Contents lists available at SciVerse ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

A linear time algorithm for consecutive permutation pattern matching

M. Kubica [a], T. Kulczyński [a], J. Radoszewski [a,*], W. Rytter [a,b,1], T. Waleń [c,a]

[a] Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, ul. Banacha 2, 02-097 Warsaw, Poland
[b] Faculty of Mathematics and Computer Science, Copernicus University, ul. Chopina 12/18, 87-100 Toruń, Poland
[c] Laboratory of Bioinformatics and Protein Engineering, International Institute of Molecular and Cell Biology in Warsaw, Poland

ARTICLE INFO

Article history:
Received 14 July 2012
Received in revised form 10 March 2013
Accepted 22 March 2013
Available online 26 March 2013
Communicated by A. Muscholl

Keywords:
Permutation pattern matching
Pattern matching
Knuth–Morris–Pratt algorithm
Analysis of algorithms

ABSTRACT

We say that two sequences x and w of length m are order-isomorphic (of the same "shape") if $w[i] \leqslant w[j]$ if and only if $x[i] \leqslant x[j]$ for each $i, j \in [1, m]$. We present a simple linear time algorithm for checking if a given sequence y of length n contains a factor which is order-isomorphic to a given pattern x. A factor is a subsequence of consecutive symbols of y, so we call our problem the consecutive permutation pattern matching. The (general) permutation pattern matching problem is related to general subsequences and is known to be NP-complete. We show that the situation for consecutive subsequences is significantly different and present an $O(n+m)$ time algorithm under a natural assumption that the symbols of x can be sorted in $O(m)$ time, otherwise the time is $O(n + m \log m)$. In our algorithm we use a modification of the classical Knuth–Morris–Pratt string matching algorithm.

1. Introduction

The problem of consecutive permutation pattern matching is a natural extension of the classical permutation pattern matching and a special variant of the so-called generalized permutation patterns. Several combinatorial results for this problem were known, see e.g. Elizalde and Noy [9], Warlimont [17,18]; see also Chapter 5 in [12]. However, there was no previous study of algorithmics of this problem. We present a linear time algorithm for consecutive permutation pattern matching.

Patterns in permutations are actively studied mostly from the combinatorial point of view. This field of study is concentrated on pattern avoidance, that is, counting the number of permutations not containing a subsequence which is order-isomorphic to a given pattern. Knuth considered permutations avoiding the pattern 312 [13], Lovász considered permutations avoiding the pattern 213 [14], and Rotem those that do not contain 231 nor 312 [15], just to mention a few most famous examples.

There are several algorithmic results related to pattern matching in permutations. Bose et al. [4] showed this problem to be NP-complete. Denote by m and n the length of the pattern and the text. A general algorithm with $O(n^{0.47m+o(m)})$ time complexity was given in [1], and an $O^*(1.79^n)$ time algorithm was recently given in [6]. For several special cases polynomial time algorithms are known. In [4] an $O(mn^6)$ time and $O(mn^4)$ space algorithm for the case of a separable pattern is given. A permutation is separable if it avoids the patterns 2413 and 3142. Afterwards, Ibarra [11] improved this result to $O(mn^4)$ time and $O(mn^3)$ space. If both the text and the pattern avoid the permutation 321, an $O(m^2n^6)$ time algorithm is known [10]. Note that the case of an increasing pattern can be reduced to searching for the longest increasing subsequence, which can be done in $O(n \log\log n)$ time for

* Corresponding author. Tel.: +48 22 55 44 484; fax: +48 22 55 44 400.
E-mail addresses: kubica@mimuw.edu.pl (M. Kubica), tomasz.kulczynski@students.mimuw.edu.pl (T. Kulczyński), jrad@mimuw.edu.pl (J. Radoszewski), rytter@mimuw.edu.pl (W. Rytter), walen@mimuw.edu.pl (T. Waleń).

[1] The author is supported by grant No. N206 566740 of the National Science Centre.

permutations [7]. Another simpler case, when the permutation pattern has length 4, was shown in [2] to be solvable in $O(n \log n)$ time.

Generalized permutation patterns (also called vincular patterns, see [12]) were introduced by Babson and Steingrímsson [3] and have proved to have connections to a variety of other combinatorial structures, see the survey [16]. A generalized pattern is a sequence in which two adjacent symbols may or may not be separated by a dash. The absence of a dash between two adjacent symbols in a pattern imposes an additional requirement that the corresponding symbols in the text must be adjacent. Thus an ordinary permutation pattern $p_1 p_2 p_3 \ldots p_k$ corresponds to a generalized pattern of the form $p_1\text{-}p_2\text{-}p_3\text{-}\cdots\text{-}p_k$. On the other hand, a generalized permutation pattern without dash represents a consecutive pattern, that must form a factor of the text (less common names: segmented pattern, segmental pattern, subword pattern, see [12]). Combinatorial properties of consecutive permutation patterns were considered in [9,17,18]. No previous algorithmic results related to consecutive patterns were known (as for the generalized patterns, only a W[1]-completeness result was given in [5]).

We present a linear time algorithm for permutation pattern matching of consecutive patterns. Our algorithm is based on a simple, yet non-trivial, modification of the Morris–Pratt pattern matching algorithm for strings.

2. Order-isomorphism

We consider sequences over an integer alphabet Σ, $x \in \Sigma^*$. The positions in x are numbered from 1 to $|x|$. Two sequences x, y of the same length are called *order-isomorphic* (or simply isomorphic), written $x \approx y$, if

$$\big(\forall 1 \leqslant i, j \leqslant |x|\big) \quad x[i] \leqslant x[j] \quad \Leftrightarrow \quad y[i] \leqslant y[j].$$

For example, $4\,1\,4\,7\,3\,5\,2\,3\,4 \approx 8\,1\,8\,10\,6\,9\,4\,6\,8$. In this section we show a linear time algorithm for checking isomorphism of two sequences.

For $i = 1, \ldots, |x|$ define

$$LMax_x[i] = j$$
$$\text{if } x[j] = \max\big\{x[k]\colon\ k \in [1, i-1],\ x[k] \leqslant x[i]\big\},$$

if there is no such j then $LMax_x[i] = 0$, similarly define

$$LMin_x[i] = j$$
$$\text{if } x[j] = \min\big\{x[k]\colon\ k \in [1, i-1],\ x[k] \geqslant x[i]\big\},$$

and $LMin_x[i] = 0$ if no such j exists. If several equally good values of j exist, an arbitrary one can be selected (we select the greatest good value of j). The *LMax* and *LMin* tables are called *location tables*, see Table 1. If the pattern is unambiguous then we omit the index in the notation.

In Lemma 1 we show that location tables can be computed as fast as sorting all the symbols of the pattern.

Lemma 1. *Let x be a sequence of length m and let $sort(x)$ be the time required to sort all the elements of x. Then location tables of x can be computed in $O(sort(x))$ time.*

Table 1
The location tables for the pattern $x = 4\,1\,4\,7\,3\,5\,2\,3\,4$.

i	1	2	3	4	5	6	7	8	9
$x[i]$	4	1	4	7	3	5	2	3	4
$LMax[i]$	0	0	1	3	2	3	2	5	3
$LMin[i]$	0	1	1	0	3	4	5	5	3

Table 2
Computation of the *LMax* table for the pattern from Table 1, as in the proof of Lemma 1.

$x[S[i]]$	1	2	3	3	4	4	4	5	7
$S[i]$	**2**	**7**	**5**	**8**	**1**	**3**	**9**	**6**	**4**
$LMax[S[i]]$	0	2	2	5	0	1	3	3	3

Proof. Let us sort positions of x with respect to their contents (the symbols they contain). In case of equal contents the smaller positions come first. Let S be the resulting sequence of positions. Then $LMax[j]$ is the nearest smaller value to the left of $S[i] = j$ (if there is no such value, $LMax[j] = 0$), see Table 2. The *LMin* table is computed similarly, by taking nearest smaller value to the right in a sequence S' constructed exactly as the sequence S but with a reversed order of positions with equal contents.

It is folklore knowledge that the problem of computing nearest smaller values for all elements of a sequence, also known as the "all nearest smaller values" problem, can be solved in linear time by a stack-based algorithm. $\square$

The following lemma provides a justification for introducing the location tables in the context of consecutive permutation pattern matching.

Lemma 2. *Assume that*

$$x[1\,..\,t] \approx y[1\,..\,t], \quad t < |x|, |y|$$
$$\text{and} \quad a = LMax_x[t+1], \qquad b = LMin_x[t+1].$$

Then

$$x[1\,..\,t+1] \approx y[1\,..\,t+1] \quad \Leftrightarrow \quad y[a] \leqslant y[t+1] \leqslant y[b].$$

In case a or b is equal to 0, we omit the respective inequality in the condition.

Proof. ($\Rightarrow$) By the definition of the location tables, we have $x[a] \leqslant x[t+1] \leqslant x[b]$. Now order-isomorphism of $x[1\,..\,t+1]$ and $y[1\,..\,t+1]$ implies that $y[a] \leqslant y[t+1] \leqslant y[b]$.

($\Leftarrow$) We need to show that $x[1\,..\,t+1] \approx y[1\,..\,t+1]$. We have $x[1\,..\,t] \approx y[1\,..\,t]$, hence it suffices to prove that, for $i \leqslant t$,

$$x[i] \leqslant x[t+1] \quad \Leftrightarrow \quad y[i] \leqslant y[t+1].$$

Assume that $x[i] \leqslant x[t+1]$ for some $i \in \{1, \ldots, t\}$. By the definition of the *LMax* table, we have $x[i] \leqslant x[a]$; by the order-isomorphism of $x[1\,..\,t]$ and $y[1\,..\,t]$, we have $y[i] \leqslant y[a]$; finally, by the assumption of the lemma, $y[a] \leqslant y[t+1]$, hence $y[i] \leqslant y[t+1]$. In a similar way we show that $x[i] \geqslant x[t+1]$ implies $y[i] \geqslant y[t+1]$, which yields the requested equivalence. $\square$

Let us make a natural assumption that the symbols of x can be sorted in $O(m)$ time, e.g. they are elements of the

Table 3
The order-borders table P for the pattern $x = 2\,5\,1\,4\,7\,3\,6\,8$.

i	1	2	3	4	5	6	7	8
$x[i]$	2	5	1	4	7	3	6	8
$P[i]$	0	1	1	2	2	3	4	5

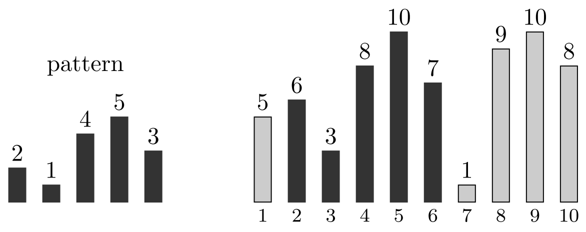

Fig. 1. An order-occurrence of the pattern 2 1 4 5 3 in the text 5 6 3 8 10 7 1 9 10 8. There is also a second order-occurrence of this pattern formed by the last 5 symbols of the text.

set $\{1, \ldots, m^{O(1)}\}$. Under this assumption, Lemma 2 (together with Lemma 1) implies an $O(1)$ time incremental criterion for checking if a sequence is isomorphic to a prefix of the pattern. This is the basic tool used in the pattern matching algorithm presented in the next section:

Lemma 3. *Let x be a pattern of length m whose symbols can be sorted in $O(m)$ time. After $O(m)$ time preprocessing one can answer queries of the following form: "assuming that $x[1\,..\,t] \approx y[1\,..\,t]$, check if $x[1\,..\,t+1] \approx y[1\,..\,t+1]$" for any sequence y in constant time.*

3. Consecutive permutation pattern matching

Let x be a pattern of length m. The *order-borders table* P for x is defined as follows:

$$P[1] = 0,$$

$$P[i] = \max\bigl\{j < i\colon x[1\,..\,j] \approx x[i-j+1\,..\,i]\bigr\} \quad \text{for } i \geqslant 2,$$

see Table 3 as an example.

The algorithm computing the order-borders table is similar to the algorithm computing (regular) borders in the Morris–Pratt algorithm.

```
Algorithm Compute the table P
    $P[0] := -1$; $t := -1$;
    for $i := 1$ to $m$ do
            invariant: $x[1\,..\,t] \approx x[i-t\,..\,i-1]$
        while $t \geqslant 0$ and $x[1\,..\,t+1] \not\approx x[i-t\,..\,i]$ do
            $t := P[t]$;
        $t := t+1$; $P[i] := t$;
```

The test $x[1\,..\,t+1] \approx x[i-t\,..\,i]$ can be done in $O(1)$ time due to Lemma 3 and the invariant of the while-loop. The number of such tests is linear which follows from the complexity analysis of the Morris–Pratt algorithm (note that t decreases after each comparison). Consequently we obtain the following lemma.

Lemma 4. *The order-borders table can be computed in linear time.*

A pattern x of length m *order-occurs* at position i of a text y if $x \approx y[i+1\,..\,i+m]$, see also Fig. 1. Let n be the length of y. We can find all order-occurrences of x in y in linear time using the algorithm below (the pseudocode resembles the implementation of Morris–Pratt pattern matching algorithm as given in [8]).

```
Algorithm Modified algorithm of Morris and Pratt
  $i := 0$; $j := 0$;
  while $i \leqslant n - m$ do begin
              invariant: $x[1\,..\,j] \approx y[i+1\,..\,i+j]$
      while $j < m$ and $x[1\,..\,j+1] \approx y[i+1\,..\,i+j+1]$
          do $j := j+1$;
      if $j = m$ then write $i$;
      $i := i + (j - P[j])$; $j := \max(0, P[j])$;
  end
```

Theorem 5 summarizes the linear time algorithm for consecutive permutation pattern matching.

Theorem 5. *All order-occurrences of a pattern in a given text can be computed in linear time.*

Proof. By Lemma 4, the order-borders table for the pattern can be computed in linear time. Recall that this algorithm involves the computation of location tables, see Lemma 1.

The procedure for finding order-occurrences mimics the Morris–Pratt pattern matching algorithm, but instead of testing equality of symbols of the pattern and the text we check order-isomorphism of a prefix of the pattern and a factor of the text. Due to the invariant in the pseudocode, each such test can be done in constant time using Lemma 3. The number of remaining operations in the pattern matching is linear just as in the original Morris–Pratt algorithm. □

References

[1] S. Ahal, Y. Rabinovich, On complexity of the subpattern problem, SIAM J. Discrete Math. 22 (2) (2008) 629–649.
[2] M.H. Albert, R.E.L. Aldred, M.D. Atkinson, D.A. Holton, Algorithms for pattern involvement in permutations, in: P. Eades, T. Takaoka (Eds.), ISAAC, in: Lecture Notes in Comput. Sci., vol. 2223, Springer, 2001, pp. 355–366.
[3] E. Babson, E. Steingrímsson, Generalized permutation patterns and a classification of the Mahonian statistics, Sem. Lothar. Combin. 44 (2000).
[4] P. Bose, J.F. Buss, A. Lubiw, Pattern matching for permutations, Inform. Process. Lett. 65 (5) (1998) 277–283.
[5] M.-L. Bruner, M. Lackner, A W[1]-completeness result for generalized permutation pattern matching, CoRR, arXiv:1109.1951, 2011.
[6] M.-L. Bruner, M. Lackner, A fast algorithm for permutation pattern matching based on alternating runs, in: F.V. Fomin, P. Kaski (Eds.), SWAT, in: Lecture Notes in Comput. Sci., vol. 7357, Springer, 2012, pp. 261–270.
[7] M.-S. Chang, F.-H. Wang, Efficient algorithms for the maximum weight clique and maximum weight independent set problems on permutation graphs, Inform. Process. Lett. 43 (1992) 293–295.
[8] M. Crochemore, W. Rytter, Jewels of Stringology, World Scientific, 2003.
[9] S. Elizalde, M. Noy, Consecutive patterns in permutations, Adv. in Appl. Math. 30 (2003) 110–125.
[10] S. Guillemot, S. Vialette, Pattern matching for 321-avoiding permutations, in: Y. Dong, D.-Z. Du, O.H. Ibarra (Eds.), ISAAC, in: Lecture Notes in Comput. Sci., vol. 5878, Springer, 2009, pp. 1064–1073.
[11] L. Ibarra, Finding pattern matchings for permutations, Inform. Process. Lett. 61 (6) (1997) 293–295.

[12] S. Kitaev, Patterns in Permutations and Words, Monogr. Theoret. Comput. Sci. EATCS Ser., 2011.
[13] D.E. Knuth, The Art of Computer Programming, vol. I: Fundamental Algorithms, second ed., Addison–Wesley, 1973.
[14] L. Lovász, Combinatorial Problems and Exercises, North-Holland, 1979.
[15] D. Rotem, Stack sortable permutations, Discrete Math. 33 (2) (1981) 185–196.
[16] E. Steingrímsson, Generalized permutation patterns – a short survey, in: S. Linton, N. Ruskuc, V. Vatter (Eds.), Permutation Patterns, in: London Math. Soc. Lecture Note Ser., Cambridge Univ. Press, 2010, pp. 137–152.
[17] R. Warlimont, Permutations avoiding consecutive patterns, Ann. Univ. Sci. Budapest. Sect. Comput. 22 (2003) 373–393.
[18] R. Warlimont, Permutations avoiding consecutive patterns, II, Arch. Math. 84 (2005) 496–502.