A Fast Order-Preserving Matching with q-neighborhood Filtration Using SIMD Instructions

Yohei Ueki, Kazuyuki Narisawa, and Ayumi Shinohara

Graduate School of Information Sciences, Tohoku University, Japan
{yohei_ueki@shino.,narisawa@,ayumi@}ecei.tohoku.ac.jp

Abstract. The *order-preserving matching problem* is a variant of the pattern matching problem focusing on shapes of sequences instead of values of sequences. Given a text and a pattern, the problem is to output all positions where the pattern and a subsequence in the text are of the same relative order. Chhabra and Tarhio proposed a fast algorithm based on filtration for the order-preserving matching problem, and Faro and Külekci improved Chhabra and Tarhio's solution by extending the filter. Furthermore, Cantone *et al.* and Chhabra *et al.* proposed solutions based on filtration using SIMD (Single Instruction Multiple Data) instructions, and showed that SIMD instructions are efficient in speeding up their algorithms. In this paper, we propose a fast matching algorithm for the order-preserving matching problem using SIMD instructions based on filtration proposed by Faro and Külekci. We show that our algorithm is practically faster than previous solutions.

Keywords: pattern matching problem, order-preserving matching problem, SIMD instructions

1 Introduction

The *order-preserving matching problem* [9, 11] is a variant of the pattern matching problem focusing on shapes of sequences instead of values of sequences. This matching can be applied to various fields, such as musical matching, sensor data analysis and stock price analysis. Kubica *et al.* [11] defined an *order-isomorphism* as one of the similarities of sequences. For two numerical sequences X and Y of the same length, the order-isomorphism expresses that the relative order of X coincides with that of Y. For example, two sequences $X = (8, 32, 40, 24, 16)$ and $Y = (18, 42, 50, 34, 26)$ are order-isomorphic because both the relative orders of X and Y are $(1, 4, 5, 3, 2)$. The order-preserving matching problem is, given a text and a pattern, to output all positions of subsequences in the text that are order-isomorphic to the pattern.

Various sequential matching algorithms for the order-preserving matching problem have been developed, based on the Knuth-Morris-Pratt Algorithm [9, 11], Horspoo Algorithm [5], and forward automaton Algorithm [1]. Moreover, Chhabra and Tarhio [4] proposed a practically fast pattern matching algorithm using a filtration method. In this method, text T and pattern P are encoded into binary sequences T' and P' respectively, based on the relationship to the adjacent values. Because the standard string matching P' in T' is much faster than the order-preserving matching P in T, it can be used to narrow

the candidate positions, although some incorrect answers may be included. Hence this methods requires verification steps for the candidates. Faro and Külekci [7] improved the filter by considering q-neighborhood values, instead of adjacent (1-neighborhood) values. More recently, solutions based on filtration using SIMD (Single Instruction Multiple Data) instructions were proposed by Cantone *et al.* [2] and Chhabra *et al.* [3]. They showed that SIMD instructions are efficient in speeding up their algorithms.

In this paper, we propose a new fast algorithm using SIMD instructions based on filtration proposed by Faro and Külekci [7]. Our experiments show that our algorithm is practically faster than previous solutions.

2 Preliminaries

2.1 Notations

Let Σ be an *ordered alphabet*, and Σ^* be the set of all sequences over Σ. $|X|$ denotes the length of a sequence $X \in \Sigma^*$, and $X[i]$ denotes the i-th value of X for $1 \leq i \leq |X|$. A *subsequence* of X beginning at i and ending at j for $1 \leq i \leq j \leq |X|$ is denoted by $X[i : j] = (X[i], X[i+1], \ldots, X[j-1], X[j])$.

2.2 Order Preserving Matching

Definition 1 (Order-isomorphism [11]). *Two sequences $X, Y \in \Sigma^*$ of the same length are* order-isomorphic *if $X[i] \leq X[j] \iff Y[i] \leq Y[j]$ for any $1 \leq i, j \leq |X|$. We write $X \approx Y$ if X is order-isomorphic to Y, and $X \not\approx Y$ otherwise.*

Example 1. For sequences $X = (8, 32, 40, 24, 16)$, $Y = (18, 42, 50, 34, 26)$ and $Z = (20, 24, 45, 38, 31)$, we have $X \approx Y$ and $X \not\approx Z$.

Definition 2 (Order-Preserving Matching Problem [9, 11]). *Given a text $T \in \Sigma^*$ of length n, and a pattern $P \in \Sigma^*$ of length m, the* order-preserving matching problem *asks for all positions i satisfying $T[i : i+m-1] \approx P$ for $1 \leq i \leq n-m+1$.*

Example 2. For a text $T = (13, 18, 42, 50, 34, 26, 12, 20, 24, 45, 38, 31)$ and a pattern $P = (8, 32, 40, 24, 16)$, the output is 2 because $T[2 : 6] \approx P$, see Fig. 1.

Solutions for the order-preserving matching problem based on filtration require a verification step. That is, each candidate $T[i : i+m-1]$ is verified whether it is order-isomorphic to the pattern P of length m. It takes $O(m^2)$ time by using a naive algorithm based on Definition 1. The previous work [2, 4] showed the following lemma for verifying the order-isomorphism of two sequences of length m in $O(m)$ time with $O(\mathrm{sort}(m))$ preprocessing time, where $\mathrm{sort}(m)$ is the time required to sort one of the sequences.

Definition 3 (relative order array [2, 4]). *For a sequence $Y \in \Sigma^*$, the* relative order array *R_Y of Y is defined as $R_Y = (rank_Y^{-1}(1), rank_Y^{-1}(2), \ldots, rank_Y^{-1}(|Y|))$, where $rank_Y(i) = |\{k : Y[k] < Y[i]$ or $(Y[k] = Y[i]$ and $k < i)\}|$.*

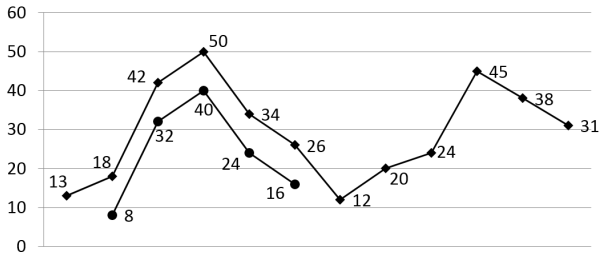

Fig. 1. An example of order-preserving matching. The pattern $P = (8, 32, 40, 24, 16)$ matches at position 2 in the text $T = (13, 18, 42, 50, 34, 26, 12, 20, 24, 45, 38, 31)$, because both the relative orders of P and $T[2:6] = (18, 42, 50, 34, 26)$ are $(1, 4, 5, 3, 2)$.

Lemma 1 ([4]). *Given two sequences $X, Y \in \Sigma^*$ of length m, and the relative order array R_Y of Y, we have $X \not\approx Y$ if and only if there exists $1 \leq j \leq m-1$ such that one of the following conditions holds:*

(1) $X[R_Y[j]] > X[R_Y[j+1]]$,
(2) $X[R_Y[j]] = X[R_Y[j+1]]$ *and* $X[R_Y[j]] \neq X[R_Y[j+1]]$, *or*
(3) $X[R_Y[j]] < X[R_Y[j+1]]$ *and* $X[R_Y[j]] = X[R_Y[j+1]]$.

The relative order array can be computed in $O(\mathrm{sort}(m))$ time [4]. Therefore, the verification algorithm based on Lemma 1 runs in $O(m)$ time with $O(\mathrm{sort}(m))$ preprocessing time.

3 Previous Work

3.1 Neighborhood Ranking Filter

Chhabra and Tarhio [4] proposed an order-preserving filtration technique. In this paper, we call it *Neighborhood Ranking filter* (shortly NR filter). It consists of two phases, the filtration phase and the verification phase.

In the filtration phase, candidates are filtered out by using *Neighborhood Ranking code* (shortly NR code) defined by the following.

Definition 4 (Neighborhood Ranking code). *For a sequence $X \in \Sigma^*$, the* Neighborhood Ranking code *of X is defined as $B(X)[i] = 1$ if $X[i] < X[i+1]$, and 0 otherwise, for $1 \leq i \leq |X| - 1$.*

At the beginning of the filtering phase, we compute the NR code $B(P)$ of the pattern P. Next, we find all positions i satisfying $B(T[i : i+m-1]) = B(P)$ for $1 \leq i \leq n-m+1$, that are candidates of the order-preserving matching. In order to find these positions, we can use any standard string matching algorithms, such as Knuth-Morris-Pratt algorithm [10]. Such candidates include no false negative results, in other words, the filter never removes correct answers by the following proposition.

Proposition 1 ([4]). *For any two sequences* $X, Y \in \Sigma^*$, $X \approx Y \Rightarrow B(X) = B(Y)$.

The converse of Proposition 1 is not always true. Hence, candidates include false positive results, in other words, the filter may pass incorrect answers.

In the verification phase, every candidate is checked whether it is order-isomorphic to the pattern or not. The verification method is based on Lemma 1, which requires pre-computing the relative order array R_P of pattern P.

Example 3. We consider again the instance in Example 1 and Fig. 1. At first, the relative order array $R_P = (1, 5, 4, 2, 3)$ of P is computed. Next, in the filtering phase, T and P are encoded into NR codes as $B(T) = (1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0)$ and $B(P) = (1, 1, 0, 0)$, respectively. The positions i that satisfy $B(T)[i : i + m - 1] = B(P)$ are $i = 2$ and $i = 8$ only. Therefore, the candidates are $T[2 : 6] = (18, 42, 50, 34, 26)$ and $T[8 : 12] = (20, 24, 45, 38, 31)$. Lastly, in the verification phase, $T[2 : 6]$ and $T[8 : 12]$ are verified whether they are order-isomorphic to P or not, using the verification algorithm based on Lemma 1. As a result, the position 2 is reported.

3.2 q-Neighborhood Ranking Filter

Faro and Külekci [7] proposed *q-Neighborhood Ranking code* (q-NR code) for the filtration technique, which is a more effective filtration technique than that of using the original neighborhood ranking code. It uses q-neighborhood relationships, while the original neighborhood ranking code uses only one-adjacent relationships.

Definition 5 (q-Neighborhood Ranking code [7]). *Let* $X \in \Sigma^*$ *be a sequence of length* n*, and* q *be an integer satisfying* $1 \leq q < n$*. The* q-Neighborhood Ranking code *of* X *is defined as* $B_q(X)[i] = \sum_{j=1}^{q} (\beta_X(i, j) \cdot 2^{q-j})$ *where* $\beta_X(i, j) = 1$ *if* $X[i] < X[i + j]$*, and* 0 *otherwise, for* $1 \leq i \leq n - q$.

Example 4. For a sequence $X = (8, 32, 40, 24, 16)$, we have $B_1(X) = (1, 1, 0, 0)$, $B_2(X) = ((11)_2, (10)_2, (00)_2) = (3, 2, 0)$ and $B_3(X) = ((111)_2, (100)_2) = (7, 4)$.

The next lemma guarantees that candidates filtered by q-NR code also contain no false negatives as well as the NR filter.

Lemma 2 ([7]). *For any two sequences* $X, Y \in \Sigma^*$, $X \approx Y \Rightarrow B_q(X) = B_q(Y)$.

The converse of Lemma 2 is not always true, similarly to Proposition 1. Hence, candidates obtained by using the q-NR filter also possibly contain false positive results.

4 Proposed Methods

4.1 Fast Implementation Using SIMD Instructions

In this section, we propose a fast implementation using SIMD (Single Instruction Multiple Data) instructions. We use SSE4.2 [8] for a SIMD instruction set. SSE4.2 supports 128-bit registers, that can contain two 64-bit, four 32-bit, eight 16-bit, or sixteen 8-bit numbers. The *packing factor* $\alpha = 128/w$ stands for the number of w-bit numbers contained in the 128-bit register. SSE4.2 allows to perform the same operation in parallel on several numbers stored in the 128-bit register.

Table 1. SIMD functions, their SSE intrinsics ($w = 8$) and their behaviors

function	SSE intrinsics	behavior
$CompLt(\tilde{X}_1, \tilde{X}_2)$	`_mm_cmplt_epi8`	Return a sequence $\tilde{C}$, where $\tilde{C}[i] = 2^w - 1$ if $\tilde{X}_1 < \tilde{X}_2$, and 0 otherwise for $1 \le i \le \alpha$.
$And(\tilde{X}_1, \tilde{X}_2)$	`_mm_and_si128`	Return a sequence $\tilde{A}$, where $\tilde{A}[i] = \tilde{X}_1[i] \,\&\, \tilde{X}_2[i]$ for $1 \le i \le \alpha$.
$Or(\tilde{X}_1, \tilde{X}_2)$	`_mm_or_si128`	Return a sequence $\tilde{O}$, where $\tilde{O}[i] = \tilde{X}_1[i] \,\|\, \tilde{X}_2[i]$ for $1 \le i \le \alpha$.
$MoveMask(\tilde{X})$	`_mm_movemask_epi8`	Return a bit mask from most significant bits of $\tilde{X}[i]$ for $1 \le i \le \alpha$.
$SearchStr(\tilde{X}_P, m, \tilde{X}_T, n)$	`_mm_cmpestrm`	Find $\tilde{X}_P$ of length m from $\tilde{X}_T$ of length n.

Algorithm 1: OPFSI$_q$ Algorithm

Input: text T of length n and pattern P of length m.
Output: all positions i satisfying $T[i : i + m - 1] \approx P$.

Compute the relative order array R_P ;
$\tilde{P} \leftarrow B_q(P)[1 : \min(\alpha, m - q)]$;
if $|\tilde{P}| < \alpha$ **then** $\tilde{P}$ is padded with trailing zeros so that $|\tilde{P}| = \alpha$;
$i \leftarrow 1$;
while $i \le n - \alpha - q + 1$ **do**
  $\tilde{T} \leftarrow Encode_q(T, i)$; $\tilde{S} \leftarrow SearchStr(\tilde{P}, \min(\alpha, m - q), \tilde{T}, \alpha)$;
  $mask \leftarrow MoveMask(\tilde{S})$; $sum \leftarrow 0$;
  while $mask \neq 0$ **do**
    $j \leftarrow ctz(mask) + 1$; $mask \leftarrow mask \gg j$; $sum \leftarrow sum + j$;
    if $P \approx T[i + sum - 1 : i + sum + m - 2]$ **then output** $i + sum - 1$;
  $i \leftarrow i + \alpha$;
Check remaining text naively;

SIMD Instructions Table 1 shows the functions and SIMD instructions used in this paper. We explain some non-trivial functions in it. The function $MoveMask(\tilde{X})$ returns a bit mask that consists of the most significant bit of each value in a sequence $\tilde{X}$, i.e., it returns $\sum_{i=1}^{\alpha} \left(\lfloor \tilde{X}[i] \cdot 2^{1-w} \rfloor \cdot 2^{i-1} \right)$. For two sequences $\tilde{X}_P$ and $\tilde{X}_T$, and two integers $1 \le m, n \le \alpha$, the function $SearchStr(\tilde{X}_P, m, \tilde{X}_T, n)$ returns a sequence $\tilde{S}$ such that

$$\tilde{S}[i] = \begin{cases} 2^w - 1 & \left(\begin{array}{ll} \tilde{X}_P[1 : m] = \tilde{X}_T[i : i + m - 1] & (1 \le i \le n - m + 1), \ or \\ \tilde{X}_P[1 : n - i + 1] = \tilde{X}_T[i : n] & (n - m + 1 < i \le n) \end{array} \right) \\ 0 & (otherwise) \end{cases}$$

for $1 \le i \le \alpha$. Note that this function compares the prefix of $\tilde{X}_P$ and the suffix of $\tilde{X}_T$ for $n - m + 1 < i \le n$.

Order-preserving Matching Algorithm Using SIMD Instructions We propose a fast algorithm, called *order-preserving matching filtration technique using SIMD instructions with q-NR code* (shortly OPFSI$_q$), using the functions in Table 1. The OPFSI$_q$

Algorithm 2: $Encode_q(X, i)$

Input: A sequence $X \in \Sigma^*$, and a position i.
Output: $B_q(X[i : i + \alpha - 1])$.

```
1 X̃ ← X[i : i + α − 1];    K̃ ← (0, 0, ..., 0)    /* |K̃| = α */
2 for j ← 1 to q do
3     X̃_j ← X[i + j : i + j + α − 1];    C̃_j ← CompLt(X̃, X̃_j);
4     M̃_j ← (2^{q−j}, 2^{q−j}, ..., 2^{q−j})    /* |M̃_j| = α */
5     K̃_j ← And(C̃_j, M̃_j);    K̃ ← Or(K̃, K̃_j);
6 return K̃;
```

algorithm is presented in Algorithm 1 and 2. The function $ctz(b)$ returns the number of trailing zeros in b from the least significant bit. For example, $ctz((11000100)_2) = 2$. This function can be computed fast by using the `bsf` instruction in x86.

In this algorithm, a chunk of length α of the text is processed by SIMD instructions. The chunk of the text is encoded to the q-NR code of length α using function $Encode_q$ shown in Algorithm 2. The matching between the encoded chunk and the encoded pattern is performed in line 6 of Algorithm 1, by a SIMD instruction. The bit mask represents candidate positions. For example, if $mask = (01000001)_2$ then candidates are i and $i + 7$.

Our algorithm has some weaknesses in the following two cases.

(1) The case that the pattern is long so that $m > \alpha - q$. In this case, this algorithm uses the prefix of $B_q(P)$, and only checks whether the prefix of $B_q(P)$ matches with $B_q(T[i : i + \alpha - 1])$ or not.
(2) The case that a subsequence order-isomorphic to the pattern is split in two or more chunks of text. In this case, this algorithm only checks whether the prefix of $B_q(P)$ matches to the suffix of $B_q(T[i : i + \alpha - 1])$ (see the detail of *SearchStr* function).

In these two cases, this algorithm can find all correct positions by Lemma 2, since $B_q(X) = B_q(Y) \Rightarrow B_q(X)[1 : i] = B_q(Y)[1 : i]$ for $1 \leq i \leq |X| - q$, although the number of candidates increases.

The main difference between our algorithm and the algorithm proposed by Faro and Külekci [7] is utilizing SIMD instructions. Their algorithm encodes the text naively and finds candidates based on the SBNDM2 [6] string matching algorithm, while our algorithm encodes α elements at once and finds candidates by the SIMD instruction.

Optimized Implementation Assume that $w = 16$ and $q \leq 8$. In this case, an 8-bit integer is enough to handle each value of q-NR code, although a 16-bit integer is used by the naive implementation of Algorithm 1. Therefore, we can pack two encoded chunks $\tilde{T}_1 = B_q(T[i : i + \alpha - 1])$ and $\tilde{T}_2 = B_q(T[i + \alpha : i + 2\alpha - 1])$ into one 128-bit register, by extracting the lower 8-bits of each value of $\tilde{T}_1$ and $\tilde{T}_2$. Then, we can perform a matching between $B_q(T[i : i + 2\alpha - 1])$ and the encoded pattern by *SearchStr* function.

In order to implement it, we encode two chunks in each iteration, and change the loop step size from α into 2α. Extracting the lower 8-bits of each value and packing into one 128-bit register can be implemented by the `_mm_shuffle_epi8` and the

`_mm_or_si128` instructions. This optimization technique is very effective because the `_mm_cmpestrm` instruction (*SearchStr* function) is much slower than other instructions [8]. This technique can be used in a similar way in the case of $w = 32$.

5 Experimental Results

We performed experiments comparing running time of our algorithm with that of previous work [2–4, 7]. We used a machine with Intel Xeon E5-2640 processor and 128GB memory on Ubuntu 14.04LTS. We implemented the OPFSI$_q$ algorithm [1] in C++, and compiled with gcc 4.8.4 . The OPFSI$_q$ algorithm was implemented by SSE4.2 intrinsic functions. Compiler options were `-O3 -msse4.2`.

We used two kinds of text data. One was random text data, consisting of 1000000 random integers of range 1 to 100. The other was temperature text data in Sendai city, consisting of 32452 integers. From a text data, 100 patterns were randomly chosen, and we computed the average running time of 100 runs for each pattern.

The algorithms proposed by Chhabra and Tarhio [4], Faro and Külekci [7], Cantone *et al.* [2] and Chhabra *et al.* [3] are respectively denoted as CT14, FK15, CFK15, and CKT15. These algorithms are implemented by themselves [2]. Similarly to previous work [2, 7], CT14 is based on the SBNDM2 algorithm. CKT15 utilizes SSE4.2 and AVX instruction set.

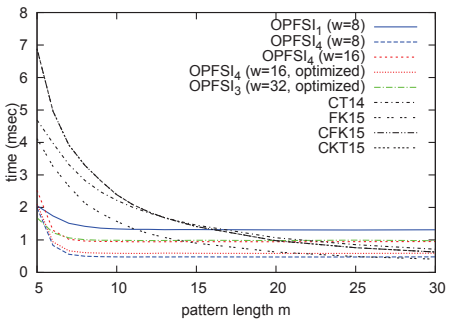

Fig. 2. Running times on random data.

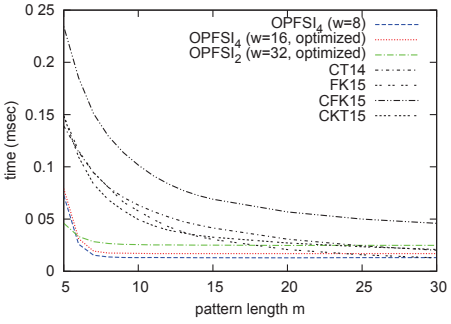

Fig. 3. Running times on temperature data.

Results are shown in Fig. 2 and Fig. 3. In FK15 and CFK15, the best results are selected among various parameters. The parameter w in OPFSI$_q$ means that the text data are treated as either w-bit integers ($w = 8, 16$) or w-bit floating points ($w = 32$), and *optimized* denotes that the optimization technique described in Section 4.1 is applied. In Fig. 2 and Fig. 3, we show the best results of the OPFSI$_q$ algorithm with $w = 8, 16, 32$. Furthermore, in Fig. 2 we also show results of OPFSI$_1$ ($w = 8$) and OPFSI$_4$ ($w = 16$) in order to discuss the effectiveness of the OPFSI$_q$ algorithm.

[1] Our implementations can be downloaded from `http://www.shino.ecei.tohoku.ac.jp/member/youhei_ueki/sofsem2016`

[2] We acknowledge the authors for sharing their program.

The OPFSI_q algorithm runs extremely faster for short patterns. Especially, for OPFSI_4 ($w = 8$) and $m = 7$, the algorithm runs approximately 4.7 times faster than all the previous work on the random text data. Comparisons between the results of OPFSI_1 ($w = 8$) with OPFSI_4 ($w = 8$), and OPFSI_4 ($w = 16$) with OPFSI_4 ($w = 16$, optimized) respectively show the effectiveness of q-NR filtration and the optimization technique described above. All previous work runs faster as the pattern length increases, whereas the OPFSI_q algorithm does not. This is because the OPFSI_q algorithm runs in linear time on average.

6 Conclusion

We proposed an effective filtration for the order-preserving matching problem using SIMD instructions, and confirmed that it practically runs faster than existing methods.

We used the SIMD instruction set SSE4.2 that supports 128 bit registers. We expect that it will become faster if we use other SIMD instruction sets that support wider registers, such as AVX2 supporting 256 bit registers and AVX-512 supporting 512 bit registers.

Acknowledgments. This work was supported by ImPACT Program of Council for Science, Technology and Innovation (Cabinet Office, Government of Japan), and KAKENHI Grant Numbers 25240003 and 15H05706.

References

1. Belazzougui, D., Pierrot, A., Raffinot, M., Vialette, S.: Single and multiple consecutive permutation motif search. In: ISAAC. 66–77 (2013)
2. Cantone, D., Faro, S., Külekci, M.O.: An efficient skip-search approach to the order-preserving pattern matching problem. 22–35 (2015)
3. Chhabra, T., Külekci, M.O., Tarhio, J.: Alternative algorithms for order-preserving matching. In: PSC. 36–46 (2015)
4. Chhabra, T., Tarhio, J.: Order-preserving matching with filtration. In: SEA. 307–314 (2014)
5. Cho, S., Na, J.C., Park, K., Sim, J.S.: A fast algorithm for order-preserving pattern matching. Information Processing Letters **115**(2) 397–402 (2015)
6. Durian, B., Holub, J., Peltola, H., Tarhio, J.: Improving practical exact string matching. Information Processing Letters 110(4) 148–152 (2010)
7. Faro, S., Külekci, M.O.: Efficient algorithms for the order preserving pattern matching problem. arXiv:1501.04001 (2015)
8. Intel Corporation: Intel (R) 64 and IA-32 Architectures Optimization Reference Manual. (2014)
9. Kim, J., Eades, P., Fleischer, R., Hong, S.H., Iliopoulos, C.S., Park, K., Puglisi, S.J., Tokuyama, T.: Order-preserving matching. Theoretical Computer Science 525(13) 68–79 (2014)
10. Knuth, D.E., Jr., J.H.M., Pratt, V.R.: Fast pattern matching in strings. SIAM Journal on Computing 6(2) 323–350 (1977)
11. Kubica, M., Kulczynski, T., Radoszewski, J., Rytter, W., Walen, T.: A linear time algorithm for consecutive permutation pattern matching. Information Processing Letters 113(12) 430–433 (2013)