_S[i] = \max_{1 \leq j \leq |S|-i+1} \{j \mid S[1:j] \approx S[i:i+j-1]\} \quad \text{for each } 1 \leq i \leq |S|.$$

In other words, $Z_S[i]$ is the length of the longest substring of S that starts at position i and is order-isomorphic to some prefix of S. An example of Z-array is illustrated in Table 1.

Table 1. Z-array of a string $S = (18, 22, 12, 50, 10, 17)$. For instance, $Z_S[3] = 3$ because $S[1:3] = (18, 22, 12) \approx (12, 50, 10) = S[3:5]$ and $S[1:4] = (18, 22, 12, 50) \not\approx (12, 50, 10, 17) = S[3:6]$. $Lmax_S$ and $Lmin_S$ are also shown.

	1	2	3	4	5	6
S	18	22	12	50	10	17
Z_S	6	1	3	1	2	1
$Lmax_S$	0	1	0	2	0	3
$Lmin_S$	0	0	1	0	3	1

Lemma 3 [12]**.** *For a string S, the Z-array Z_S can be computed in $O(|S|)$ time, assuming that $Lmax_S$ and $Lmin_S$ are already computed.*

Note that in their original work, Hasan et al. [12] assumed that each character in S is distinct. However, we can extend their algorithm by using Lemma 2 to verify order-isomorphism even when S contains duplicate characters.

In the remainder of this paper, we fix a text T of length n and a pattern P of length m.

3 Duel-and-sweep Algorithm for Order-Preserving Matching

In this section, we will propose an algorithm for OPPM based on the "duel-and-sweep" paradigm [1,18]. The duel-and-sweep paradigm screens all substrings of length m of the text, called *candidates*, in two stages, called the *dueling* and *sweeping* stages. Suppose when P is superimposed on itself with the offset

$a < m$, the two overlapped substrings of P are not order-isomorphic. Then it is impossible that two candidates with offset a are both order-isomorphic to P. The dueling stage lets each pair of candidates with such an offset a "duel" and eliminates one based on this observation. This test is quick but not perfect. This stage can remove many candidates, although there would still remain candidates which are actually not order-isomorphic to the pattern. On the other hand, it is guaranteed that if distinct candidates that survive the dueling stage overlap, their prefixes of certain length are order-isomorphic. The sweeping stage takes the advantage of this property when checking the order-isomorphism between surviving candidates and the pattern so that this stage can be done also quickly.

Prior to the dueling stage, the pattern is preprocessed to construct a *witness table* based on which the dueling stage decides which pair of overlapping candidates should duel and how they should duel.

3.1 Pattern Preprocessing

For each offset $0 < a < m$, the original duel-and-sweep algorithm [18] saves a position i such that $P[i] \neq P[i + a]$. However, in order-preserving pattern matching, the order-isomorphism of two strings cannot be determined by comparing a symbol in one position. We need two positions as a *witness* to say that the two strings are not order-isomorphic. Therefore, for each offset $0 < a < m$, when the overlapped regions obtained by superimposing P on itself with offset a are not order-isomorphic, we use a pair $\langle i, j \rangle$ of locations called *a witness pair for the offset* a if either of the following holds:

- $P[i] = P[j]$ and $P[i + a] \neq P[j + a]$,
- $P[i] > P[j]$ and $P[i + a] \leq P[j + a]$,
- $P[i] < P[j]$ and $P[i + a] \geq P[j + a]$.

Next, we describe how to construct a *witness table* for P, that stores witness pairs for all possible offsets a $(0 < a < m)$. The witness table WIT_P is an array of length $m - 1$, such that $WIT_P[a]$ is a witness pair for offset a. In the case when there are multiple witness pairs for offset a, we take the pair $\langle i, j \rangle$ with the smallest value of j and some $i < j$. When the overlap regions are order-isomorphic for offset a, which implies that no witness pair exists for a, we express it as $WIT_P[a] = \langle 0, 0 \rangle$. Table 2 shows an example of a witness table.

Table 2. Witness table WIT_P for a string $P = (18, 22, 12, 50, 10, 17)$. For instance, the witness pair $WIT_P[2]$ for offset 2 is $\langle 2, 4 \rangle$, due to $P[2] = 22 < 50 = P[4]$ and $P[2 + 2] = 50 > 17 = P[4 + 2]$. On the other hand, $WIT_P[4] = \langle 0, 0 \rangle$, since $P[1 : 2] \approx P[5 : 6]$.

	1	2	3	4	5	6
P	18	22	12	50	10	17
WIT_P	$\langle 1, 2 \rangle$	$\langle 2, 4 \rangle$	$\langle 1, 2 \rangle$	$\langle 0, 0 \rangle$	$\langle 0, 0 \rangle$	–

Algorithm 1. Algorithm for constructing the witness table WIT_P

```
1  Function Witness(P) /* Construct the witness table WIT_P */
2      compute the Z-array Z_P for the pattern P;
3      for a = 1 to m − 1 do
4          j = Z_P[a + 1] + 1, i_min = Lmin_P[j] and i_max = Lmax_P[j];
5          if j = m − a + 1 then WIT_P[a] = ⟨0, 0⟩;
6          else if i_max = 0 then WIT_P[a] = ⟨i_min, j⟩;
7          else if i_min = 0 then WIT_P[a] = ⟨i_max, j⟩;
8          else if P[i_min] = P[j] ∧ P[i_min + a] ≠ P[j + a]
9                  ∨ P[i_min] > P[j] ∧ P[i_min + a] ≤ P[j + a] then
10             WIT_P[a] = ⟨i_min, j⟩
11         else
12             WIT_P[a] = ⟨i_max, j⟩
```

Lemma 4. *For a pattern P of length m, Algorithm 1 constructs WIT_P in $O(m)$ time assuming that Z_P is already computed.*

Proof. Clearly the algorithm runs in $O(m)$ time.

We show that for each $1 \leq a < m$, Algorithm 1 computes $WIT_P[a]$ correctly. Recall that $Z_P[a+1]$ is the length of the longest prefix of $P[a+1:m]$ that is order-isomorphic to a prefix of P. Let $j = Z_P[a+1]+1$, for which we have $P[1:j-1] \approx P[1+a:j-1+a]$. Suppose that $j = m-a+1$. This means that $P[1:j-1] \approx P[1+a:j-1+a] = P[1+a:m]$, i.e., there is no witness pair for the offset a. Indeed Algorithm 1 gets $WIT_P[a] = \langle 0, 0\rangle$ for this case.

Otherwise, we have $P[1:j] \not\approx P[1+a:j+a]$. Let $i_{max} = Lmax_P[j]$ and $i_{min} = Lmin_P[j]$. If $i_{max} = 0$, $P[j] < P[k]$ for all $k < j$. Note that $i_{min} \neq 0$ by $j \geq 2$. Since $P[1:j-1] \approx P[1+a:j-1+a]$ and $P[1:j] \not\approx P[1+a:j+a]$, there exists $1 \leq k < j$ such that $P[j+a] \geq P[k+a]$. By $P[i_{min}] \leq P[k]$ and $(P[i_{min}], P[k]) \approx (P[i_{min}+a], P[k+a])$, we have $P[i_{min}+a] \leq P[k+a] \leq P[j+a]$. Therefore, $\langle i_{min}, j\rangle$ is a witness pair for the offset a. The case where $i_{min} = 0$ can be discussed in the exactly symmetric way.

Let us assume $i_{min} \neq 0$ and $i_{max} \neq 0$. If $P[i_{min}] = P[j] \wedge P[i_{min}+a] \neq P[j+a]$ or $P[i_{min}] > P[j] \wedge P[i_{min}+a] \leq P[j+a]$, clearly $\langle i_{min}, j\rangle$ is a witness pair for a. Otherwise, by Corollary 1, either $P[i_{max}] = P[j] \wedge P[i_{max}+a] \neq P[j+a]$ or $P[i_{max}] < P[j] \wedge P[i_{max}+a] \geq P[j+a]$ holds, in which case $\langle i_{max}, j\rangle$ is a witness pair for a. □

3.2 Dueling Stage

Let us denote the candidate that starts at the location x as $T_x = T[x:x+m-1]$. In the dueling stage, we "duel" all pairs of overlapping candidates T_x and T_{x+a} such that $WIT_P[a] \neq \langle 0, 0\rangle$. Witness pairs are used in the following manner. Suppose that $WIT_P[a] = \langle i, j\rangle$, where $P[i] < P[j]$ and $P[i+a] \geq P[j+a]$, for example. Then, it holds that

Algorithm 2. Dueling

```
1  Function Dueling(x, a) /* Duel between candidates T_x and T_{x+a}   */
2      ⟨i, j⟩ = WIT_P[a];
3      if P[i] = P[j] then
4          if T[x + a + i − 1] ≠ T[x + a + j − 1] then return x;
5          else return x + a;
6      if P[i] < P[j] then
7          if T[x + a + i − 1] ≥ T[x + a + j − 1] then return x;
8          else return x + a;
9      if P[i] > P[j] then
10         if T[x + a + i − 1] ≤ T[x + a + j − 1] then return x;
11         else return x + a;
```

- if $T[x+a+i-1] \geq T[x+a+j-1]$, then $T_{x+a} \not\approx P$,
- if $T[x+a+i-1] < T[x+a+j-1]$, then $T_x \not\approx P$.

Based on this observation, we can safely eliminate either candidate T_x or T_{x+a} without looking into other locations. We can perform this process similarly for other equality/inequality cases. This process is called *dueling*. The procedure for all cases of the dueling is described in Algorithm 2.

On the other hand, if T_x and T_{x+a} do not overlap or the offset a has no witness pair, i.e. $P[1:m-a] \approx P[a+1:m]$, no dueling is performed on them. We say that a position x *is consistent with* $x+a$ if either $0 < a < m$ and $WIT_P[a] = \langle 0, 0\rangle$ or $a \geq m$. Note that the consistency property is determined by a and P only, and x and T are irrelevant. The consistency property is transitive.

Lemma 5. *For any a, b and x such that $1 \leq a < a+b < m$ and $1 \leq x < m-a-b$, if x is consistent with $x+a$ and $x+a$ is consistent with $x+a+b$, then x is consistent with $x+a+b$.*

Proof. Since x is consistent with $x+a$, it follows that $P[1:m-a] \approx P[a+1:m]$, so that $P[b+1:m-a] \approx P[(a+b)+1:m]$. Moreover, since $x+a$ is consistent with $x+a+b$, it follows that $P[1:m-b] \approx P[b+1:m]$, so that $P[1:m-b-a] \approx P[b+1:m-a]$. Thus, $P[1:m-(a+b)] \approx P[(a+b)+1:m]$, which implies that x is consistent with $x+a+b$. □

The whole process of the dueling stage is shown in Algorithm 3, which follows Amir et al. [1] for ordinary pattern matching. This stage eliminates candidates until all surviving candidates are pairwise consistent. The algorithm uses a stack to maintain candidates which are consistent with each other. A new candidate y will be pushed to the stack if the stack is empty. Otherwise y is checked by comparing it to the topmost element x of the stack. By Lemma 5, if x is consistent with y, all the other elements in the stack are consistent with y, too. Thus we can push y to the stack. On the other hand, if x is not consistent with y, we should exclude one of the candidates by dueling them. If x wins the duel,

Algorithm 3. The dueling stage algorithm

```
1  Function DuelingStage(P, T)
2      create stack;
3      for y = 1 to n − m + 1 do
4          while stack is not empty do
5              pop x from stack;
6              if y − x ≥ m or WIT_P[y − x] = ⟨0, 0⟩ then
7                  push x and y to stack;
8                  break;
9              else
10                 z = Dueling(x, y − x);
11                 if z = x then
12                     push x to stack;
13                     break;
14         if stack is empty then
15             push y to stack;
```

we put x back to the stack, discard y, and get a new candidate. If y wins the duel, we exclude x and continue comparison of y with the top element of the stack unless the stack is empty. If the stack is empty, y will be pushed to the stack. Figure 1 gives an example run of the dueling stage.

Lemma 6 [1]. *The dueling stage can be done in $O(n)$ time by using WIT_P.*

3.3 Sweeping Stage

The goal of the sweeping stage is to prune inconsistent candidates until all remaining candidates are order-isomorphic to the pattern P. Suppose that we need to check whether some surviving candidate T_x is order-isomorphic to P. It suffices to successively check the conditions (5) and (6) in Lemma 2, starting from the leftmost location in T_x. If the conditions are satisfied for all locations in T_x, then $T_x \approx P$. Otherwise, $T_x \not\approx P$, and obtain a mismatch position j.

A Naive implementation of sweeping requires $O(n^2)$ time. Algorithm 4 takes advantage of the fact that all the remaining candidates are pairwise consistent, we can reduce the time complexity to $O(n)$ time. Suppose there is a mismatch at position j when comparing P with T_x, that is, $T_x[1:j-1] \approx P[1:j-1]$ and $T_x[1:j] \not\approx P[1:j]$. If the next candidate is T_{x+a} with $a < j$, since $P[1:j-a-1] \approx P[a+1:j-1] \approx T_x[a+1:j-1] = T_{x+a}[1:j-a-1]$, we can start comparison of P and T_{x+a} from the position where the mismatch with T_x occurred. If $P \approx T_x$, the above discussion holds for $j = m+1$. Therefore, the total number of comparison is bounded by $O(n)$, by applying the same argument on the complexity of the KMP algorithm for exact matching.

Lemma 7. *The sweeping stage can be completed in $O(n)$ time.*

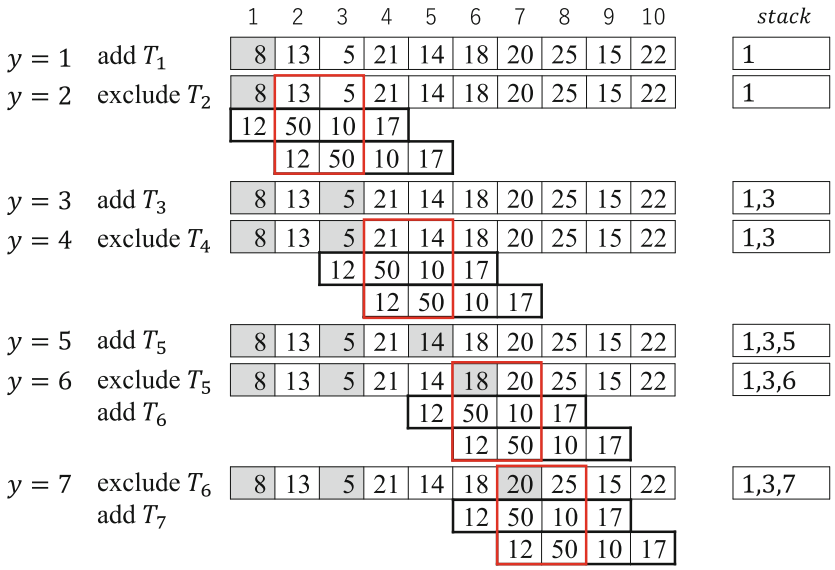

Fig. 1. An example run of the dueling stage for $T = (8, 13, 5, 21, 14, 18, 20, 25, 15, 22)$, $P = (12, 50, 10, 17)$, and $WIT_P = (\langle 1, 2\rangle, \langle 0, 0\rangle, \langle 0, 0\rangle)$. First, the position 1 is pushed to the stack. Next, T_2 duels with T_1 and then T_2 loses because $P[1] < P[2]$ and $T_2[1] > T_2[2]$. The next position 3 is pushed to the stack by $WIT_P[3-1] = \langle 0, 0\rangle$. Similarly, T_4 loses against T_3, and 5 is accepted to the stack. For $y = 6$, T_5 is removed and T_6 is added because $P[1] < P[2]$, $T_6[1] < T_6[2]$, and 3 is consistent with 6. Finally T_7 defeats T_6 and the contents of the stack become 1, 3, and 7.

By Lemmas 4, 6, and 7, we summarize this section as follows.

Theorem 1. *Given a text T of length n and a pattern P of length m, the dueling-and-sweep algorithm solves the OPPM Problem in $O(n + m \log m)$ time. Moreover, the running time is $O(n + m)$ under the natural assumption that the characters of P can be sorted in $O(m)$ time.*

4 Experiments

In order to compare the performance of proposed algorithm with the KMP-based algorithm [14,16] on solving the OPPM problem, we performed two sets of experiments. In the first experiment set, the pattern size m is fixed to 10, while the text size n is changed from 100000 to 1000000. In the second experiment set, the text size n is fixed to 1000000 while the pattern size m is changed from 5 to 100. We measured the average of running time and the number of comparisons for 50 repetitions on each experiment. We used randomly generated texts and patterns with alphabet size $|\Sigma| = 1000$. Experiments are executed on a machine with Intel Xeon CPU E5-2609 8 cores 2.40 GHz, 256 GB memory, and Debian Wheezy operating system.

Algorithm 4. The sweeping stage algorithm

```
1  Function SweepingStage()
2      while there are unchecked candidates to the right of T_x do
3          let T_x be the leftmost unchecked candidate;
4          if there are no candidates overlapping with T_x then
5              if T_x ≉ P then eliminate T_x;
6          else
7              let T_{x+a} be the leftmost candidate that overlaps with T_x;
8              if T_x ≈ P then start checking T_{x+a} from the location m − a + 1;
9              else
10                 let j be the mismatch position;
11                 eliminate T_x;
12                 start checking T_{x+a} from the location j − a;
```

The results of our experiments are shown in Figs. 2 and 3. We can see that our algorithm is better than the KMP-based algorithm in running time and the number of comparisons when the pattern size and text size are large. However, our algorithm was slower when the pattern is very short, namely $m = 5$. The reason why the proposed algorithm makes fewer comparisons than the KMP-based algorithm may be explained as follows. The KMP-based algorithm relies on Lemma 2, which compares symbols at three positions[2] to check the order-isomorphism between a prefix of the pattern and a substring of the text when the prefix is extended by one. On the other hand, the dueling stage of our algorithm compares only two positions determined by the witness table. By pruning candidates in the dueling phase, the number of precise tests of order-isomorphism in the sweeping stage is reduced.

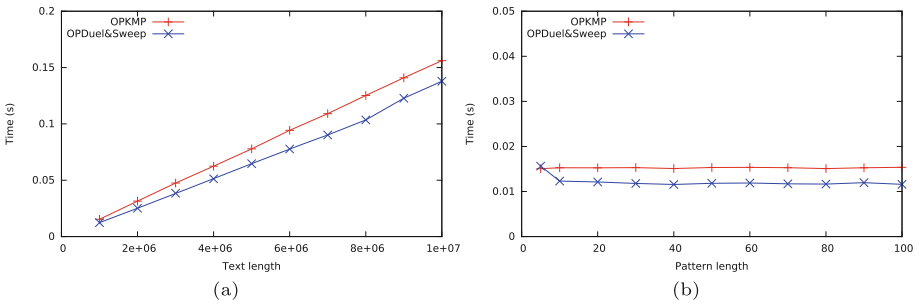

Fig. 2. Running time of the algorithms with respect to (a) text length, and (b) pattern length.

[2] Each of (5) and (6) of Lemma 2 involves four (in)equalities but checking three is enough thanks to the properties (3) and (4).

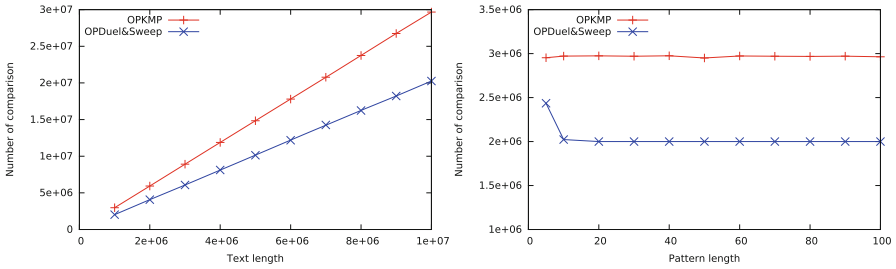

Fig. 3. Number of comparisons in the algorithms with respect to (a) text length, and (b) pattern length.

5 Discussion

We proposed a new algorithm for the OPPM problem by extending Vishkin's duel-and-sweep algorithm [18] for the exact matching problem. Our algorithm runs in linear time, that is theoretically fast. The experimental results showed that our algorithm is practically faster than the KMP-based algorithm [14,16], which has the same theoretical running time. Actually, our algorithm makes fewer comparisons than the KMP-based algorithm.

Since Vishkin's algorithm has been designed for parallel computing [18], we expect that our duel-and-sweep algorithm for order preserving pattern matching could also be extended for parallel computing. This extension is not trivial because the periodicity property of a string in order preserving pattern matching is different from the one in ordinary pattern matching.

Another potential of the duel-and-sweep paradigm is in solving *two-dimensional* pattern matching problems. Amir et al. [1] and Cole et al. [7] have designed duel-and-sweep algorithms for solving two-dimensional exact and parameterized pattern matching problems, respectively. Currently no fast algorithm for the two-dimensional order-preserving pattern matching problem has been proposed. Actually we have already developed a dueling algorithm for two-dimensional OPPM that runs in linear time with respect to the input text size [9]. However, we do not have a linear time algorithm for the sweeping stage yet. We hope the two-dimensional OPPM problem can be solved more efficiently by finding a more sophisticated method based on some combinatorial properties, like Cole et al. did for the two-dimensional parameterized matching problem. This is left for future work.

Acknowledgements. This work is supported by Tohoku University Division for Interdisciplinary Advance Research and Education, ImPACT Program of Council for Science, Technology and Innovation (Cabinet Office, Government of Japan), and JSPS KAKENHI Grant Number JP15H05706.

References

1. Amir, A., Benson, G., Farach, M.: An alphabet independent approach to two-dimensional pattern matching. SIAM J. Comput. **23**(2), 313–323 (1994)
2. Boyer, R.S., Moore, J.S.: A fast string searching algorithm. Commun. ACM **20**(10), 762–772 (1977)
3. Cantone, D., Faro, S., Külekci, M.O.: An efficient skip-search approach to the order-preserving pattern matching problem. In: PSC, pp. 22–35 (2015)
4. Chhabra, T., Külekci, M.O., Tarhio, J.: Alternative algorithms for order-preserving matching. In: PSC, pp. 36–46 (2015)
5. Chhabra, T., Tarhio, J.: A filtration method for order-preserving matching. Inf. Process. Lett. **116**(2), 71–74 (2016)
6. Cho, S., Na, J.C., Park, K., Sim, J.S.: A fast algorithm for order-preserving pattern matching. Inf. Process. Lett. **115**(2), 397–402 (2015)
7. Cole, R., Hazay, C., Lewenstein, M., Tsur, D.: Two-dimensional parameterized matching. ACM Trans. Algorithms **11**(2), 12:1–12:30 (2014)
8. Crochemore, M., Iliopoulos, C.S., Kociumaka, T., Kubica, M., Langiu, A., Pissis, S.P., Radoszewski, J., Rytter, W., Waleń, T.: Order-preserving indexing. Theor. Comput. Sci. Pattern Matching **638**, 122–135 (2016). Text Data Structures and Compression
9. Davaajav, J.: A study on the two-dimensional order-preserving matching problem. Bachelor thesis, Tohoku University (2017)
10. Faro, S., Külekci, M.O.: Efficient algorithms for the order preserving pattern matching problem. In: Dondi, R., Fertin, G., Mauri, G. (eds.) AAIM 2016. LNCS, vol. 9778, pp. 185–196. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41168-2_16
11. Gusfield, D.: Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press, Cambridge (1997)
12. Hasan, M.M., Islam, A.S., Rahman, M.S., Rahman, M.S.: Order preserving pattern matching revisited. Pattern Recogn. Lett. **55**, 15–21 (2015)
13. Horspool, R.N.: Practical fast searching in strings. Softw. Pract. Experience **10**(6), 501–506 (1980)
14. Kim, J., Eades, P., Fleischer, R., Hong, S.H., Iliopoulos, C.S., Park, K., Puglisi, S.J., Tokuyama, T.: Order-preserving matching. Theor. Comput. Sci. **525**, 68–79 (2014)
15. Knuth, D.E., Morris Jr., J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM J. Comput. **6**(2), 323–350 (1977)
16. Kubica, M., Kulczyński, T., Radoszewski, J., Rytter, W., Waleń, T.: A linear time algorithm for consecutive permutation pattern matching. Inf. Process. Lett. **113**(12), 430–433 (2013)
17. Ueki, Y., Narisawa, K., Shinohara, A.: A fast order-preserving matching with q-neighborhood filtration using SIMD instructions. In: SOFSEM (Student Research Forum Papers/Posters), pp. 108–115 (2016)
18. Vishkin, U.: Optimal parallel pattern matching in strings. In: Brauer, W. (ed.) ICALP 1985. LNCS, vol. 194, pp. 497–508. Springer, Heidelberg (1985). https://doi.org/10.1007/BFb0015775
19. Vishkin, U.: Deterministic sampling - a new technique for fast pattern matching. SIAM J. Comput. **20**(1), 22–40 (1991)