

# Efficient Algorithms for the Order Preserving Pattern Matching Problem

Simone Faro<sup>1(✉)</sup> and M. Oğuzhan Külekci<sup>2</sup>

<sup>1</sup> Department of Mathematics and Computer Science,  
Università di Catania, Catania, Italy  
faro@dmi.unict.it

<sup>2</sup> Informatics Institute, Istanbul Technical University, Istanbul, Turkey  
kulekci@itu.edu.tr

**Abstract.** Given a pattern  $x$  of length  $m$  and a text  $y$  of length  $n$ , both over an ordered alphabet, the *order-preserving pattern matching* problem consists in finding all substrings of the text with the same relative order as the pattern. The OPPM, which might be viewed as an approximate variant of the well known *exact pattern matching* problem, has gained attention in recent years. This interesting problem finds applications in a lot of fields as from time series analysis, like share prices on stock markets or weather data analysis, to musical melody matching. In this paper we present two new filtering approaches which turn out to be much more effective in practice than the previously presented methods by reducing the number of false positives up to 99%. From our experimental results it turns out that our proposed solutions are up to 2 times faster than the previous solutions.

**Keywords:** Approximate text analysis · Experimental algorithms · Filtering algorithms · Text processing

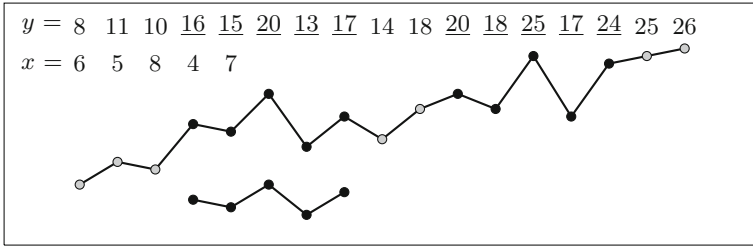
## 1 Introduction

Given a pattern  $x$  of length  $m$  and a text  $y$  of length  $n$ , both over a common alphabet  $\Sigma$ , the *exact string matching problem* consists in finding all occurrences of the string  $x$  in  $y$ . The *order-preserving pattern matching problem* [2,3,8,9] (OPPM in short) is an approximate variant of the exact pattern matching problem which has gained attention in recent years. In this variant the characters of  $x$  and  $y$  are drawn from an ordered alphabet  $\Sigma$  with a total order relation defined on it. The task of the problem is to find all substrings of the text with the same relative order as the pattern.

For instance the relative order of the sequence  $x = \langle 6, 5, 8, 4, 7 \rangle$  is the sequence  $\langle 3, 1, 0, 4, 2 \rangle$  since 6 has rank 3, 5 as rank 1, and so on. Thus  $x$  occurs in the string  $y = \langle 8, 11, 10, 16, 15, 20, 13, 17, 14, 18, 20, 18, 25, 17, 20, 25, 26 \rangle$  at position 3, since  $x$  and the subsequence  $\langle 16, 15, 20, 13, 17 \rangle$  share the same relative order. An other occurrence of  $x$  in  $y$  is at position 10 (see Fig. 1).

---

A preliminary version of this paper appeared in a technical report [8].



**Fig. 1.** Example of a pattern  $x$  of length 5 over an integer alphabet with two order preserving occurrences in a text  $y$  of length 17, at positions 3 and 10.

The OPPM problem finds applications in the fields where we are interested in finding patterns affected by relative orders, not by their absolute values. For example, it can be applied to time series analysis like share prices on stock markets, weather data or to musical melody matching of two musical scores.

In the last few years some solutions have been proposed for the order-preserving pattern matching problem. The first solution was presented by Kubica et al. [12] in 2013. They proposed a  $\mathcal{O}(n + m \log m)$  solution over generic ordered alphabets based on the Knuth-Morris-Pratt algorithm [11] and a  $\mathcal{O}(n + m)$  solution in the case of integer alphabets. Some months later Kim et al. [10] presented a similar solution running in  $\mathcal{O}(n + m \log m)$  time based on the KMP approach. Although Kim et al. stressed some doubts about the applicability of the Boyer-Moore approach [2] to order-preserving matching problem, in 2013 Cho et al. [7] presented a method for deciding the order-isomorphism between two sequences showing that the Boyer-Moore approach can be applied also to the order-preserving variant of the pattern matching problem. In addition in [1] the authors showed that the Aho-Corasik approach can be applied to the OPPM problem for searching a set of patterns.

Chhabra and Tarhio in 2014 presented a new practical solution [5, 6] based on filtration. Their algorithm translates the input sequences in two binary sequences and then use any standard exact pattern matching algorithm as a filtration procedure. In particular in their approach a sequence  $s$  is translated in a binary sequence  $\beta$  of length  $|s| - 1$  according to the following position

$$\beta[i] = \begin{cases} 1 & \text{if } s[i] \geq s[i + 1] \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

for each  $0 \leq i < |s| - 1$ . This translation is unique for a given sequence  $s$  and can be performed on line on the text, requiring constant time for each text character.

Thus when a candidate occurrence is found during the filtration phase an additional verification procedure is run in order to check for the order-isomorphism of the candidate substring and the pattern. Despite its quadratic time complexity, this approach turns out to be simpler and more effective in practice than earlier solutions. It is important to notice that any algorithm for exact string matching can be used as a filtration method. The authors also proved

that if the underlying filtration algorithm is sublinear and the text is translated on line, the complexity of the algorithm is sublinear on average. Experiments conducted in [5] show that the approach is faster than the algorithm by Cho *et al.*

In 2015 other efficient filtration algorithms have been presented. Specifically, in [3] the authors presented an efficient solution which combines the Skip-Search approach for searching and a multiple hash function technique for reducing the number of false positives during each bucket inspection. Additionally, in [4] the authors proposed two filtration online solutions based on SSE and AVX instructions, respectively. It turns out from experimental results conducted in [3] and in [4] that such solutions are faster than the previous algorithms in most cases.

In this paper we present two new families of filtering approaches which turn out to be much more effective in practice than the previously presented methods. While the technique proposed by Chhabra and Tarhio translates the input strings in binary sequences, our methods work on sequences over larger alphabets in order to speed up the searching process and reduce the number of false positives. From our experimental results it turns out that our proposed solutions are up to 2 times faster than the previous solutions reducing the number of false positives up to 99% under suitable conditions.

The paper is organized as follows. In Sect. 2 we give preliminary notions and definitions relative to the order-preserving pattern matching problem. Then we present our new solutions in Sect. 3 and evaluate their performances against the previous algorithms in Sect. 4.

## 2 Notions and Basic Definitions

A string  $x$  over an ordered alphabet  $\Sigma$ , of size  $\sigma$ , is defined as a sequence of elements in  $\Sigma$ . We shall assume that a total order relation " $\leq$ " is defined on it.

By  $|x|$  we denote the length of a string  $x$ . We refer to the  $i$ -th element in  $x$  as  $x[i]$  and use the notation  $x[i..j]$  to denote the subsequence of  $x$  from the element at position  $i$  to the element at position  $j$  (including the extremes), where  $0 \leq i \leq j < |x|$ . We say that two (nonnull) sequences  $x, y$  over  $\Sigma$  are order-isomorphic if the relative order of their elements is the same. More formally:

**Definition 1 (Order-isomorphism).** *Two nonnull sequences  $x, y$  of the same length, over a totally ordered alphabet  $(\Sigma, \leq)$ , are said to be order-isomorphic, and we write  $x \approx y$ , if the following condition holds*

$$x[i] \leq x[j] \iff y[i] \leq y[j], \text{ for } 0 \leq i, j < |x|.$$

From a computational point of view, it is convenient to characterize the order of a sequence by means of two functions: the *rank* and the *equality* functions.

**Definition 2 (Rank function).** *Let  $x$  be a nonnull sequence over a totally ordered alphabet  $(\Sigma, \leq)$ . The rank function of  $x$  is the bijection from  $\{0, 1, \dots, |x| - 1\}$  onto itself defined, for  $0 \leq i < |x|$ , by*

$$rk_x(i) = |\{k : x[k] < x[i] \text{ or } (x[k] = x[i] \text{ and } k < i)\}|.$$

The following property is a trivial consequence of the Definition 2.

**Corollary 1.** *Let  $x$  be a non-null sequence drawn over a totally ordered alphabet  $(\Sigma, \leq)$ . Then we have  $x[rk_x^{-1}(i)] \leq x[rk_x^{-1}(i + 1)]$ , for  $0 \leq i < |x| - 1$ . ■*

Given any non-null sequence  $x$ , we shall refer to the corresponding non-null sequence  $\langle rk_x^{-1}(0), rk_x^{-1}(1), \dots, rk_x^{-1}(|x| - 1) \rangle$  as the *relative order* of  $x$  (see Example 1). From Corollary 1, it follows that the relative order of  $x$  can be computed in time proportional to the time required to (stably) sort  $x$ .

The rank function alone allows one to characterize order-isomorphic sequences only when characters are pairwise distinct. To handle the more general case in which multiple occurrences of the same character are permitted, we also need the *equality function*, which is defined next.

**Definition 3 (Equality function).** *Let  $x$  be a sequence of length  $m \geq 2$  over a totally ordered alphabet  $(\Sigma, \leq)$ . The equality function of  $x$  is the binary map  $eq_x: \{0, 1, \dots, m - 2\} \rightarrow \{0, 1\}$  where, for  $0 \leq i \leq m - 2$ ,*

$$eq_x(i) = \begin{cases} 1 & \text{if } x[rk_x^{-1}(i)] = x[rk_x^{-1}(i + 1)] \\ 0 & \text{otherwise.} \end{cases}$$

**Lemma 1.** *For any two sequences  $x$  and  $y$  of the same length  $m \geq 2$ , over a totally ordered alphabet,  $x \approx y$  if and only if  $rk_x = rk_y$  and  $eq_x = eq_y$ . ■*

*Example 1.* Let  $x = \langle 6, 3, 8, 3, 10, 7, 10 \rangle$ ,  $y = \langle 2, 1, 4, 1, 5, 3, 5 \rangle$ , and let also  $z = \langle 6, 3, 8, 4, 9, 7, 10 \rangle$ . They have the same rank function  $\langle 2, 0, 4, 1, 5, 3, 6 \rangle$  and, therefore, the same relative order  $\langle 1, 3, 0, 5, 2, 4, 6 \rangle$ . However,  $x$  and  $y$  are order-isomorphic, whereas  $x$  and  $z$  (as well as  $y$  and  $z$ ) are not. Notice that we have  $eq_x = eq_y = \langle 1, 0, 0, 0, 0, 1 \rangle$  and  $eq_z = \langle 0, 0, 0, 0, 0, 0 \rangle$ .

Thus in order to establish whether two given sequences of the same length  $m$  are order-isomorphic, it is enough to compute their rank and equality functions. The cost of the test is dominated by the cost  $\mathcal{O}(m \log m)$  of sorting the sequences.

**Lemma 2.** *Let  $x$  and  $y$  be two sequences of the same length  $m \geq 2$ , over a totally ordered alphabet. Then  $x \approx y$  if and only if the following conditions hold:*

- (i)  $y[rk_x^{-1}(i)] \leq y[rk_x^{-1}(i + 1)]$ , for  $0 \leq i < m - 1$
- (ii)  $y[rk_x^{-1}(i)] = y[rk_x^{-1}(i + 1)]$  if and only if  $eq_x(i) = 1$ , for  $0 \leq i < m - 1$ . ■

Based on Lemma 2, the procedure ORDER-ISOMORPHIC verifies correctly whether a sequence  $y$  is order-isomorphic to a sequence  $x$  of the same length as  $y$ . It receives as input the functions  $rk_x$  and  $eq_x$  and the sequence  $y$ , and returns true if  $x \approx y$ , false otherwise. A mismatch occurs when one of the three conditions of lines 2, 3, or 4 holds. Notice that the time complexity of the procedure ORDER-ISOMORPHIC is linear in the size of its input sequence  $y$ .

**Definition 4 (Order-preserving pattern matching).** *Let  $x$  and  $y$  be two sequences of length  $m$  and  $n$ , respectively, with  $n > m$ , both over an ordered alphabet  $(\Sigma, \leq)$ . The order-preserving pattern matching problem consists in finding all positions  $i$ , with  $0 \leq i \leq n - m$ , such that  $y[i .. i + m - 1] \approx x$ .*

If  $y[i .. i + m - 1] \approx x$ , we say that  $x$  has an *order-preserving occurrence* in  $y$  at position  $i$ .

### 3 New Efficient Filter Based Algorithms

In this section we present two new general approaches for the OPPM problem. Both of them are based on a filtration technique, as in [5], but we use information extracted from groups of integers in the input string, as in [7], in order to make the filtration phase more effective in terms of efficiency and accuracy.

In our approaches we make use of the following definition of  $q$ -neighborhood of an element in an integer string.

**Definition 5 ( $q$ -neighborhood).** *Given a string  $x$  of length  $m$ , we define the  $q$ -neighborhood of the element  $x[i]$ , with  $0 \leq i < m - q$ , as the sequence of  $q + 1$  elements from position  $i$  to  $i + q$  in  $x$ , i.e. the sequence  $\langle x[i], x[i + 1], \dots, x[i + q] \rangle$ .*

The *accuracy* of a filtration method is a value indicating how many false positives are detected during the filtration phase, i.e. the number of candidate occurrences detected by the filtration algorithm which are not real occurrences of the pattern. The *efficiency* is instead related with the time complexity of the procedure we use for managing grams and with the time efficiency of the overall searching algorithm.

When using  $q$ -grams, a great accuracy translates in involving greater values of  $q$ . However, in this context, the value of  $q$  represents a trade-off between the computational time required for computing the  $q$ -grams for each window of the text and the computational time needed for checking false positive candidate occurrences. The larger is the value of  $q$ , the more time is needed to compute each  $q$ -gram. On the other hand, the larger is the value of  $q$ , the smaller is the number of false positives the algorithm finds along the text during the filtration.

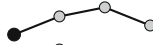
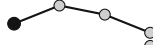
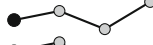
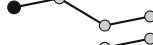
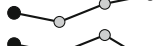
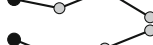

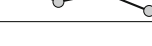
#### 3.1 The Neighborhood Ranking Approach

Given a string  $x$  of length  $m$ , we can compute the relative position of the element  $x[i]$  compared with the element  $x[j]$  by querying the inequality  $x[i] \geq x[j]$ . For brevity we will write in symbol  $\beta_x(i, j)$  to indicate the boolean value resulting from the above inequality, extending the formal definition given in Eq. (1). Formally we have

$$\beta_x(i, j) = \begin{cases} 1 & \text{if } x[i] \geq x[j] \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

It is easy to observe that if  $\beta_x(i, j) = 1$  we have that  $rk_x^{-1}(i) \geq rk_x^{-1}(j)$  ( $x[j]$  precedes  $x[i]$  in the ordering of the elements of  $x$ ), otherwise  $rk_x^{-1}(i) < rk_x^{-1}(j)$ .

The neighborhood ranking (NR) approach associates each position  $i$  of the string  $x$  (where  $0 \leq i < m - q$ ) with the sequence of the relative positions between  $x[i]$  and  $x[i + j]$ , for  $j = 1, \dots, q$ . In other words we compute the binary sequence  $\langle \beta_x(i, i + 1), \beta_x(i, i + 2), \dots, \beta_x(i, i + q) \rangle$  of length  $q$  indicating the relative positions of the element  $x[i]$  compared with other values in its  $q$ -neighborhood. Of course, we do not include in the sequence the relative position of  $\beta(i, i)$ , since it doesn't give any additional information.

NEIGHBORHOOD RANKING	EXAMPLE	NR SEQ.	$\chi_x^3[i]$
$x[i] \leq x[i + 1], x[i + 2], x[i + 3]$		$\langle 0, 0, 0 \rangle$	0
$x[i + 3] \leq x[i] \leq x[i + 1], x[i + 2]$		$\langle 0, 0, 1 \rangle$	1
$x[i + 2] \leq x[i] \leq x[i + 1], x[i + 3]$		$\langle 0, 1, 0 \rangle$	2
$x[i + 2], x[i + 3] \leq x[i] \leq x[i + 1]$		$\langle 0, 1, 1 \rangle$	3
$x[i + 1] \leq x[i] \leq x[i + 2], x[i + 3]$		$\langle 1, 0, 0 \rangle$	4
$x[i + 1], x[i + 3] \leq x[i] \leq x[i + 2]$		$\langle 1, 0, 1 \rangle$	5
$x[i + 1], x[i + 2] \leq x[i] \leq x[i + 3]$		$\langle 1, 1, 0 \rangle$	6
$x[i + 1], x[i + 2], x[i + 3] \leq x[i]$		$\langle 1, 1, 1 \rangle$	7

**Fig. 2.** The  $2^3$  possible 3-neighborhood ranking sequences associated with element  $x[i]$ , and their corresponding NR value. In the leftmost column we show the ranking position of  $x[i]$  compared with other elements in its neighborhood  $\langle x[i], x[i + 1], x[i + 2], x[i + 3] \rangle$ .

Since there are  $2^q$  possible configurations of a binary sequence of length  $q$  the string  $x$  is converted in a sequence  $\chi_x^q$  of length  $m - q$ , where each element  $\chi_x^q[i]$ , for  $0 \leq i < m - q$ , is a value such that  $0 \leq \chi_x^q[i] < 2^q$  (Fig. 2).

More formally we have the following definition

**Definition 6 ( $q$ -NR sequence).** Given a string  $x$  of length  $m$  and an integer  $q < m$ , the  $q$ -NR sequence associated with  $x$  is a numeric sequence  $\chi_x^q$  of length  $m - q$  over the alphabet  $\{0, \dots, 2^q\}$  where

$$\chi_x^q[i] = \sum_{j=1}^q (\beta_x(i, i + j) \times 2^{q-j}), \text{ for all } 0 \leq i < m - q$$

*Example 2.* Let  $x = \langle 5, 6, 3, 8, 10, 7, 1, 9, 10, 8 \rangle$  be a sequence of length 10. The 4-neighborhood of the element  $x[2]$  is the subsequence  $\langle 3, 8, 10, 7, 1 \rangle$ . Observe that  $x[2]$  is greater than  $x[6]$  and less than all other values in its 4-neighborhood. Thus the ranking sequence associated with the element of position 2 is  $\langle 0, 0, 0, 1 \rangle$  which translates in a NR value equal to 1. In a similar way we can observe that the NR sequence associated with the element of position 3 is  $\langle 0, 1, 1, 0 \rangle$  which translates in a NR value equal to 6. The whole 4-NR sequence of length 6 associated to  $x$  is  $\chi_x^4 = \langle 4, 8, 1, 6, 15, 8 \rangle$ .

The following Lemma 3 and Corollary 2 prove that the NR approach can be used to filter a text  $y$  in order to search for all order preserving occurrences of a pattern  $x$ , i.e.  $\{i \mid x \approx y[i \dots i + m - 1]\} \subseteq \{i \mid \chi_x^q = \chi_y^q[i \dots i + m - k]\}$ .

**Lemma 3.** Let  $x$  and  $y$  be two sequences of length  $m$  and let  $\chi_x^q$  and  $\chi_y^q$  the  $q$ -ranking sequences associated to  $x$  and  $y$ , respectively. If  $x \approx y$  then  $\chi_x^q = \chi_y^q$ .

*Proof.* Let  $rk$  be the rank function associated to  $x$  and suppose by hypothesis that  $x \approx y$ . Then the following statements hold

1. by Definition 2 we have  $x[rk_x^{-1}(i)] \leq x[rk_x^{-1}(i+1)]$ , for  $0 \leq i < m-1$ ;
2. by hypothesis and Definition 1,  $y[rk_x^{-1}(i)] \leq y[rk_x^{-1}(i+1)]$ , for  $0 \leq i < m-1$ ;
3. then by 1 and 2,  $x[i] \leq x[j]$  iff  $y[i] \leq y[j]$ , for  $0 \leq i, j < m-1$ ;
4. the previous statement implies that  $x[i] \geq x[i+j]$  iff  $y[i] \geq y[i+j]$  for  $0 \leq i < m-q$  and  $1 \leq j < q$ ;
5. by statement 4 we have that  $\beta_x(i, i+j) = \beta_y(i, j+j)$  for  $0 \leq i < m-q$  and  $1 \leq j < q$ ;
6. finally, by statement 5 and Definition 6,  $\chi_x^q[i] = \chi_y^q[i]$ , for  $0 \leq i < m-q$ .

This last statement proves the thesis. ■

The following corollary proves that the NR approach can be used as a filtering. It trivially follows from Lemma 3.

**Corollary 2.** *Let  $x$  and  $y$  be two sequences of length  $m$  and  $n$ , respectively. Let  $\chi_x^q$  and  $\chi_y^q$  the  $q$ -ranking sequences associated to  $x$  and  $y$ , respectively. If  $x \approx y[j \dots j+m-1]$  then  $\chi_x^q[i] = \chi_y^q[j+i]$ , for  $0 \leq i < m-q$ . ■*

Figure 4 (on the left) shows the procedure for computing the NR value associated with the element of the string  $x$  at position  $i$ . The time complexity of the procedure is  $\mathcal{O}(q)$ . Thus, given a pattern  $x$  of length  $m$ , a text  $y$  of length  $n$  and an integer value  $q < m$ , we can solve the OPPM problem by searching  $\chi_y^q$  for all occurrences of  $\chi_x^q$ , using any algorithm for the exact string matching problem. During the preprocessing phase we compute the sequence  $\chi_x^q$  and the functions  $rk_x$  and  $eq_x$ . When an occurrence of  $\chi_x^q$  is found at position  $i$  the verification procedure  $\text{ORDER-ISOMORPHIC}(inv-rk, eq, y[i \dots i+m-1])$  (shown in Fig. 3) is run in order to check if  $x \approx y[i \dots i+m-1]$ .

Since in the worst case the algorithm finds a candidate occurrence at each text position and each verification costs  $\mathcal{O}(m)$ , the worst case time complexity of the algorithm is  $\mathcal{O}(nm)$ , while the filtration phase can be performed with a  $\mathcal{O}(nq)$  worst case time complexity. However, following the same analysis of [5], we easily prove that verification time approaches zero when the length of the pattern grows, so that the filtration time dominates. Thus if the filtration algorithm is sublinear, the total algorithm is sublinear.

### 3.2 The Neighborhood Ordering Approach

The neighborhood ranking approach described in the previous subsection gives partial information about the relative ordering of the elements in the neighborhood of an element in  $x$ . The binary sequence used to represent each element  $x[i]$  is not enough to describe the full ordering information of a set of  $q+1$  elements.

The  $q$ -neighborhood ordering (NO) approach, which we describe in this section, associates each element of  $x$  with a binary sequence which completely

describes the ordering disposition of the elements in the  $q$ -neighborhood of  $x[i]$ . The number of comparisons we need to order a sequence of  $q + 1$  elements is between  $q$  (the best case) and  $q(q + 1)/2$  (the worst case). In this latter case it is enough to compare the element  $x[j]$ , where  $i \leq j < i + q$ , with each element  $x[h]$ , where  $j < h \leq i + q$ . Thus each element of position  $i$  in  $x$ , with  $0 \leq i < m - q$ , is associated with a binary sequence of length  $q(q + 1)/2$  which completely describes the relative order of the subsequence  $x[i, \dots, i + q]$ . Since there are  $(q + 1)!$  possible permutations of a set of  $q + 1$  elements, the string  $x$  is converted in a sequence  $\varphi_x^q$  of length  $m - q$ , where each element  $\varphi_x^q[i]$  is a value such that  $0 \leq \varphi_x^q[i] < q(q + 1)/2$ . More formally we have the following definition

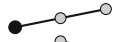
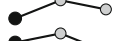
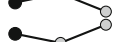
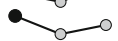
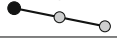

**Definition 7 ( $q$ -NO sequence).** *Given a string  $x$  of length  $m$  and an integer  $q < m$ , the  $q$ -NO sequence associated with  $x$  is a numeric sequence  $\varphi_x^q$  of length  $m - q$  over the alphabet  $\{0, \dots, q(q + 1)/2\}$  where*

$$\varphi_x^q[i] = \sum_{k=1}^q \left( \chi_x^k[i + q - k] \times 2^{(k)(k-1)/2} \right), \text{ for all } 0 \leq i < m - q \quad (3)$$

Thus the  $q$ -NO value associated to  $x[i]$  is the combination of  $q$  different NR sequences  $\chi_x^q[i], \chi_x^{q-1}[i + 1], \dots, \chi_x^1[i + q - 1]$ .

For instance the 4-NO value associated to  $x[i]$  is computed as

$$\varphi_x^4[i] = \chi_x^4[i] \times 2^6 + \chi_x^3[i + 1] \times 2^2 + \chi_x^2[i + 2] \times 2 + \chi_x^1[i + 3]$$

NEIGHBORHOOD ORDERING	Example	NO SEQ.	$\varphi_x^4[i]$
$\langle x[i], x[i + 1], x[i + 2] \rangle$		$\langle 0, 0, 0 \rangle$	0
$\langle x[i], x[i + 2], x[i + 1] \rangle$		$\langle 0, 0, 1 \rangle$	1
$\langle x[i + 2], x[i], x[i + 1] \rangle$		$\langle 0, 1, 1 \rangle$	3
$\langle x[i + 1], x[i], x[i + 2] \rangle$		$\langle 1, 0, 0 \rangle$	4
$\langle x[i + 1], x[i + 2], x[i] \rangle$		$\langle 1, 1, 0 \rangle$	6
$\langle x[i + 2], x[i + 1], x[i] \rangle$		$\langle 1, 1, 1 \rangle$	7

**Fig. 3.** The  $3!$  possible ordering of the sequence  $\langle x[i], x[i + 1], x[i + 2] \rangle$  and the corresponding binary sequence  $\langle \beta_x(i, i + 1), \beta_x(i, i + 2), \beta_x(i + 1, i + 2) \rangle$ .

*Example 3.* As in *Example 2*, let  $x = \langle 5, 6, 3, 8, 10, 7, 1, 9, 10, 8 \rangle$  be a sequence of length 10. The 3-neighborhood of the element  $x[3]$  is the subsequence  $\langle 8, 10, 7, 1 \rangle$ . The NO sequence of length 6 associated with the element of position 2 is therefore  $\langle 0, 1, 1, 1, 1, 1 \rangle$  which translates in a NO value equal to  $\varphi_x[3] = 31$ . In a similar way we can observe that the NR sequence associated with the element of position 2 is  $\langle 0, 0, 0, 0, 1, 1 \rangle$  which translates in a NO value equal to  $\varphi_x^4[2] = 3$ . The whole sequence of length 7 associated to  $x$  is  $\varphi_x^4 = \langle 20, 32, 3, 31, 60, 32, 3 \rangle$ .



COMPUTE-NR-VALUE( $x, i, q$ )	COMPUTE-NO-VALUE( $x, i, q$ )
1. $\delta \leftarrow 0$	5. $\delta \leftarrow 0$
2. for $j \leftarrow 1$ to $q$ do	6. for $k \leftarrow q$ downto 1 do
3. $\delta = (\delta \ll 1) + \beta_x(i, i + j)$	7. for $j \leftarrow 1$ to $k$ do
4. return $\delta$	8. $\delta = (\delta \ll 1) + \beta_x(i + q - k, i + q - k + j)$
	9. return $\delta$

**Fig. 4.** The two functions which compute the  $q$ -neighborhood ranking value (on the left) and the  $q$ -neighborhood ranking value (on the right).

The following Lemma 4 and Corollary 3 prove that the NO approach can be used to filter a text  $y$  in order to search for all order preserving occurrences of a pattern  $x$ . In other words they prove that  $\{i \mid x \approx y[i \dots i + m - 1]\} \subseteq \{i \mid \varphi_x^q = \varphi_y^q[i \dots i + m - k]\}$ . The proof easily follows from Definition 7 and Lemma 3.

**Lemma 4.** *Let  $x$  and  $y$  be two sequences of length  $m$  and let  $\varphi_x^q$  and  $\varphi_y^q$  the  $q$ -ranking sequences associated to  $x$  and  $y$ , respectively. If  $x \approx y$  then  $\varphi_x^q = \varphi_y^q$ .*

The following corollary proves that the NR approach can be used as a filtering. It trivially follows from Lemma 4.

**Corollary 3.** *Let  $x$  and  $y$  be two sequences of length  $m$  and  $n$ , respectively. Let  $\chi_x^q$  and  $\chi_y^q$  the  $q$ -ranking sequences associated to  $x$  and  $y$ , respectively. If  $x \approx y[j \dots j + m - 1]$  then  $\chi_x^q[i] = \chi_y^q[j + i]$ , for  $0 \leq i < m - q$ . ■*

Figure 4 (on the right) shows the procedure used for computing the NO value associated with the element of the string  $x$  at position  $i$ . The time complexity of the procedure is  $\mathcal{O}(q^2)$ . Thus, given a pattern  $x$  of length  $m$ , a text  $y$  of length  $n$  and an integer value  $q < m$ , we can solve the OPPM problem by searching  $\varphi_y^q$  for all occurrences of  $\varphi_x^q$ , using any algorithm for the exact string matching problem. During the preprocessing phase we compute the sequence  $\varphi_x^q$  and the functions  $rk_x$  and  $eq_x$ . When an occurrence of  $\varphi_x^q$  is found at position  $i$  the verification procedure ODER-ISOMORPHIC( $inv-rk, eq, y[i \dots i + m - 1]$ ) is run in order to check if  $x \approx y[i \dots i + m - 1]$ . Also in this case, if the filtration algorithm is sublinear on average, the NO approach has a sublinear behavior on average.

## 4 Experimental Evaluations

In this section we present experimental results in order to evaluate the performances of our new filter based algorithms presented in this paper. In particular we tested our filter approaches against three algorithms: the filter approach of Chhabra and Tarhio [5]; the SkSop( $k, q$ ) algorithm [3], using  $k$  hash functions and  $q$ -grams. We implemented it using  $1 \leq k \leq 5$  and  $3 \leq q \leq 8$ ; the algorithm based on SSE instructions presented in [4].

According to the experimental evaluations conducted in [5] and in [4] in our experimental evaluation we use in all cases the SBNDM2 algorithm [9]. In our

dataset we use the following names to identify the tested algorithms: FCT to identify the SBNDM2 based algorithm by Chhabra and Jorma Tarhio [5]; NRq to identify the SBNDM2 algorithm based on the NR approach (Sect. 3.1); and Noq: to identify the SBNDM2 algorithm based on the NO approach (Sect. 3.2).

We evaluated our filter based solutions in terms of efficiency, i.e. the running times. In particular for the FCT algorithm we will report the average running times, in milliseconds. Instead, for all other algorithms in the set, we will report the speed up of the running times obtained when compared with the time used by the FCT algorithm. In the case of the SkSop( $k, q$ ) algorithm we reported only the best speed-up among all different implementations, indicating in brackets the best values of  $k$  and  $q$ .

We tested our solutions on sequences of short integer values (each element is an integer in the range  $[0 \dots 256]$ ), long integer values (where each element is an integer in the range  $[0 \dots 10.000]$ ) and floating point values (each element is a floating point in the range  $[0.0 \dots 10000.99]$ ). However we don't observe sensible differences in the results, thus in the following tables we report for brevity the results obtained on short integer sequences. All texts have 1 million of elements. In particular we tested our algorithm on two sequences: a RAND- $\delta$  sequence of random integer values varying around a fixed mean equal to 100 with a variability of  $\delta$ ; a PERIOD- $\delta$  sequence of random integer values varying around a periodic function with a period of 10 elements with a variability of  $\delta$ .

**Table 1.** Experimental results on a RAND- $\delta$  short integer sequence.

$\delta$	$m$	FCT	SkSop	SSE	NR2	NR3	NR4	NR5	NR6	No2	No3	No4
5	8	44.29	1.27 <sup>(5,4)</sup>	1.45	1.16	1.28	1.25	1.25	1.24	1.89	1.71	1.11
	12	28.39	1.37 <sup>(4,5)</sup>	1.38	1.16	1.37	1.37	1.33	1.19	1.64	<u>2.00</u>	1.64
	16	20.65	1.52 <sup>(4,5)</sup>	1.23	1.15	1.30	1.43	1.34	1.14	1.42	<u>2.01</u>	1.83
	20	16.29	1.58 <sup>(4,5)</sup>	1.12	1.15	1.30	1.45	1.41	1.14	1.39	<u>2.00</u>	1.93
	24	13.64	1.63 <sup>(5,4)</sup>	1.05	1.16	1.29	1.42	1.44	1.12	1.34	1.91	<u>2.01</u>
	28	11.48	1.62 <sup>(5,4)</sup>	0.99	1.16	1.28	1.44	1.45	1.11	1.31	1.88	<u>1.96</u>
	32	10.34	1.60 <sup>(5,4)</sup>	0.83	1.18	1.30	1.40	1.46	1.12	1.30	1.83	<u>2.05</u>
20	8	42.34	1.22 <sup>(3,4)</sup>	1.68	1.13	1.27	1.25	1.26	1.22	<u>1.92</u>	1.68	1.08
	12	27.93	1.40 <sup>(4,5)</sup>	1.44	1.17	1.40	1.37	1.32	1.21	1.71	<u>2.04</u>	1.63
	16	20.05	1.46 <sup>(4,5)</sup>	1.32	1.15	1.32	1.41	1.33	1.15	1.48	<u>2.04</u>	1.81
	20	15.85	1.51 <sup>(4,5)</sup>	1.21	1.15	1.29	1.42	1.37	1.11	1.38	<u>2.00</u>	1.90
	24	13.31	1.55 <sup>(5,6)</sup>	1.12	1.17	1.31	1.47	1.42	1.12	1.36	1.99	<u>2.02</u>
	28	11.38	1.58 <sup>(5,6)</sup>	1.01	1.17	1.31	1.42	1.45	1.09	1.35	1.94	<u>2.07</u>
	32	9.96	1.58 <sup>(3,7)</sup>	0.97	1.16	1.29	1.45	1.46	1.09	1.29	1.87	<u>2.09</u>
40	8	42.62	1.19 <sup>(3,4)</sup>	1.91	1.16	1.28	1.28	1.25	1.25	<u>1.94</u>	1.70	1.09
	12	28.35	1.39 <sup>(4,5)</sup>	1.82	1.19	1.41	1.39	1.36	1.21	1.75	<u>2.06</u>	1.65
	16	20.37	1.43 <sup>(4,5)</sup>	1.63	1.18	1.32	1.44	1.37	1.17	1.49	<u>2.09</u>	1.83
	20	16.12	1.52 <sup>(5,6)</sup>	1.45	1.15	1.29	1.46	1.39	1.12	1.39	<u>2.04</u>	1.95
	24	13.35	1.57 <sup>(5,6)</sup>	1.21	1.18	1.30	1.46	1.44	1.13	1.36	1.97	<u>1.99</u>
	28	11.60	1.57 <sup>(5,6)</sup>	1.17	1.18	1.32	1.47	1.50	1.14	1.37	1.96	<u>2.06</u>
	32	10.06	1.58 <sup>(5,7)</sup>	0.99	1.16	1.29	1.45	1.48	1.10	1.33	1.89	<u>2.07</u>

For each text in the set we randomly select 100 patterns extracted from the text and compute the average running time over the 100 runs. We also computed

the average number of false positives detected by the algorithms during the search. All the algorithms have been implemented using the C programming language and have been compiled on an MacBook Pro using the gcc compiler Apple LLVM version 5.1 (based on LLVM 3.4svn) with 8 Gb Ram. During the compilation we use the -O3 optimization option. In the following table running times are expressed in milliseconds. Best results have been underlined.

Experimental results on RAND- $\delta$  numeric sequences have been conducted with values of  $\delta = 5, 20,$  and  $40$  (see Table 1). The results show as the NO approach is the best choice in all cases, achieving a speed up of 2.0 if compared with the FCT algorithm. Also the NR approach achieves always a good speed up which is between 1.15 and 1.50. In addition we can observe that in all cases the best speed-up achieved by the new algorithms are greater than that achieved by the SkSop and SSE algorithms. For the sake of completeness we report also that the gain in number of detected false positives in most cases between 90% and 100%.

**Table 2.** Experimental results on a PERIOD- $\delta$  short integer sequence.

$\delta$	$m$	FCT	SkSop	SSE	NR2	NR3	NR4	NR5	NR6	No2	No3	No4
5	8	41.08	0.83 <sup>(3,4)</sup>	1.10	<u>0.99</u>	<u>1.05</u>	0.88	0.79	0.90	0.88	0.73	0.60
	12	36.42	0.88 <sup>(4,6)</sup>	1.03	<u>1.06</u>	1.02	0.94	0.86	0.91	0.81	0.67	0.69
	16	34.03	0.90 <sup>(4,6)</sup>	0.97	<u>1.04</u>	0.86	0.78	0.74	1.00	0.77	0.64	0.60
	20	35.31	0.97 <sup>(4,7)</sup>	0.90	<u>0.98</u>	0.89	0.88	0.84	0.92	0.73	0.60	0.55
	24	37.90	1.05 <sup>(4,7)</sup>	0.82	<u>1.34</u>	1.33	1.30	1.18	1.15	0.99	0.82	0.76
	28	36.26	1.11 <sup>(4,7)</sup>	0.75	<u>1.17</u>	1.09	1.10	1.04	0.97	0.78	0.64	0.56
	32	35.38	1.16 <sup>(4,8)</sup>	0.69	<u>1.10</u>	<u>1.15</u>	1.05	0.95	0.94	0.82	0.65	0.59
20	8	42.35	0.93 <sup>(3,4)</sup>	1.12	0.98	<u>1.18</u>	0.91	0.81	0.89	1.02	0.83	0.68
	12	39.09	0.97 <sup>(4,5)</sup>	1.05	1.11	<u>1.14</u>	1.06	0.98	1.00	1.02	0.88	0.93
	16	34.25	1.05 <sup>(4,6)</sup>	1.01	<u>1.11</u>	1.01	1.02	1.01	1.08	0.96	0.87	0.87
	20	35.41	1.11 <sup>(4,6)</sup>	0.96	1.10	1.09	1.21	<u>1.21</u>	1.07	0.97	0.89	0.89
	24	35.15	1.18 <sup>(3,7)</sup>	0.89	1.31	1.51	<u>1.67</u>	1.60	1.14	1.15	1.10	1.18
	28	32.23	1.24 <sup>(3,7)</sup>	0.81	1.23	1.40	<u>1.56</u>	1.36	1.07	1.04	1.08	1.15
	32	30.34	1.36 <sup>(3,7)</sup>	0.75	1.43	<u>1.60</u>	1.53	1.43	1.22	1.19	1.11	1.07
40	8	45.07	1.10 <sup>(3,4)</sup>	1.15	0.93	<u>1.18</u>	0.94	0.81	0.89	1.12	0.91	0.78
	12	37.91	1.14 <sup>(4,5)</sup>	1.08	1.08	1.12	1.03	0.93	1.03	<u>1.13</u>	1.03	1.08
	16	32.41	1.19 <sup>(4,5)</sup>	1.04	1.11	1.04	1.06	<u>1.13</u>	1.07	1.07	1.02	1.10
	20	28.63	1.22 <sup>(4,6)</sup>	1.00	1.05	1.09	1.24	<u>1.35</u>	1.08	1.04	1.04	1.15
	24	27.25	1.28 <sup>(4,6)</sup>	0.97	1.18	1.39	<u>1.59</u>	1.53	1.10	1.12	1.14	1.40
	28	24.91	1.32 <sup>(4,6)</sup>	0.95	1.20	1.51	<u>1.67</u>	1.41	1.05	1.17	1.30	1.50
	32	23.63	1.38 <sup>(4,6)</sup>	0.91	1.39	<u>1.63</u>	1.55	1.31	1.20	1.27	1.41	1.41

Experimental results on PERIOD- $\delta$  problem have been conducted on a periodic sequence with a period equal to 10 and with  $\delta = 5, 20$  and  $40$  (see Table 2). The results show as the NR approach is the best choice in most of the cases, achieving a speed up of 1.3 in suitable conditions. However in some cases the FCT algorithm turns out to be the best choice especially on short patterns. The NO approach is always less efficient of the FCT algorithm. When the size of  $\delta$  increases the performances of the NO approach get better achieving a speed up

of 1.4 in the best cases. However the NR approach turns out to be always the best solutions with a speed up close to 1.7 for long patterns. Also in these cases the best speed-up achieved by the new algorithms are greater than that achieved by the SkSop and SSE algorithms.

## References

1. Belazzougui, D., Pierrot, A., Raffinot, M., Vialette, S.: Single and multiple consecutive permutation motif search. In: Cai, L., Cheng, S.-W., Lam, T.-W. (eds.) *Algorithms and Computation*. LNCS, vol. 8283, pp. 66–77. Springer, Heidelberg (2013)
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: Holub and Zdárek [10], pp. 22–35
4. Chhabra, T., Külekci, M.O., Tarhio, J.: Alternative algorithms for order-preserving matching. In: Holub and Zdárek [10], pp. 36–46
5. Chhabra, T., Tarhio, J.: Order-preserving matching with filtration. In: Gudmundsson, J., Katajainen, J. (eds.) *SEA 2014*. LNCS, vol. 8504, pp. 307–314. Springer, Heidelberg (2014)
6. Chhabra, T., Tarhio, J.: A filtration method for order-preserving matching. *Inf. Process. Lett.* **116**(2), 71–74 (2016)
7. Cho, S., Na, J.C., Park, K., Sim, J.S.: Fast order-preserving pattern matching. In: Widmayer, P., Xu, Y., Zhu, B. (eds.) *COCOA 2013*. LNCS, vol. 8287, pp. 295–305. Springer, Heidelberg (2013)
8. Faro, S., Külekci, M.O.: Efficient algorithms for the order preserving pattern matching problem. *CoRR* abs/1501.04001 (2015). <http://arxiv.org/abs/1501.04001>
9. Holub, J., Durian, B.: Talk: fast variants of bit parallel approach to suffix automata. In: *International Stringology Research Workshop* (2005). <http://www.cri.haifa.ac.il/events/2005/string/presentations/Holub.pdf>
10. Kim, J., Eades, P., Fleischer, R., Hong, S., Iliopoulos, C.S., Park, K., Puglisi, S.J., Tokuyama, T.: Order-preserving matching. *Theor. Comput. Sci.* **525**, 68–79 (2014)
11. Knuth, D.E., Morris Jr., J.H., Pratt, V.R.: Fast pattern matching in strings. *SIAM J. Comput.* **6**(1), 323–350 (1977)
12. Kubica, M., Kulczynski, T., Radoszewski, J., Rytter, W., Walen, T.: A linear time algorithm for consecutive permutation pattern matching. *Inf. Process. Lett.* **113**(12), 430–433 (2013)