arXiv:1501.04001v1 [cs.DS] 16 Jan 2015

Efficient Algorithms for the Order Preserving Pattern Matching Problem*

Simone Faro[†] and M. Oğuzhan Külekci[‡]

[†]Università di Catania, Department of Mathematics and Computer Science, Italy
[‡]Istanbul Medipol University, Department of Biomedical Engineering, Turkey
faro@dmi.unict.it, okulekci@medipol.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. It is an approximate variant of the well known *exact pattern matching* problem which has gained attention in recent years. This interesting problem finds applications in a lot of fields as time series analysis, like share prices on stock markets, weather data analysis or 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. 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%.

1 Introduction

Given a pattern x of length m and a text y of length n, both over a common alphabet Σ, the *exact string matching problem* consists in finding all occurrences of the string x in y. String matching is a very important subject in the wider domain of text processing and algorithms for the problem are also basic components used in the implementations of practical softwares existing under most operating systems. Moreover, they emphasize programming methods that serve as paradigms in other fields of computer science. Finally they also play an important role in theoretical computer science by providing challenging problems. The worst case lower bound of the string matching problem is $\mathcal{O}(n)$ and was achieved the first time by the well known algorithm by Knuth, Morris and Pratt [8]. However many string matching algorithms have been also developed to obtain sublinear $\mathcal{O}(n \log m/m)$ performance on average. Among them the Boyer-Moore algorithm [1] deserves a special mention, since it has been particularly successful and has inspired much work.

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

* This work has been supported by the Scientific & Technological Research Council Of Turkey (TUBITAK), the Department Of Science Fellowships & Grant Programs (BIDEB), 2221 Fellowship Program, and by G.N.C.S., Istituto Nazionale di Alta Matematica "Francesco Severi".

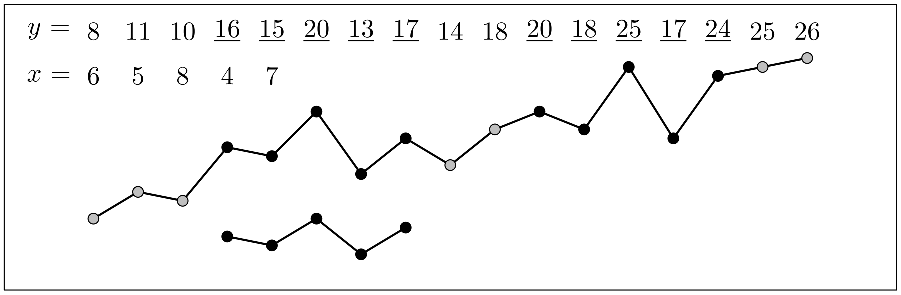


Fig. 1. Example of a pattern x of length 5 over an integer alphabet with two order preserving occurrences in a text y of length 17, at positions 3 and 10.

gained attention in recent years. In this variant the characters of x and y are drawn from an ordered alphabet Σ 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).

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. [7] in 2013. They proposed a $\mathcal{O}(n + m \log m)$ solution over generic ordered alphabets based on the Knuth-Morris-Pratt algorithm [8] and a $\mathcal{O}(n+m)$ solution in the case of integer alphabets. Some months later Kim et al. [6] 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 [1] to order-preserving matching problem, in 2013 Cho et al. [3] 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. More recently Chhabra and Tarhio [2] presented a more practical solution based on approximate string matching. Their technique is based on a conversion of the input sequences in binary sequences and on the application of any standard algorithm for exact string matching as a filtration method.

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 Section 2 we give preliminary notions and definitions relative to the order-preserving pattern matching problem while in Section 3 we briefly describe the previous solutions to the problem. Then we present our new solutions in Section 4 and evaluate their performances against the previous algorithms in Section 5. Conclusions are drawn in Section 6.

2 Notions and Basic Definitions

A string x over an ordered alphabet Σ, of size σ, is defined as a sequence of elements in Σ. We suppose that a total order relation "$\leq$" is defined on it, so that we could establish if $a \leq b$ for each $a, b \in \Sigma$.

We indicate the length of a string x with the symbol $|x|$. We refer to the elements in x with the symbol $x[i]$, for $0 \leq i < |x|$. Moreover we indicate with $x[i \dots j]$ the subsequence of x from the element of position i to the element of position j (including the extremes), for $0 \leq i \leq j < |x|$.

We say that two sequences $x, y \in \Sigma^*$ are order isomorphic if the relative order of their elements is the same. More formally we give the following definition.

Definition 1 (order isomorphism). *Given an ordered alphabet Σ and two sequences $x, y \in \Sigma^*$ of the same length, we say that x and y are order-isomorphic, and write $x \approx y$, if the following conditions hold*

1. $|x| = |y|$
2. $x[i] \leq x[j]$ *if and only if* $y[i] \leq y[j]$, *for* $0 \leq i, j < |x|$

Definition 2 (rank function). *Let x be a sequence of length m over an ordered alphabet Σ. The rank function of x if a mapping $r : \{0, 1, \dots, m-1\} \rightarrow \{0, 1, \dots, m-1\}$ such that $x[r(i)] \leq x[r(j)]$ holds for each pair $0 \leq i < j < m$. Formally we define*

$$r(i) = |\{j \ : \ x[j] < x[i] \ or \ (x[j] = x[i] \ and \ j < i)\}|$$

for $0 \leq i < m$.

We will refer to the value $r(i)$ as the *rank* of $x[i]$ in x, while we will refer to the sequence $\langle r(0), r(1), \dots r(m-1)\rangle$ as the *relative order* of x.

According to Definition 2 we have that $x[r(0)]$ is the smallest number while $x[r(m-1)]$ is the greater number in x. If we assume that $sort(x)$ is the time required to sort all the elements of x, then it is easy to observe that the relative order of x can be computed in $\mathcal{O}(sort(x))$ time.

In addition, we define the *equality function* of x which indicates which elements of the sequence are equal (if any). More formally we have the following definition.

```
NODER-ISOMORPHISM$(r, eq, y, i)$
1.  for $i \leftarrow 0$ to $|x| - 1$ do
2.      if $(y[r(i)] > y[r(j + i + 1)])$ then return false
3.      if $(y[r(i)] < y[r(j + i + 1)]$ and $eq(i) = 1)$ then return false
4.      if $(y[r(i)] = y[r(j + i + 1)]$ and $eq(i) = 0)$ then return false
5.  return true
```

Fig. 2. The function used to verify if two sequences x and $y[i \ldots i + |x| - 1]$ are order isomorphic. We assume that the function receives as input the parameter r and eq which represent the rank function and the equality function of x, respectively.

Definition 3 (equality function). *Let x be a sequence of length m over an ordered alphabet Σ and let r be the rank function of x. The equality function of x if a mapping $eq : \{0, 1, \ldots, m-2\} \rightarrow \{0, 1\}$ such that, for each $0 \leq i < m$*

$$eq(i) = \begin{cases} 1 & \textit{if} \quad x[r(i)] = x[r(i+1)] \\ 0 & \textit{otherwise} \end{cases}$$

Let r be the rank function of a string x, such that $m = |x|$, and let q be its equality function. It is easy to prove that x and y are order isomorphic if and only if they share the same rank and equality function, i.e. if and only if the following two conditions hold

1. $y[r(i)] \leq y[r(i+1)]$, for $0 \leq i < m - 1$
2. $y[r(i)] = y[r(i+1)]$ if and only if $q(i) = 1$, for $0 \leq i < m - 1$

Example 1. Let $x = \langle 6, 3, 8, 3, 10, 7, 10 \rangle$ and $y = \langle 2, 1, 4, 1, 5, 3, 5 \rangle$ two sequences of size 7. We have that the relative order of x is $(1, 3, 0, 5, 2, 4, 6)$ while its equality function is $eq(x[i]) = (1, 0, 0, 0, 0, 1)$. The two string are order isomorphic according to the definition given above, i.e. $x \approx y$.

The procedure to verify that two numeric sequences, x and y, are order isomorphic is shown in Fig.2. It receives as input the functions r and q, computed on x and returns a boolean value indicating if $x \approx y$. The algorithm requires $\mathcal{O}(m)$ time, where m is the length of the sequences. A mismatch occurs when one of the three conditions of lines 2, 3 and 4, holds.

The OPPM problem consists in finding all substring of the text with the same relative order as the pattern. Specifically we have the following formal definition.

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

If an occurrence of the pattern x starts at portion i of the text y, we say that x has an order-preserving occurrence at position i.

3 Previous Results

The OPPM problem has drawn particular attention in the last few years, during which some efficient results have been proposed.

The first algorithm to solve the OPPM problem was presented by Kubica *et al.* in [7]. Their solution was an adaptation of the well Known Knuth-Morris-Pratt algorithm for the exact string matching problem, where the fail function is adapted to compute the order-borders table. The authors proved that this table can be computed in linear time in the length of the pattern x, if the relative order of x is known in advance. The overall time complexity of the algorithm is $\mathcal{O}(n + m \log m)$, where m is the length of the pattern while n is the length of the text. However in [3] Cho *et al.* proved that the algorithm presented in [7] can decide incorrectly when there are equal values in the string.

The second algorithm based on Knuth-Morris-Pratt was presented later by Kim et al. [6]. Their algorithm is based on the prefix representation and it is further optimized according to the nearest neighbor representation. The prefix representation is based on finding the rank of each integer in the prefix. It can be computed easily by inserting each character to the dynamic order statistic tree and then computing the rank of each character in the prefix. The time complexity of computing such prefix representation is $O(m \log m)$. The failure function is then computed as in the Knuth-Morris-Pratt algorithm in $O(m \log m)$ time. The overall time complexity of this algorithm is $O(n+m \log m)$. Again, this solution does not work properly when there are equal values in the pattern.

The first sublinear solution for the OPPM problem was presented by Cho *et al.* in [3]. Their algorithm is an adaptation to OPPM of the well known Boyer-Moore approach. They apply a q-grams technique, i.e. groups of q consecutive characters are treated as a single condensed character, in order to make the shifts longer. In this way, a large amount of text can be skipped for long patterns.

More recently Chhabra and Tarhio presented a new practical solution [2] based on a filtration technique. 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 β 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 overall complexity of the algorithm is sublinear on average. Experimental results conducted in [2] show that the filter approach was considerably faster than the algorithm by Cho *et al.*

For the sake of completeness we notice that Crochemore *et al.* presented in [4] a solution for the offline version of the OPPM problem based on a new data structure called order-preserving suffix tree. Their solution finds all occurrences of x in y in $\mathcal{O}((m \log n)/\log\log m + z)$ where z is the number of occurrences of x in y. In this paper we concentrate on the online version of the OPPM problem.

4 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 [2], but we use information extracted from groups of integers in the input string, as in [3], in order to make the filtration phase more effective in terms of efficiency and accuracy, as discussed below.

Text filtration is a largely used technique in the field of exact and approximate string matching. Specifically, instead of checking at each position of the text if the pattern occurs, it seems to be more efficient to filter text positions and check only when a substring looks like the pattern. When a resemblance has been detected a naive check of the occurrence is performed. In literature filtration techniques are generally improved by using q-grams, i.e. groups of adjacent characters of the string which are considered as a single character of a condensed alphabet.

It is always convenient to use a filtration method which better and faster localize candidate occurrences, which imply accuracy and efficiency of the method, respectively.

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 q-grams and with the time efficiency of the overall searching algorithm. It is clear that these two values are strongly related since a low accuracy implies an high number of false positives and, as a consequence, a decrease in the performance of the 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.

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 $\langle x[i], x[i+1], \ldots, x[i+q]\rangle$.*

Both the filtration methods presented below translate the input sequence in a target numeric sequence which is used for the filtration. Specifically each position i of the sequence is associated with a numeric value computed from the structure of the q-neighborhood of the element $x[i]$.

4.1 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 Equation (1). Formally we have

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

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

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 $j = 1, \ldots, q$. In other words we compute the binary sequence $\langle \beta_x(i, i{+}1), \beta_x(i, i{+}2), \ldots, \beta_x(i, i{+}q)\rangle$ 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.

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

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 χ_x^q of length $m-q$ over the alphabet $\{0, \ldots, 2^q\}$ where*

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

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 $\chi_x^4 = \langle 4, 8, 1, 6, 15, 8\rangle$.

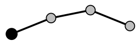
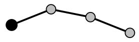
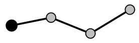
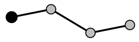
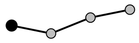
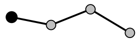
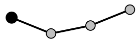
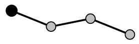
Neighborhood Ranking	Example	NR seq.	$\chi_x^3[i]$
$x[i] \leq x[i+1], x[i+2], x[i+3]$		$\langle 0,0,0\rangle$	0
$x[i+3] \leq x[i] \leq x[i+1], x[i+2]$		$\langle 0,0,1\rangle$	1
$x[i+2] \leq x[i] \leq x[i+1], x[i+3]$		$\langle 0,1,0\rangle$	2
$x[i+2], x[i+3] \leq x[i] \leq x[i+1]$		$\langle 0,1,1\rangle$	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. 3. 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 $\langle x[i], x[i+1], x[i+2], x[i+3]\rangle$.

The following Lemma 1 and Corollary 1 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. In other words it proves that

$$\{i \mid x \approx y[i \dots i+m-1]\} \subseteq \{i \mid \chi_x^q = \chi_y^q[i \dots i+m-k]\}.$$

Lemma 1. *Let x and y be two sequences of length m and let χ_x^q and χ_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 r 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 $x[r(i)] \leq x[r(i+1)]$, for $0 \leq i < m-1$;
2. by hyphotesis and Def.1, $y[r(i)] \leq y[r(i+1)]$, 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 *5* and Definition 6, we have $\chi_x^q[i] = \chi_y^q[i]$, for $0 \leq i < m-q$.

This last statement proves the thesis. ■

The following corollary prices that the NR approach can be used as a filtering. It trivially follows from Lemma 1.

```
Compute-NR-Value($x, i, q$)
    $\delta \leftarrow 0$
    for $j \leftarrow 1$ to $q$ do
        $\delta = (\delta \ll 1) + \beta_x(i, i+j)$
    return $\delta$
```

Fig. 4. The function which computes the q-neighborhood ranking value of the element of position i in a sequence x. The value id computed in $\mathcal{O}(q)$ time.

Corollary 1. *Let x and y be two sequences of length m and n, respectively. Let χ_x^q and χ_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 $0 \leq i < m-q$.* ■

Fig. 4 shows the procedure used 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 χ_y^q for all occurrences of χ_x^q, using any algorithm for the exact string matching problem. During the preprocessing phase we compute the sequence χ_x^q and the functions r_x and q_x. When an occurrence of χ_x^q is found at position i the verification procedure Noder-Isomorphism(r, q, y, i) (shown in Fig.2) 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 [2], 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.

4.2 The Neighborhood Ordering Approach

The neighborhood ranking approach described in the previous section gives partial information about the relative ordering of the elements in the q-neighborhood of an element in x. The q 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 the 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

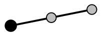
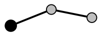
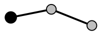
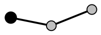
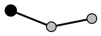
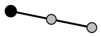
Neighborhood Ordering	Example	NO seq.	$\varphi^4_x[i]$
$\langle x[i], x[i+1], x[i+2]\rangle$		$\langle 0,0,0\rangle$	0
$\langle x[i], x[i+2], x[i+1]\rangle$		$\langle 0,0,1\rangle$	1
$\langle x[i+2], x[i], x[i+1]\rangle$		$\langle 0,1,1\rangle$	3
$\langle x[i+1], x[i], x[i+2]\rangle$		$\langle 1,0,0\rangle$	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. 5. The 3! possible ordering of the sequence $\langle x[i], x[i+1], x[i+2]\rangle$ and the corresponding binary sequence $\langle \beta_x(i, i+1), \beta_x(i, i+2), \beta_x(i+1, i+2)\rangle$.

```
Compute-NO-Value($x, i, q$)
$\delta \leftarrow 0$
for $k \leftarrow q$ downto 1 do
    for $j \leftarrow 1$ to $k$ do
        $\delta = (\delta \ll 1) + \beta_x(i+q-k, i+q-k+j)$
return $\delta$
```

Fig. 6. The function which computes the q-neighborhood ranking value of the element of position i in a sequence x. The value is computed in $\mathcal{O}(q^2)$ time.

of the susequence $x[i, \dots, i+q]$. Since there are $(q+1)!$ possible permutations of a set of $q+1$ elements, the string x is converted in a sequence φ^q_x of length $m-q$, where each element $\varphi^q_x[i]$ is a value such that $0 \leq \varphi^q_x[i] < q(q+1)/2$.

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 φ^q_x of length $m-q$ over the alphabet $\{0, \dots, q(q+1)/2\}$ where*

$$\varphi^q_x[i] = \sum_{k=1}^{q} \left(\chi^k_x[i+q-k] \times 2^{(k)(k-1)/2} \right), \textit{ for all } 0 \leq i < m-q \qquad (3)$$

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

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

$$\varphi^4_x[i] = \chi^4_x[i] \times 2^6 + \chi^3_x[i+1] \times 2^2 + \chi^2_x[i+2] \times 2 + \chi^1_x[i+3]$$

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 $\varphi_x^4 = \langle 20, 32, 3, 31, 60, 32, 3\rangle$.

The following Lemma 2 and Corollary 2 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

$$\{i \mid x \approx y[i \dots i + m - 1]\} \subseteq \{i \mid \varphi_x^q = \varphi_y^q[i \dots i + m - k]\}.$$

Lemma 2. *Let x and y be two sequences of length m and let φ_x^q and φ_y^q the q-ranking sequences associated to x and y, respectively. If $x \approx y$ then $\varphi_x^q = \varphi_y^q$.*

Proof. The theorem easily follows from Definition 7 and Lemma 1. ■

The following corollary proves that the NR approach can be used as a filtering. It trivially follows from Lemma 2.

Corollary 2. *Let x and y be two sequences of length m and n, respectively. Let χ_x^q and χ_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 $0 \leq i < m - q$.* ■

Fig. 6 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 φ_y^q for all occurrences of φ_x^q, using any algorithm for the exact string matching problem. During the preprocessing phase we compute the sequence φ_x^q and the functions r_x and q_x. When an occurrence of φ_x^q is found at position i the verification procedure NODER-ISOMORPHISM(r, q, y, i) (shown in Fig.2) 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.

5 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 the filter approach of Chhabra and Tarhio [2], which is, to the best of our knowledge, the most effective solution in practical cases. In the experimental evaluation conducted in [2] the SBNDM2 and SBNDM4 algorithms [5] turned out to be the most effective exact string matching algorithms which can be used in combination with the filter technique. Following the same line, in our experimental evaluation we use in all cases the SBNDM2 algorithm. However any other exact string matching algorithm could be used for this purpose. In our dataset we use the following names to identify the tested algorithms

- FCT: the SBNDM2 algorithm based on the filter approach by Chhabra and Jorma Tarhio presented in [2];
- NRq: the SBNDM2 algorithm based on the Neighborhood Ranking approach presented in Section 4.1
- NOq: the SBNDM2 algorithm based on the Neighborhood Ordering approach presented in Section 4.2

We do not compare our solution with the Boyer-Moor approach by Cho *et al.* [3] since it was shown to be less efficient than the algorithm by Chhabra and Tarhio in all cases. We evaluated our filter based solutions in terms of efficiency, i.e. the running times, and accuracy, i.e. the percentage of false positives detected during the filtration phase. In particular for the FCT algorithm we will report the average running times, in milliseconds, and the average number of false positives detected every 2^{20} text characters. Instead, for all other algorithms in the set, we will report the following two values

- the speed up of the running times obtained when compared with the time used by the FCT algorithm. If $\mathit{time}(\text{FCT})$ is the running time of the FCT algorithm and t is the running time of our algorithm, then the speed up is computed as $\mathit{time}(\text{FCT})/t$.
- the percentage of the gain in the number of false positives detected by the algorithm when compared with the FCT algorithm. If $\mathit{fp}(\text{FCT})$ is the number of false positives detected on average by the FCT algorithm and fp is number of false positives detected by our filter approach, then the gain is computed as $(100 \times (\mathit{fp}(\text{FCT}) - fp)/\mathit{fp}(\text{FCT})$.

We tested our solutions on sequences of short integer values (each element is an integer in the range $[0 \ldots 256]$), long integer values (where each element is an integer in the range $[0 \ldots 10.000]$) and floating point values (each element is a floating point in the range $[0.0 \ldots 10000.99]$). However we don't observe sensible differences in the results, thus in the following table 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 the following set of short integer sequences.

- RAND-δ: a sequence of random integer values ringing around a fixed mean equal to 100. Each value of the sequence is randomly chosen around the mean with a variability of δ, so that the text can be seen as a random sequence of integers between $100 - \delta$ and $100 + \delta$ with a uniform distribution.
- PERIOD-δ: a sequence of random integer values ringing around a periodic function with a period of 10 elements. Each value of the sequence is randomly chosen around the function with a variability of δ. All values of the sequences are always in the range $\{0 \ldots 200 + \delta\}$.

m	FCT	NR2	NR3	NR4	NR5	NR6	NO2	NO3	NO4
8	44.29	1.16	1.28	1.25	1.25	1.24	1.89	1.71	1.11
12	28.39	1.16	1.37	1.37	1.33	1.19	1.64	2.00	1.64
16	20.65	1.15	1.30	1.43	1.34	1.14	1.42	2.01	1.83
20	16.29	1.15	1.30	1.45	1.41	1.14	1.39	2.00	1.93
24	13.64	1.16	1.29	1.42	1.44	1.12	1.34	1.91	2.01
28	11.48	1.16	1.28	1.44	1.45	1.11	1.31	1.88	1.96
32	10.34	1.18	1.30	1.40	1.46	1.12	1.30	1.83	2.05
8	15713.46	84.1	92.4	95.1	94.0	90.2	97.5	99.1	99.6
12	1420.78	95.8	99.3	99.7	99.8	97.5	99.8	100.0	100.0
16	123.22	99.4	100.0	100.0	100.0	99.7	100.0	100.0	100.0
20	12.07	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
24	1.01	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
28	0.02	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
32	0.00	-	-	-	-	-	-	-	-

Table 1. Experimental results on a RAND-5 short integer sequence.

m	FCT	NR2	NR3	NR4	NR5	NR6	NO2	NO3	NO4
8	42.34	1.13	1.27	1.25	1.26	1.22	1.92	1.68	1.08
12	27.93	1.17	1.40	1.37	1.32	1.21	1.71	2.04	1.63
16	20.05	1.15	1.32	1.41	1.33	1.15	1.48	2.04	1.81
20	15.85	1.15	1.29	1.42	1.37	1.11	1.38	2.00	1.90
24	13.31	1.17	1.31	1.47	1.42	1.12	1.36	1.99	2.02
28	11.38	1.17	1.31	1.42	1.45	1.09	1.35	1.94	2.07
32	9.96	1.16	1.29	1.45	1.46	1.09	1.29	1.87	2.09
8	14326.78	83.6	92.3	95.6	92.9	90.2	97.7	99.3	99.7
12	1295.88	96.4	99.5	99.9	99.9	97.8	99.9	100.0	100.0
16	118.79	99.3	100.0	100.0	100.0	99.7	100.0	100.0	100.0
20	10.43	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
24	0.71	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
28	0.00	-	-	-	-	-	-	-	-
32	0.00	-	-	-	-	-	-	-	-

Table 2. Experimental results on a RAND-20 short integer sequence.

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 8Gb 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 Random Sequences

Experimental results on RAND-δ numeric sequences have been conducted with values of $\delta = 5, 20, 40$ (see Table 1, Table 2 and Table 3). The results show

m	Fct	Nr2	Nr3	Nr4	Nr5	Nr6	No2	No3	No4
8	42.62	1.16	1.28	1.28	1.25	1.25	1.94	1.70	1.09
12	28.35	1.19	1.41	1.39	1.36	1.21	1.75	2.06	1.65
16	20.37	1.18	1.32	1.44	1.37	1.17	1.49	2.09	1.83
20	16.12	1.15	1.29	1.46	1.39	1.12	1.39	2.04	1.95
24	13.35	1.18	1.30	1.46	1.44	1.13	1.36	1.97	1.99
28	11.60	1.18	1.32	1.47	1.50	1.14	1.37	1.96	2.06
32	10.06	1.16	1.29	1.45	1.48	1.10	1.33	1.89	2.07
8	15413.57	86.6	93.7	95.9	94.4	91.9	98.1	99.4	99.8
12	1492.39	97.0	99.6	99.9	99.9	98.1	99.9	100.0	100.0
16	114.82	99.3	100.0	100.0	100.0	99.7	100.0	100.0	100.0
20	9.83	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
24	0.83	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
28	0.00	-	-	-	-	-	-	-	-
32	0.00	-	-	-	-	-	-	-	-

Table 3. Experimental results on a Rand-40 short integer sequence.

m	Fct	Nr2	Nr3	Nr4	Nr5	Nr6	No2	No3	No4
8	41.08	0.99	1.05	0.88	0.79	0.90	0.88	0.73	0.60
12	36.42	1.06	1.02	0.94	0.86	0.91	0.81	0.67	0.69
16	34.03	1.04	0.86	0.78	0.74	1.00	0.77	0.64	0.60
20	35.31	0.98	0.89	0.88	0.84	0.92	0.73	0.60	0.55
24	37.90	1.34	1.33	1.30	1.18	1.15	0.99	0.82	0.76
28	36.26	1.17	1.09	1.10	1.04	0.97	0.78	0.64	0.56
32	35.38	1.10	1.15	1.05	0.95	0.94	0.82	0.65	0.59
8	48697.90	78.1	78.2	75.0	61.8	83.0	89.1	94.2	95.8
12	45427.73	66.4	72.8	74.6	71.4	67.7	76.3	82.4	84.9
16	32091.18	54.1	63.6	66.0	63.9	55.3	66.3	72.7	74.5
20	26337.31	41.0	49.0	52.6	53.6	43.0	53.4	59.1	61.5
24	23100.22	42.3	56.9	61.6	62.5	44.0	60.5	66.7	69.6
28	23296.19	53.2	63.0	70.7	73.1	55.1	65.8	73.8	76.7
32	17959.33	49.7	66.6	72.0	75.6	50.5	68.9	75.2	79.4

Table 4. Experimental results on a Period-5 short integer sequence.

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. The gain in number of detected false positives is impressive and is in most cases between 90% and 100%. It is interesting to observe also that the value of δ do not affect the running times and the number of false positives detected during the search, which are very similar in the three tables.

Experimental Results on Periodic Sequences

Experimental results on Period-δ problem have been conducted on a periodic sequence with a period equal to 10 and with $\delta = 5$ (see Table 4). The results show as the Nr1 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

m	Fct	Nr2	Nr3	Nr4	Nr5	Nr6	No2	No3	No4
8	42.35	0.98	1.18	0.91	0.81	0.89	1.02	0.83	0.68
12	39.09	1.11	1.14	1.06	0.98	1.00	1.02	0.88	0.93
16	34.25	1.11	1.01	1.02	1.01	1.08	0.96	0.87	0.87
20	35.41	1.10	1.09	1.21	1.21	1.07	0.97	0.89	0.89
24	35.15	1.31	1.51	1.67	1.60	1.14	1.15	1.10	1.18
28	32.23	1.23	1.40	1.56	1.36	1.07	1.04	1.08	1.15
32	30.34	1.43	1.60	1.53	1.43	1.22	1.19	1.11	1.07
8	62122.44	56.9	77.8	71.5	57.1	60.9	84.7	91.8	95.9
12	50264.79	56.5	72.8	77.3	76.7	58.6	77.0	85.0	88.1
16	32026.85	60.0	73.8	79.4	80.5	62.4	78.8	86.3	89.2
20	23138.04	61.1	77.4	83.2	86.3	63.3	81.2	87.8	91.3
24	16535.75	65.1	82.8	88.6	91.0	68.0	85.3	91.3	94.2
28	12181.13	72.7	85.4	92.7	94.9	74.8	88.8	94.8	96.8
32	9276.84	75.2	90.4	94.2	97.0	76.1	91.4	95.4	98.0

Table 5. Experimental results on a Period-20 short integer sequence.

m	Fct	Nr2	Nr3	Nr4	Nr5	Nr6	No2	No3	No4
8	45.07	0.93	1.18	0.94	0.81	0.89	1.12	0.91	0.78
12	37.91	1.08	1.12	1.03	0.93	1.03	1.13	1.03	1.08
16	32.41	1.11	1.04	1.06	1.13	1.07	1.07	1.02	1.10
20	28.63	1.05	1.09	1.24	1.35	1.08	1.04	1.04	1.15
24	27.25	1.18	1.39	1.59	1.53	1.10	1.12	1.14	1.40
28	24.91	1.20	1.51	1.67	1.41	1.05	1.17	1.30	1.50
32	23.63	1.39	1.63	1.55	1.31	1.20	1.27	1.41	1.41
8	61386.36	50.0	73.3	67.7	50.7	56.3	81.3	89.0	94.9
12	36298.84	59.3	76.3	80.6	82.1	62.4	81.8	89.3	93.2
16	19385.18	70.4	84.0	88.8	90.8	72.8	88.7	94.2	96.5
20	10325.29	74.6	88.3	93.7	96.1	78.8	92.9	97.0	98.5
24	6566.03	82.4	94.9	97.5	98.7	84.9	96.1	98.4	99.4
28	3141.06	82.8	94.4	98.0	99.1	85.2	96.2	98.8	99.5
32	2399.46	88.3	97.1	99.1	99.7	89.6	97.8	99.3	99.8

Table 6. Experimental results on a Period-40 short integer sequence.

turns out to be the best choice especially on short patterns. The No approach is always less efficient of the Fct algorithm although the gain in number of detected false positives is always between 65% and 95%. This behavior is due to the high number of candidate occurrences detected by the algorithm, despite its gain in number of false positives, and to the relative effort in the construction of the filters values.

When the size of δ increases (see Table 5 and Table 6) 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.

The gain in number of false positives is always in the range between 50% and 99.7% for the Nr algorithm, and between 80% and 99.8% in the case of the No algorithm. The gain of the No4 algorithm is in most cases close the 100%.

6 Conclusions

In this paper we discussed the Order Preserving Pattern Matching Problem and presented two new families of filtering approaches to solve such problem which turn out to be much more effective in practice than the previously presented methods. The presented methods translate the original sequence on new sequences over large 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.

References

1. Boyer, R.S., Moore, J.S.: A fast string searching algorithm. Communications of the ACM 20(10), 762–772 (1977)
2. Chhabra, T., Tarhio, J.: Order-preserving matching with filtration. In: Proc. SEA '14, 13th International Symposium on Experimental Algorithms. Lecture Notes in Computer Science 8504, Springer, 307–314 (2014).
3. 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, Chengdu (2013)
4. Crochemore, M, Iliopoulos, C.S., Kociumaka, T., Kubica, M., Langiu, A., Pissis, S.P., Radoszewski, J., Rytter, W., Walen, T.: Order-preserving incomplete suffix trees and order-preserving indexes. In: Proc. SPIRE 2013, 20th International Symposium. LNCS, vol. 8214, pp. 84–95. Springer, Jerusalem (2013)
5. Durian, B., Holub, J., Peltola, H., Tarhio, J.: Improving practical exact string matching. Information Processing Letters 110(4): 148–152 (2010)
6. 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, 68–79 (2014)
7. 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)
8. Knuth, D.E., Morris, J.M., Pratt, V.R.: Fast pattern matching in strings. SIAM Journal on Computing 6(2), 323–350 (1977)
9. Navarro, G., Raffinot, M.: Flexible pattern matching in strings. Practical on-line search algorithms for texts and biological sequences. Cambridge University Press, New York, NY, 2002