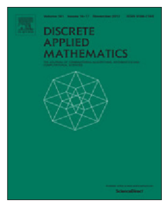
ARTICLE IN PRESS
Discrete Applied Mathematics ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam



The order-preserving pattern matching problem in practice☆

Domenico Cantone [a], Simone Faro [a,*], M. Oğuzhan Külekci [b]

[a] Università di Catania, Dipartimento di Matematica e Informatica, Italy
[b] Istanbul Technical University, Informatics Institute, Turkey

ARTICLE INFO

Article history:
Received 1 March 2017
Received in revised form 7 August 2018
Accepted 22 October 2018
Available online xxxx

Keywords:
Approximate text analysis
Experimental algorithms
Filtering algorithms
Text processing

ABSTRACT

Given a pattern x of length m and a text y of length n, both over a totally ordered alphabet, the *order-preserving pattern matching* (OPPM) problem consists in finding all substrings of the text with the same relative order as the pattern. The OPPM problem, 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 such diverse fields as time series analysis (as share prices on stock markets or weather data analysis) and musical melody matching, just to mention a few.

In this paper we present two new filtering approaches which turn out to be much more effective in practice than the previously presented methods, reducing the number of false positives up to 99%. We also present a new efficient approach inspired by the well-known Skip Search algorithm for the exact string matching problem. It makes use of efficient SIMD SSE instructions for speeding up the searching phase. Experimental results show that our proposed algorithms are up to twice faster than previous solutions.

© 2018 Elsevier B.V. All rights reserved.

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 one of the most important subjects in the wider domain of text processing, and algorithms which solve such problem are also basic components used in the implementations of practical software. String matching plays also an important role in theoretical computer science by providing challenging problems. The worst-case lower bound for string matching is $\mathcal{O}(n)$. This was achieved for the first time by the well-known algorithm by Knuth, Morris and Pratt [17]. However, many string matching algorithms have reached a sublinear $\mathcal{O}(n \log m/m)$ performance on average. Among them, the Boyer–Moore algorithm [3] deserves a special mention, since it has been particularly successful and has inspired much work.

The *order-preserving pattern matching problem* [16,9,2,12] (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 a totally ordered alphabet Σ. The task of the problem is to find all the substrings of the text having the same relative order as the pattern.

☆ This paper is based on preliminary results appeared in Cantone et al., (0000) [4] and in Faro and Külekci (2016) [12].
∗ Corresponding author.
E-mail addresses: cantone@dmi.unict.it (D. Cantone), faro@dmi.unict.it (S. Faro), kulekci@itu.edu.tr (M.O. Külekci).

https://doi.org/10.1016/j.dam.2018.10.023
0166-218X/© 2018 Elsevier B.V. All rights reserved.

Please cite this article in press as: D. Cantone, et al., The order-preserving pattern matching problem in practice, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.10.023.

ARTICLE IN PRESS

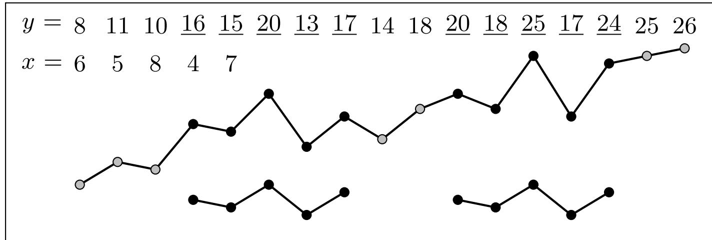


Fig. 1. Example of a pattern x of length 5 over an integer alphabet with two order-preserving occurrences in a text y of length 17, at positions 3 and 10.

For instance, the relative order of the sequence $x = \langle 6, 5, 8, 4, 7\rangle$ is the sequence $\langle 2, 1, 4, 0, 3\rangle$, since 6, 5, 8, 4, 7 have rank 2, 1, 4, 0, 3, respectively.[1] Thus x has an order-preserving occurrence in the sequence

$$y := \langle 8, 11, 10, \underline{16}, 15, 20, 13, 17, 14, 18, \underline{20}, 18, 25, 17, 24, 25, 26\rangle$$

at position 3, since x and the subsequence $\langle 16, 15, 20, 13, 17\rangle$ share the same relative order. Another order-preserving occurrence of x in y is at position 10 (see Fig. 1).

The OPPM problem finds applications in fields in which one is interested in finding patterns affected by relative orders, not by their absolute values. For example, it can be applied to time series analysis such as share prices on stock markets, weather data, or to musical melody matching in musical scores.

Related works

In the last few years, some solutions have been proposed for the order-preserving pattern matching problem. The first solution has been presented by Kubica et al. in 2013 (see [18]). They proposed an $\mathcal{O}(n+m\log m)$ solution over generic ordered alphabets, based on the Knuth–Morris–Pratt algorithm [17], and an $\mathcal{O}(n+m)$ solution in the case of integer alphabets. Some months later, Kim et al. [16] presented a similar solution, also based on the Knuth–Morris–Pratt approach and running in time $\mathcal{O}(n + m\log m)$. Although Kim et al. stressed some doubts about the applicability of the Boyer–Moore approach [3] to the order-preserving matching problem, in 2013 Cho et al. [9] presented a method for deciding the order-isomorphism between two sequences, proving that the Boyer–Moore approach can be applied also to the order-preserving variant of the pattern matching problem. We also mention that, in [2], it was shown 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 [7,8] for the OPPM based on filtration. Their algorithm translates the input sequences into two binary sequences and then uses any standard exact pattern matching algorithm as a filtration procedure. In particular, in their approach, a sequence s is translated into a binary sequence β of length $|s| - 1$ such that

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

for each $0 \leqslant i < |s|-1$. This translation is unique for any given sequence s, and can be performed online on the text, requiring constant time for each text character. When a candidate occurrence is found during the filtration phase, an additional verification procedure is run for checking whether the candidate substring is order-isomorphic with 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 the exact string matching problem 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 obtained in [7] show that the approach is faster than the algorithm by Cho et al. [9].

In 2015, other efficient filtration algorithms have been presented. Specifically, in [4] an efficient solution was presented 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 [6] two filtration online solutions, based on SSE and AVX instructions, respectively, have been proposed. It turns out from the experimental results conducted in [4] and in [6] that such solutions are faster than the previous algorithms in most cases.

[1] Conventionally, we count positions in a sequence starting from 0.

ARTICLE IN PRESS

Our contribution

In this paper, we present two new families of filtering approaches which turn out to be much more effective in practice than all previously presented methods. While the technique proposed by Chhabra and Tarhio translates the input strings into 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.

We also present an algorithm for the OPPM problem based on the well-known Skip Search algorithm [5] for the exact string matching problem, which consists in processing separately chunks of the text for any occurrence of the pattern. For all substrings of a given length of the pattern, a fingerprint is computed and indexed. Then, by using this information, candidate occurrences of the pattern are located in the text. We propose to use efficient SIMD SSE instructions [15] for computing the fingerprints of the pattern substrings.

Experimental results show that our solutions based on filtering are up to twice as fast as previous solutions and, under suitable conditions, the number of false positives is reduced up to 99%. Also, our approach based on Skip-Search leads to algorithmic variants that are up to two times faster than previous solutions present in the literature.

Organization of the paper

The paper is organized as follows. In Section 2, we provide preliminary notions and definitions relative to the order-preserving pattern matching problem. Subsequently, we present our new filter-based solutions in Section 3, whereas our solutions based on Skip-Search will be presented in Section 4. Finally, in Section 5 we shall evaluate the performances of our presented solutions against those obtained by the previous algorithms.

2. Notions and basic definitions

A string x over an alphabet Σ, of size σ, is defined as a finite sequence of elements in Σ. In this paper, we shall assume that the alphabet Σ is totally ordered by a relation "$\leqslant$".

By $|x|$, we denote the length of the string x. We refer to the ith 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 $0 \leqslant i \leqslant j < |x|$. Thus, $x[0]$ and $x[|x|-1]$ are respectively the first and last characters of x, and $x[0\,..\,|x|-1] = x$.

For integers i and j, with $i \leqslant j$, $[i\,..\,j]$ will denote the interval of all integers between i and j, including both i and j, whereas $[i\,..\,j)$ will denote the integer interval $[i\,..\,j-1]$.

2.1. Order-Isomorphism and related properties

We say that two (nonnull) sequences x, y over Σ 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 $(\Sigma, \leqslant)$, are said to be *order-isomorphic*, and we write $x \approx y$, if the following conditions hold

(a) $x[i] < x[j] \iff y[i] < y[j]$, for $0 \leqslant i, j < |x|$;
(b) $x[i] = x[j] \iff y[i] = y[j]$, for $0 \leqslant i, j < |x|$.

More succinctly, conditions (a) and (b) of Definition 1 can be summarized in the condition present in the following immediate lemma:

Lemma 1. *Let x and y be two nonnull sequences of the same length, over a totally ordered alphabet $(\Sigma, \leqslant)$. Then $x \approx y$ if and only if the following condition holds:*

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

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. These are provided next, together with some of their properties.

Definition 2 (*Rank Function*)**.** Let x be a nonnull sequence over a totally ordered alphabet $(\Sigma, \leqslant)$. For $i \in [0\,..\,|x|)$, let us set

$$x^{<}(i) := \{k : x[k] < x[i]\}$$
$$x^{\doteq}(i) := \{k : x[k] = x[i] \text{ and } k < i\}\,.$$

Then, the *rank function of* x is the bijection

$$rk_x \colon [0\,..\,|x|) \to [0\,..\,|x|)$$

Please cite this article in press as: D. Cantone, et al., The order-preserving pattern matching problem in practice, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.10.023.

defined by

$$rk_x(i) := |x^{<}(i)| + |x^{\doteq}(i)|,$$

namely,

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

for $0 \leqslant i < |x|$.

Thus, $rk_x(i)$ is the rank of $x[i]$ in the sequence x, namely the number of items in x strictly less than $x[i]$ plus the number of items in x equal to $x[i]$ and appearing in x before $x[i]$; hence the rank function rk_x is clearly bijective.

Example 1. Consider the following sequence of integers of length 7:

$$x = \langle 6, 3, 12, 3, 10, 11, 10 \rangle.$$

We have

- $x^{<}(0) = \{1, 3\}$ (since $x[1] = x[3] = 3 < 6 = x[0]$ and $x[i] \geqslant x[0]$, for $i \neq 1, 3$), and
- $x^{\doteq}(0) = \emptyset$.

Thus, $rk_x(0) = 2$. Similarly, we have

- $x^{<}(4) = \{0, 1, 3\}$ (since $x[0], x[1], x[3] < 10 = x[4]$ and $x[i] \geqslant x[4]$, for $i \neq 0, 1, 3$), and
- $x^{\doteq}(4) = \emptyset$.

Hence, $rk_x(4) = 3$.

Finally, we have $rk_x(6) = 4$, since $x^{<}(4) = x^{<}(6) = \{0, 1, 3\}$ and $x^{\doteq}(6) = \{4\}$.

The remaining values are: $rk_x(1) = 0$, $rk_x(2) = 6$, $rk_x(3) = 1$, and $rk_x(5) = 5$.

The following properties are easy consequences of Definition 2.

Lemma 2. *Given a nonnull sequence x over a totally ordered alphabet $(\Sigma, \leqslant)$, we have:*

(a) if $x[i] < x[j]$, then $rk_x(i) < rk_x(j)$, for $0 \leqslant i, j < |x|$;
(b) if $x[i] = x[j]$ and $0 \leqslant i < j < |x|$, then $rk_x(i) < rk_x(j)$.

Proof. Concerning (a), let $x[i] < x[j]$, with $0 \leqslant i, j < |x|$. Then $x^{<}(i) \cup x^{\doteq}(i) \cup \{i\} \subseteq x^{<}(j)$, and therefore $rk_x(i) + 1 \leqslant |x^{<}(j)| \leqslant rk_x(j)$, proving (a).

Next, concerning (b), let $x[i] = x[j]$, where $0 \leqslant i < j < |x|$. We have $x^{<}(i) = x^{<}(j)$ and $x^{\doteq}(i) \cup \{i\} \subseteq x^{\doteq}(j)$. Thus, $x^{<}(i) \cup x^{\doteq}(i) \cup \{i\} \subseteq x^{<}(j) \cup x^{\doteq}(j)$, so that $rk_x(i) + 1 \leqslant rk_x(j)$, proving (b). □

Following customary notations, by rk_x^{-1} we denote the inverse function of rk_x (which exists, as rk_x is bijective).

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

Proof. By way of contradiction, assume that $x[rk_x^{-1}(i)] > x[rk_x^{-1}(i+1)]$. Then, by Lemma 2(a), we have $i + 1 = rk_x\big(rk_x^{-1}(i+1)\big) < rk_x\big(rk_x^{-1}(i)\big) = i$, a contradiction. □

For any nonnull sequence x, we shall refer to the sequence

$$\langle rk_x^{-1}(0), rk_x^{-1}(1), \ldots, rk_x^{-1}(|x| - 1)\rangle$$

as the *relative order* of x (see Example 2).

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*.

Definition 3 (*Equality Function*). Let x be a sequence of length $m \geqslant 2$ over a totally ordered alphabet $(\Sigma, \leqslant)$. The *equality function* of x is the binary map $eq_x \colon [0 \,..\, m-2] \to \{0, 1\}$, where

$$eq_x(i) := \begin{cases} 1TIZ@DTIZ@DTIZ@DTIZ@DTIZ@DTIZ@DTIZ@DTIZ@DAMZ@Pif\ TIZ@Dx[rk_x^{-1}(i)] = x[rk_x^{-1}(i+1)] \\ 0AMZ@Potherwise, \end{cases}$$

for $0 \leqslant i \leqslant m - 2$.

Please cite this article in press as: D. Cantone, et al., The order-preserving pattern matching problem in practice, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.10.023.

The rank and equality functions allow a complete characterization of order-isomorphism, as stated in the following lemma.

Lemma 3. *For any two sequences x and y of the same length $m \geqslant 2$, over a common totally ordered alphabet, we have*

$$x \approx y \quad \Longleftrightarrow \quad rk_x = rk_y \text{ and } eq_x = eq_y.$$

Proof. Let us first assume that $x \approx y$. We show that $rk_x = rk_y$, by proving that $rk_x(i) = rk_y(i)$ for every $0 \leqslant i < m$. Thus, let $0 \leqslant i < m$. Then, by conditions (a) and (b) of Definition 1, we have:

$$\begin{aligned} rk_x(i) &= \big|\{k : x[k] < x[i] \text{ or } (x[k] = x[i] \text{ and } k < i)\}\big| \\ &= \big|\{k : y[k] < y[i] \text{ or } (y[k] = y[i] \text{ and } k < i)\}\big| \qquad \text{(by Lemma 1)} \\ &= rk_y(i)\,, \end{aligned}$$

so that $rk_x = rk_y$.

Next we show that $eq_x = eq_y$, by proving that $eq_x(i) = eq_y(i)$ for every $i \in [0\,..\,m-2]$. Thus, let $0 \leqslant i \leqslant m-2$. Then, by Definition 1(b) and since $rk_x = rk_y$ (as we have just shown), we have:

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

and therefore $eq_x = eq_y$.

Conversely, let $rk_x = rk_y$ and $eq_x = eq_y$ hold, but assume, by way of contradiction, that $x \not\approx y$. Then, without loss of generality, we have that $x[i] \leqslant x[j]$ and $y[i] > y[j]$ hold, for some $0 \leqslant i, j < m$. The latter yields $rk_y(i) > rk_y(j)$ and, therefore, we have also $rk_x(i) > rk_x(j)$. Since $x[i] \leqslant x[j]$, Lemma 2(a) yields $x[i] = x[j]$. Hence, for every k such that $rk_x(j) \leqslant k < rk_x(i)$, we have $x[rk_x^{-1}(k)] = x[i] = x[rk_x^{-1}(k+1)]$, so that $eq_x(k) = 1$, and therefore $eq_y(k) = 1$. Thus, for every k such that $rk_y(j) \leqslant k < rk_y(i)$, we have $y[rk_y^{-1}(k)] = y[rk_y^{-1}(k+1)]$; hence, by induction, $y[rk_y^{-1}(rk_y(j))] = y[rk_y^{-1}(rk_y(i))]$, i.e., $y[j] = y[i]$, contradicting our assumption that $y[i] > y[j]$. Therefore, $x \approx y$ holds, completing the proof of the lemma. □

Example 2. Consider the following three sequences of integers, all of length 7:

$$x = \langle 6, 3, 8, 3, 10, 7, 10\rangle, \quad y = \langle 2, 1, 4, 1, 5, 3, 5\rangle, \quad z = \langle 6, 3, 8, 4, 9, 7, 10\rangle.$$

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, in agreement with Lemma 3, we have $eq_x = eq_y = \langle 1, 0, 0, 0, 0, 1\rangle$ and $eq_z = \langle 0, 0, 0, 0, 0, 0\rangle$.

Based on the preceding lemma, 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, and then compare them. The cost of such a test is dominated by the cost $\mathcal{O}(m \log m)$ of sorting the two sequences. However, if one needs to find all the sequences from a set $\mathcal{S}$ that are order-isomorphic to a fixed sequence (all sequences having the same size m), the simple iteration of the previous test would lead to an overall complexity of $\mathcal{O}(|\mathcal{S}| \cdot m \log m)$. In this case a better approach is possible, with an overall complexity of $\mathcal{O}\big((|\mathcal{S}| + \log m) \cdot m\big)$, based on the following characterization of order-isomorphism which requires the computation of the rank and equality functions of the fixed sequence only.

Lemma 4. *Let x and y be two sequences of the same length $m \geqslant 2$, over a totally ordered alphabet. Then $x \approx y$ if and only if the following conditions hold:*

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

Proof. Let us first assume that $x \approx y$. Then $rk_x = rk_y$ and $eq_x = eq_y$. Thus, Corollary 1 yields that, for $0 \leqslant i < m-1$,

$$y[rk_x^{-1}(i)] = y[rk_y^{-1}(i)] \leqslant y[rk_y^{-1}(i+1)] = y[rk_x^{-1}(i+1)],$$

proving (i). Concerning (ii), let us first consider the case in which $y[rk_x^{-1}(i)] = y[rk_x^{-1}(i+1)]$, for some $0 \leqslant i < m-1$. Then

$$y[rk_y^{-1}(i)] = y[rk_x^{-1}(i)] = y[rk_x^{-1}(i+1)] = y[rk_y^{-1}(i+1)]$$

Please cite this article in press as: D. Cantone, et al., The order-preserving pattern matching problem in practice, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.10.023.

```
ORDER-ISOMORPHIC(inv-rk, eq, y)
1.  for i ← 0 to |y| − 2 do
2.     if (y[inv-rk(i)] > y[inv-rk(i + 1)]) then return false
3.     if (y[inv-rk(i)] < y[inv-rk(i + 1)] and eq(i) = 1) then return false
4.     if (y[inv-rk(i)] = y[inv-rk(i + 1)] and eq(i) = 0) then return false
5.  return true
```

Fig. 2. The procedure ORDER-ISOMORPHIC verifies whether an input sequence y is order-isomorphic to another sequence (of length $|y|$) having inverse rank function *inv-rk* and equality function *eq*.

and, therefore, $eq_y(i) = 1$, so that $eq_x(i) = 1$ holds. Conversely, if $eq_x(i) = 1$, then $eq_y(i) = 1$, and therefore $y[rk_y^{-1}(i)] = y[rk_y^{-1}(i+1)]$, which in turn implies $y[rk_x^{-1}(i)] = y[rk_x^{-1}(i+1)]$, completing the proof of (ii).

For the second half of the proof of the lemma, let us assume now that (i) and (ii) hold. By Definition 1, it is enough to show that x and y have the same rank and equality functions.

To prove that the rank functions of x and y coincide, we can just show that, for $0 \leqslant i < m$,

$$\{k : y[k] < y[i] \text{ or } (y[k] = y[i] \text{ and } k < i)\} = \{k : rk_x(k) < rk_x(i)\} \tag{2}$$

holds, since the cardinalities of the sets on the left- and right-hand side of (2) are $rk_y(i)$ and $rk_x(i)$, respectively. We prove (2), by showing that the following three facts hold.

Fact 1. If $y[k] < y[i]$, then $rk_x(k) < rk_x(i)$.

Indeed, let $y[k] < y[i]$, $k' = rk_x(k)$, and $i' = rk_x(i)$. Then, $y[rk_x^{-1}(k')] < y[rk_x^{-1}(i')]$, so that, by (i), $k' < i'$, i.e., $rk_x(k) < rk_x(i)$.

Fact 2. If $y[k] = y[i]$ and $k < i$, then $rk_x(k) < rk_x(i)$.

Let $y[k] = y[i]$, with $k < i$, and, as before, $k' = rk_x(k)$ and $i' = rk_x(i)$. Then, $y[rk_x^{-1}(k')] = y[rk_x^{-1}(i')]$. Toward a contradiction, let us assume that $i' < k'$. Then, by (i), $y[rk_x^{-1}(i')] = y[rk_x^{-1}(i'+1)] = \cdots = y[rk_x^{-1}(k')]$, so that, by (ii), $eq_x(i') = eq_x(i'+1) = \cdots = eq_x(k'-1) = 1$, and therefore $x[rk_x^{-1}(i')] = x[rk_x^{-1}(k')]$, i.e., $x[i] = x[k]$. Hence, by Lemma 2(b), we have $k' = rk_x(k) < rk_x(i) = i'$, contradicting our assumption that $i' < k'$. Thus, $k' < i'$ must hold, i.e., $rk_x(k) < rk_x(i)$.

Fact 3. If $rk_x(k) < rk_x(i)$, then either (a) $y[k] < y[i]$ or (b) $y[k] = y[i]$ and $k < i$.

Let $rk_x(k) < rk_x(i)$. Then, from Fact 1, we have $y[k] \leqslant y[i]$, so that it only remains to show that if $y[k] = y[i]$, then $k < i$. Thus, let us assume that $y[k] = y[i]$ holds, i.e., $y[rk_x^{-1}(k')] = y[rk_x^{-1}(i')]$ (where, as above, we have put $k' = rk_x(k)$ and $i' = rk_x(i)$). Reasoning as in Fact 2, one can show that $x[rk_x^{-1}(k')] = x[rk_x^{-1}(i')]$ holds as well. But since, by hypothesis, $k' = rk_x(k) < rk_x(i) = i'$, by Lemma 2(b) we get $k < i$, concluding the proof of Fact 3.

From Facts 1–3, (2) follows at once (for $0 \leqslant i < m$), so that, as already observed, $rk_x = rk_y$ holds.

Next we show that the equality functions of x and y coincide as well. Let $0 \leqslant i < m - 1$. Then we have

$$\begin{aligned} eq_y(i) = 1 &\iff y[rk_y^{-1}(i)] = y[rk_y^{-1}(i+1)] && \text{(by definition)} \\ &\iff y[rk_x^{-1}(i)] = y[rk_x^{-1}(i+1)] && (\text{since } rk_x = rk_y) \\ &\iff eq_x(i) = 1 && \text{(by (ii))}. \end{aligned}$$

Thus, $eq_x = eq_y$. Since, as we have already proved, we have also $rk_x = rk_y$, it follows that $x \approx y$, concluding the second half of the proof of the lemma. □

Based on Lemma 4, the procedure ORDER-ISOMORPHIC in Fig. 2 correctly verifies whether an input sequence y is order-isomorphic to another sequence of the same length as y, say x, given its inverse rank and equality functions. In other words, it receives as input the functions rk_x^{-1} 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 at 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.

The OPPM problem consists in finding all the substrings of the text with the same relative order as the pattern. More precisely, we have:

Definition 4 (*Order-preserving Pattern Matching*)**.** Let x and y be two sequences of length m and n, respectively, with $n > m$, both over a totally ordered alphabet $(\Sigma, \leqslant)$. The *order-preserving pattern matching problem* consists in finding all positions $i \in [0 \,..\, n-m]$ 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*.

In Section 4.2, we shall present an algorithm for the OPPM problem, based on the Alpha Skip Search algorithm. For convenience, we briefly review it next.

Please cite this article in press as: D. Cantone, et al., The order-preserving pattern matching problem in practice, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.10.023.

3. New efficient filter based algorithms

In this section, we present two new general approaches to the OPPM problem. Both of them are based on a filtration technique, as in [7], but we use information extracted from groups of integers in the input string, as in [9], 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, the *q-neighborhood* of the element $x[i]$ at position i in x, with $0 \leqslant i < m - q$, is the sequence of $q + 1$ elements from position i to $i + q$ in x, i.e., the sequence $\langle x[i], x[i + 1], \ldots, x[i + q]\rangle$.

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 used to manage grams and with the time efficiency of the overall searching algorithm.

When using q-grams, greater accuracies require larger 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 $x[i] \geqslant x[j]$. For brevity, $\beta_x(i, j)$ will denote the Boolean value resulting from the above inequality, extending the formal definition given in (1). Formally, we have

$$\beta_x(i,j) := \begin{cases} 1 & \text{if } x[i] \geqslant x[j] \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

It is easy to observe that if $\beta_x(i, j) = 1$ we have that $rk_x^{-1}(i) \geqslant rk_x^{-1}(j)$ ($x[j]$ 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 $0 \leqslant i < m - q$) 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 $\beta(i, i)$, as it does not give any additional information.

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

More formally, we have the following definition.

Definition 6 (*q-NR Sequences*)**.** 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 $\{0, 1, \ldots, 2^q\}$ where

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

Example 3. 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 into an 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 into an NR value equal to 6. The whole 4-NR sequence of length 6 associated to x is $\chi_x^4 = \langle 4, 8, 1, 6, 15, 8\rangle$ (see Fig. 3).

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

Lemma 5. *Let x and y be two sequences of length m and let χ_x^q and χ_y^q be the q-ranking sequences associated to x and y, respectively. If $x \approx y$, then $\chi_x^q = \chi_y^q$.*

Proof. Let rk_x be the rank function associated to x. Assuming by hypothesis that $x \approx y$, the following statements hold:

1. $x[rk_x^{-1}(i)] \leqslant x[rk_x^{-1}(i + 1)]$, for $0 \leqslant i < m - 1$ (by Definition 2);

Neighborhood Ranking	Example	nr seq.	$\chi^3_x[i]$
$x[i] \leq x[i+1], x[i+2], x[i+3]$		$\langle 0,0,0\rangle$	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 values. In the leftmost column we show the ranking position of $x[i]$ compared with other elements in its neighborhood $\langle x[i], x[i+1], x[i+2], x[i+3]\rangle$.

2. $y[rk_x^{-1}(i)] \leqslant y[rk_x^{-1}(i+1)]$, for $0 \leqslant i < m-1$ (by hypotheses and Lemma 1);
3. $x[i] \leqslant x[j] \iff y[i] \leqslant y[j]$, for $0 \leqslant i, j < m-1$ (by 1 and 2 above);
4. $x[i] \geqslant x[i+j] \iff y[i] \geqslant y[i+j]$, for $0 \leqslant i < m-q$ and $1 \leqslant j < q$ (by 3);
5. $\beta_x(i, i+j) = \beta_y(i, j+j)$, for $0 \leqslant i < m-q$ and $1 \leqslant j < q$ (by 4);
6. $\chi_x^q[i] = \chi_y^q[i]$, for $0 \leqslant i < m-q$ (by 5 and Definition 6).

Statement 6 proves the lemma. □

The following corollary, which follows trivially from Lemma 5, proves that the NR approach can be used as a filtering.

Corollary 2. *Let x and y be two sequences of length m and n, respectively, and let χ_x^q and χ_y^q be the q-ranking sequences associated to x and y, respectively. If $x \approx y[j\mathinner{..}j+m-1]$, then $\chi_x^q[i] = \chi_y^q[j+i]$, for $0 \leqslant i < m-q$.* □

Fig. 5 (on the left) shows the procedure for computing the NR value associated with the element of the string x at position i, whose time complexity is linear in the size q of the neighborhood. 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, the sequence χ_x^q and the functions rk_x and eq_x are computed. When an occurrence of χ_x^q is found at position i, the verification procedure Order-Isomorphic shown in Fig. 2 is run to check whether $x \approx y[i\mathinner{..}i+m-1]$.

Since, in the worst case, the algorithm finds a candidate occurrence at each text position, and each verification cost $\mathcal{O}(m)$, the worst-case time complexity of the algorithm is $\mathcal{O}(nm)$, while the filtration phase can be performed with an $\mathcal{O}(nq)$ worst-case time complexity. However, following the same analysis as in [7], one can easily prove that the verification time approaches to 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 too.

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 the latter case, it is enough to compare the element $x[j]$, where $i \leqslant j < i+q$, with each element $x[h]$, where $j < h \leqslant i+q$. Thus each element at position i in x, with $0 \leqslant i < m-q$, is associated with a binary sequence of length $q(q+1)/2$ which completely describes the relative order of the subsequence $x[i\mathinner{..}i+q]$. Since there are $(q+1)!$ possible permutations of a set of $q+1$ elements, the string x is converted into a sequence φ_x^q of length $m-q$, where each element $\varphi_x^q[i]$ is a value such that $0 \leqslant \varphi_x^q[i] < q(q+1)/2$. More formally, we have the following definition.

Please cite this article in press as: D. Cantone, et al., The order-preserving pattern matching problem in practice, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.10.023.

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. 4. The 3! possible orderings 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-NR-Value(x, i, q)
1. δ ← 0
2. for j ← 1 to q do
3.     δ = (δ ≪ 1) + β_x(i, i + j)
4. return δ
```

```
Compute-NO-Value(x, i, q)
1. δ ← 0
2. for k ← q downto 1 do
3.     for j ← 1 to k do
4.         δ = (δ ≪ 1) + β_x(i + q − k, i + q − k + j)
5. return δ
```

Fig. 5. The two functions which compute the q-neighborhood ranking value (on the left) and the q-neighborhood ranking value (on the right).

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

$$\varphi_x^q[i] = \sum_{k=1}^{q} \left(\chi_x^k[i+q-k] \times 2^{(k)(k-1)/2}\right), \text{ for } 0 \leqslant i < m-q. \tag{4}$$

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

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

Example 4. As in Example 3, 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 at position 2 is therefore $\langle 0, 1, 1, 1, 1, 1\rangle$ which translates into an NO value equal to $\varphi_x[3] = 31$. In a similar way, we can observe that the NR sequence associated with the element at position 2 is $\langle 0, 0, 0, 0, 1, 1\rangle$ which translates into an NO value equal to $\varphi_x^4[2] = 3$. The whole sequence of length 7 associated to x is $\varphi_x^4 = \langle 20, 32, 3, 31, 60, 32, 3\rangle$ (see Fig. 4).

The following lemma (which easily follows from Definition 7 and Lemma 5) and related corollary yield that the NO approach can be used to filter a text y to search for all order-preserving occurrences of a pattern x. In other words, they imply that

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

Lemma 6. *Let x and y be sequences of length m, and let φ_x^q and φ_y^q be the q-NO sequences associated to x and y, respectively. If $x \approx y$, then $\varphi_x^q = \varphi_y^q$.* □

The following corollary, which trivially follows from Lemma 6, states that the NO approach can be used as filtering.

Corollary 3. *Let x and y be two sequences of length m and n, respectively, and let φ_x^q and φ_y^q be the q-ordering sequences associated to x and y, respectively. If $x \approx y[j \mathinner{..} j+m-1]$ then $\varphi_x^q[i] = \varphi_y^q[j+i]$, for $0 \leqslant i < m-q$.* □

Fig. 5 (on the right) shows a procedure for computing the NO value associated with the element of the string x at position i. Its time complexity is $\mathcal{O}(q^2)$. Thus, given a pattern x and a text y, of length m and n respectively, and an integer constant $q < m$, 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, the sequence φ_x^q and the functions rk_x and eq_x are computed. When an occurrence of φ_x^q is found at position i, the verification procedure Order-Isomorphic is run to check whether $x \approx y[i \mathinner{..} i+m-1]$. As before, if the filtration algorithm is sublinear on average, the NO approach has a sublinear behavior on average too.

4. New efficient skip-search based algorithms

In this section, we present a new algorithm, called Order-Preserving-Skip-Search, for the OPPM problem. However, for brevity, in the following we shall often refer to it as SkSop algorithm.

Our algorithm combines the same approach of the Skip Search algorithm with the power of the SIMD (Single Instruction Multiple Data) instruction set, and specifically the Intel SSE (Streaming SIMD Extensions) instruction set, as discussed below.

Before entering into details, we briefly review in the following subsection the well-known Skip Search algorithm.

4.1. The skip search algorithm and its alpha variant

The Skip Search algorithm is an elegant and efficient solution to the exact pattern matching problem, first presented in [5] and subsequently adapted to many other problems and variants of the exact pattern matching problem.

Let x and y be a pattern and a text of length m and n, respectively, over a common alphabet Σ of size σ. For each character c of the alphabet, the Skip Search algorithm collects in a bucket $B[c]$ all the positions of that character in the pattern x, so that for each $c \in \Sigma$ we have:

$$B[c] = \{i \ : \ 0 \leqslant i < m \text{ and } x[i] = c\}.$$

Clearly, the space and time complexity needed for the construction of the array B of buckets is $\mathcal{O}(m + \sigma)$. Notice that when the pattern is shorter than the alphabet size, some buckets are empty.

The search phase of the Skip Search algorithm examines all the characters $y[j]$ in the text at positions $j = km - 1$, for $k = 1, 2, \ldots, \lfloor n/m \rfloor$. For each such character $y[j]$, the bucket $B[y[j]]$ allows one to compute the possible positions h of the text in the neighborhood of j at which the pattern could occur.

By performing a character-by-character comparison between x and the subsequence $y[h \,..\, h + m - 1]$ until either a mismatch is found, or all the characters in the pattern x have been considered, it can be tested whether x actually occurs at position h of the text.

The Skip Search algorithm has a quadratic worst-case time complexity; however, as shown in [5], the expected number of text character inspections is $\mathcal{O}(n)$.

Among the variants of the Skip Search algorithm, the most relevant one for our purposes is the Alpha Skip Search algorithm [5], which collects buckets for substrings of the pattern rather than for single characters.

During the preprocessing phase of the Alpha Skip Search algorithm, all the factors of length $\ell = \lfloor \log_\sigma m \rfloor$ occurring in the pattern x are arranged in a trie T_x for fast retrieval. In addition, for each leaf ν of T_x, a bucket is maintained which stores the positions in x of the factor corresponding to ν. Provided that the alphabet size is considered as a constant, the worst-case running time of the preprocessing phase is $\mathcal{O}(m)$.

The searching phase consists in looking into the buckets of the text factors $y[j \,..\, j + \ell - 1]$, for all $j = k(m - \ell + 1) - 1$ such that $1 \leqslant k \leqslant \lfloor (n - \ell)/m \rfloor$, and then test, as in the previous case, whether there is an occurrence of the pattern at the indicated positions of the text.

The worst-case time complexity of the searching phase is quadratic, though the expected number of text character comparisons is $\mathcal{O}(n \log_\sigma m/(m - \log_\sigma m))$.

4.2. A skip-search approach to order-preserving pattern matching

In the last two decades, a lot of efforts have been spent to exploit the power of the word-RAM model of computation to speed-up string matching algorithms for a single pattern.

In this model, the computer operates on words of length w, so that usual arithmetic and logic operations on words take one unit of time. Most of the solutions which exploit the word-RAM model are based on the *bit-parallelism* technique [1] or on the *packed string matching* technique [10,11]. In the packed string matching technique, multiple characters can be packed into a single word, so that the characters can be compared in bulk rather than individually.

Next we discuss our model in detail.

4.3. The model

In the design of our algorithm, we use specialized word-size packed string matching instructions, based on the Intel streaming SIMD extensions (SSE) technology. SIMD instructions exist in many recent microprocessors supporting parallel execution of some operations on multiple data via a set of special instructions working on a limited number of special registers.

In our model of computation, we assume that w is the number of bits in a word and σ is the size of the alphabet. The *packing factor* $\alpha = w/\log \sigma$ (or, rather, its floor) is the number of characters which fit into a single computer word, whereas the number of bits used to encode an alphabet character is $\gamma = \log \sigma$.

In most practical applications, we have $\sigma = 256$ (ASCII code). Moreover SSE specialized instructions allow one to work on 128-bit registers, so that blocks of sixteen 8-bit characters can be read and processed in a single time unit ($\alpha = 16$). In particular, our algorithm makes use of specialized word-size packed instructions which we call wsrv (*word-size rank vector*) and wsrp (*word-size relative position*). These are reviewed next.

ARTICLE IN PRESS

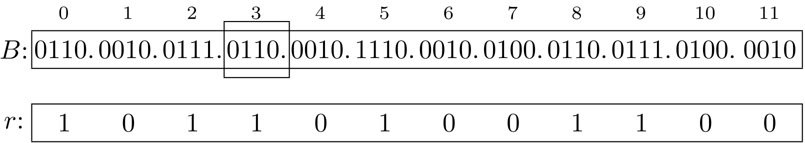


Fig. 6. An example of the application of wsrv$(B, 3)$, assuming $w = 48$ and $\alpha = 12$.

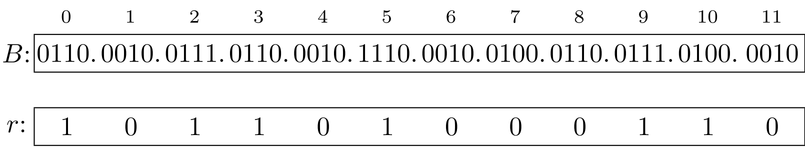


Fig. 7. An example of the application of wsrp(B), assuming $w = 48$ and $\alpha = 12$.

The instruction wsrv

The instruction wsrv(B, i) computes an α-bit fingerprint from a w-bit register B handled as a block of α small integer values. Assuming that $B[0 \mathinner{..} \alpha - 1]$ is a w-bit integer parameter, wsrv(B, i) returns an α-bit value $r[0 \mathinner{..} \alpha - 1]$, where $r[j] = 1$ if and only if $B[i] \geqslant B[j]$, and $r[j] = 0$ otherwise (see Fig. 6).

Fig. 7 shows an example of the application of wsrv$(B, 3)$ assuming $w = 48$ and $\alpha = 12$. The wsrv(B, i) specialized instruction can be emulated in constant time by the following sequence of specialized SIMD instructions:

wsrv(B, i)

$D \leftarrow$ _mm_set1_epi8$(B[i])$

$C \leftarrow$ _mm_cmpgt_epi8(B, D)

$r \leftarrow$ _mm_movemask_epi8(C)

return r

Specifically, the _mm_set1_epi8$(B[i])$ instruction creates a w-bit register D handled as a block of α small integers values, where $D[j] = B[i]$ for $0 \leqslant j < \alpha$. The _mm_cmpgt_epi8(B, D) instruction compares the α integers in B and the α integers in D for "greater than". It creates a w-bit register C handled as a block of α small integers where $C[j] = 1^\gamma$ if $B[j] \geqslant D[j]$, and $C[j] = 0^\gamma$ otherwise (we recall that $\gamma = \log \sigma$ is the number of bits to encode an alphabet character). Finally, the _mm_movemask_epi8(D) instruction gets a 128 bit parameter D, handled as sixteen 8-bit integers, and creates a 16-bit mask from the most significant bits of the 16 integers in D, with zero extending the upper bits.

The instruction wsrp

The instruction wsrp(B) computes an α-bit fingerprint from a w-bit register B handled as a block of α small integers values. Assuming that $B[0 \mathinner{..} \alpha - 1]$ is a w-bit integer parameter, wsrp(B) returns an α-bit value $r[0 \mathinner{..} \alpha - 1]$, where $r[j] = 1$ if and only if $B[j] \geqslant B[j + 1]$, and $r[j] = 0$ otherwise (we put $r[\alpha - 1] = 0$).

Fig. 7 shows an example of the application of wsrp(B), assuming $w = 48$ and $\alpha = 12$. The wsrp(B) specialized instruction can be emulated in constant time by the following sequence of specialized SIMD instructions

wsrp(B)

$D \leftarrow$ _mm_slli_si128$(B, 1)$

$C \leftarrow$ _mm_cmpgt_epi8(B, D)

$r \leftarrow$ _mm_movemask_epi8(C)

return r

where the _mm_slli_si128$(B, 1)$ instruction shifts the w-bit register in B to the left by one position (α bits) while shifting in zeros, and the _mm_cmpgt_epi8 and the _mm_movemask_epi8 instructions are as described above.

Please cite this article in press as: D. Cantone, et al., The order-preserving pattern matching problem in practice, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.10.023.

```
FNG(x, i, q, k)
1.  B ← 0^(α−q).x[i .. i + q − 1]
2.  v ← wsrp(B)
3.  for j ← 0 to k − 2 do
4.      v ← (v ≪ 1)+ wsrv(B, α − q + j)
5.  return v
```

```
EXAMPLE (q = 5, k = 3, and α = 8)
x[i .. i + q − 1] = ⟨3, 6, 2, 4, 7⟩
               B = [0, 0, 0, 3, 6, 2, 4, 7]
         wsrp(B) = [0, 0, 0, 1, 0, 1, 1, 0] = 22_10
      wsrv(B, 3) = [0, 0, 0, 1, 1, 0, 1, 1] = 27_10
      wsrv(B, 4) = [0, 0, 0, 0, 1, 0, 0, 1] = 9_10
v = 22 × 2^2 + 27 × 2^1 + 9 = 151_10
```

Fig. 8. On the left: the pseudo-code of the procedure FNG for the computation of the fingerprint of a substring a length q combining k distinct fingerprints. On the right: an example of a computation of a fingerprint by the procedure FNG.

```
SKSOP-PREPROCESSING(x, q, m, k)
1.  for v ← 0 to 2^α − 1 do
2.      F[v] ← ∅
3.  for i ← 0 to m − q do
4.      v ← FNG(x, i, q, k)
5.      F[v] ← F[v] ∪ {(i + q − 1)}
6.  return F
```

```
SKSOP(x, r, y, n, q, k)
1.  F ← SKSOP-PREPROCESSING(x, q, m, k)
2.  for j ← m − 1 to n step m − q + 1 do
3.      v ← FNG(y, j, q, k)
4.      for each i ∈ F[v] do
5.          z ← y[j − i .. j − i + m − 1]
6.          if ORDER-ISOMORPHIC(rk_x^−1, eq_x, z)
7.              then output (j)
```

Fig. 9. The pseudo-codes of the SKSOP algorithm and its preprocessing for the OPPM problem.

4.4. The fingerprint functions

The preprocessing phase of the algorithm indexes the subsequences of the pattern (of length q) in order to locate them during the searching phase. For efficiency reasons, each numeric sequence of length q is converted into a numeric value, called *fingerprint*, which is used to index the substring. A fingerprint value ranges in the interval $[0 \mathinner{..} \tau - 1]$, for a given bound τ. The value τ is set to 2^{16}, so that a fingerprint can fit into a single 16-bit register.

The procedure FNG for computing the fingerprints is shown in Fig. 8 (on the left). Given a sequence x of length m, an index i such that $0 \leqslant i < m - q$, and two integers k and q such that $k \leqslant q \leqslant m$, the procedure FNG combines k different values computed on the substring $x[i \mathinner{..} i + q - 1]$ in order to calculate the fingerprint v. Preliminarily, the substring $x[i \mathinner{..} i + q - 1]$ is inserted in the rightmost portion of a w-bit register B. Then the fingerprint v is computed as

$$v = \mathsf{wsrp}(B) \times 2^{k-1} + \sum_{j=0}^{k-2} \left(\mathsf{wsrv}(B, \alpha - q + j) \times 2^{k-2-j} \right).$$

Clearly, the time complexity of the procedure FNG is $\mathcal{O}(k)$.

Fig. 8 (on the right) shows an example of how the procedure FNG works on a subsequence $x[i \mathinner{..} i + q - 1] = \langle 3, 6, 2, 4, 7 \rangle$ of length $q = 5$, combining $k = 3$ different values.

4.5. The order-preserving skip search algorithm

We are now ready to briefly describe our algorithm, named SKSOP, for the OPPM problem, based on the Alpha variant of the Skip Search algorithm. We distinguish in it a preprocessing and a searching phase.

The preprocessing phase of the SKSOP algorithm, reported in Fig. 9 (on the left), consists in compiling the fingerprints of all possible substrings of length q contained in the pattern x. Thus, a fingerprint value v, with $0 \leqslant v < 2^{\alpha}$, is computed for each subsequence $x[i \mathinner{..} i + q - 1]$, for $0 \leqslant i < m - q$.

To this purpose, a table F of size 2^{α} is maintained for storing, for any possible fingerprint value v, the set of positions i such that $\textsc{fng}(x, i, q, k) = v$. Hence, for $0 \leqslant v < 2^{\alpha}$, we have

$$F[v] := \left\{ i \mid 0 \leqslant i < m - q \text{ and } \textsc{fng}(x, i, q, k) = v \right\}.$$

The preprocessing phase of the SKSOP algorithm requires some additional space to store the $(m - q)$ possible alignments in the 2^{α} locations of the table F. Thus, the space requirement of the algorithm is $\mathcal{O}(m - q + 2^{\alpha})$ that approximates to $\mathcal{O}(m)$ since α is constant. The first for-loop of the preprocessing phase just initializes the table F, whereas the second for-loop is run $(m - q)$ times, which makes the overall time complexity of the preprocessing phase $\mathcal{O}(m + 2^{\alpha})$. The latter, again, approximates to $\mathcal{O}(m)$.

The basic idea of the searching phase is to compute a fingerprint value every $(m - q)$ positions of the text y and to check whether the pattern appears in y, involving the block $y[j \mathinner{..} j + q - 1]$. If the fingerprint value indicates that some of the alignments are possible, then the candidate positions are checked naively for matching.

The pseudo-code provided in Fig. 9 (on the right) reports the skeleton of the SKSOP algorithm. The main loop investigates the blocks of the text y in steps of $(m - q + 1)$ blocks. If the fingerprint v computed on $y[j \mathinner{..} j + q - 1]$ points to a nonempty bucket of the table F, then the positions listed in $F[v]$ are verified accordingly.

Please cite this article in press as: D. Cantone, et al., The order-preserving pattern matching problem in practice, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.10.023.

ARTICLE IN PRESS

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

δ	m	FCT	SkSop	SSE	NR2	NR3	NR4	NR5	NR6	NO2	NO3	NO4
5	8	44.29	$1.27^{(5,4)}$	1.45	1.16	1.28	1.25	1.25	1.24	1.89	1.71	1.11
	12	28.39	$1.37^{(4,5)}$	1.38	1.16	1.37	1.37	1.33	1.19	1.64	2.00	1.64
	16	20.65	$1.52^{(4,5)}$	1.23	1.15	1.30	1.43	1.34	1.14	1.42	2.01	1.83
	20	16.29	$1.58^{(4,5)}$	1.12	1.15	1.30	1.45	1.41	1.14	1.39	2.00	1.93
	24	13.64	$1.63^{(5,4)}$	1.05	1.16	1.29	1.42	1.44	1.12	1.34	1.91	2.01
	28	11.48	$1.62^{(5,4)}$	0.99	1.16	1.28	1.44	1.45	1.11	1.31	1.88	1.96
	32	10.34	$1.60^{(5,4)}$	0.83	1.18	1.30	1.40	1.46	1.12	1.30	1.83	2.05
20	8	42.34	$1.22^{(3,4)}$	1.68	1.13	1.27	1.25	1.26	1.22	1.92	1.68	1.08
	12	27.93	$1.40^{(4,5)}$	1.44	1.17	1.40	1.37	1.32	1.21	1.71	2.04	1.63
	16	20.05	$1.46^{(4,5)}$	1.32	1.15	1.32	1.41	1.33	1.15	1.48	2.04	1.81
	20	15.85	$1.51^{(4,5)}$	1.21	1.15	1.29	1.42	1.37	1.11	1.38	2.00	1.90
	24	13.31	$1.55^{(5,6)}$	1.12	1.17	1.31	1.47	1.42	1.12	1.36	1.99	2.02
	28	11.38	$1.58^{(5,6)}$	1.01	1.17	1.31	1.42	1.45	1.09	1.35	1.94	2.07
	32	9.96	$1.58^{(3,7)}$	0.97	1.16	1.29	1.45	1.46	1.09	1.29	1.87	2.09
40	8	42.62	$1.19^{(3,4)}$	1.91	1.16	1.28	1.28	1.25	1.25	1.94	1.70	1.09
	12	28.35	$1.39^{(4,5)}$	1.82	1.19	1.41	1.39	1.36	1.21	1.75	2.06	1.65
	16	20.37	$1.43^{(4,5)}$	1.63	1.18	1.32	1.44	1.37	1.17	1.49	2.09	1.83
	20	16.12	$1.52^{(5,6)}$	1.45	1.15	1.29	1.46	1.39	1.12	1.39	2.04	1.95
	24	13.35	$1.57^{(5,6)}$	1.21	1.18	1.30	1.46	1.44	1.13	1.36	1.97	1.99
	28	11.60	$1.57^{(5,6)}$	1.17	1.18	1.32	1.47	1.50	1.14	1.37	1.96	2.06
	32	10.06	$1.58^{(5,7)}$	0.99	1.16	1.29	1.45	1.48	1.10	1.33	1.89	2.07

In particular, $F[v]$ contains a linked list of the values i marking the pattern x and the beginning position of the pattern in the text. While looking for occurrences on $y[j\,..\,j+q-1]$, if $F[v]$ contains the value i, this indicates the pattern x may potentially begin at position $(j-i)$ of the text. In such a case, a matching test is performed between x and $y[j-i\,..\,j-i+m-1]$ via a character-by-character inspection.

The total number of filtering operations is exactly $n/(m-q)$. At each attempt, the maximum number of verification requests is $(m-q)$, since the filter provides information about that number of appropriate alignments of the patterns. On the other hand, if the computed fingerprint points to an empty location in F, then there is obviously no need for verification. Using a brute-force checking approach, the verification cost for a pattern x of length m is $\mathcal{O}(m)$. Hence, in the worst case, the time complexity of the verification is $\mathcal{O}(m(m-q))$, which is actually reached when all alignments in x must be verified at any possible beginning position. Hence, the best case complexity is $\mathcal{O}(n/(m-q))$, whereas the worst-case complexity is $\mathcal{O}(nm)$.

5. Experimental evaluations

In this section, we present experimental results in order to evaluate the performances of the new filter-based algorithms presented in this paper. In particular, the following algorithms have been tested:

- the filter approach of Chhabra and Tarhio [7];

ARTICLE IN PRESS

Table 2
Experimental results on a PERIOD-δ short integer sequence.

δ	m	FCT	SkSop	SSE	NR2	NR3	NR4	NR5	NR6	NO2	NO3	NO4
5	8	41.08	$0.83^{(3,4)}$	1.10	0.99	1.05	0.88	0.79	0.90	0.88	0.73	0.60
	12	36.42	$0.88^{(4,6)}$	1.03	1.06	1.02	0.94	0.86	0.91	0.81	0.67	0.69
	16	34.03	$0.90^{(4,6)}$	0.97	1.04	0.86	0.78	0.74	1.00	0.77	0.64	0.60
	20	35.31	$0.97^{(4,7)}$	0.90	0.98	0.89	0.88	0.84	0.92	0.73	0.60	0.55
	24	37.90	$1.05^{(4,7)}$	0.82	1.34	1.33	1.30	1.18	1.15	0.99	0.82	0.76
	28	36.26	$1.11^{(4,7)}$	0.75	1.17	1.09	1.10	1.04	0.97	0.78	0.64	0.56
	32	35.38	$1.16^{(4,8)}$	0.69	1.10	1.15	1.05	0.95	0.94	0.82	0.65	0.59
20	8	42.35	$0.93^{(3,4)}$	1.12	0.98	1.18	0.91	0.81	0.89	1.02	0.83	0.68
	12	39.09	$0.97^{(4,5)}$	1.05	1.11	1.14	1.06	0.98	1.00	1.02	0.88	0.93
	16	34.25	$1.05^{(4,6)}$	1.01	1.11	1.01	1.02	1.01	1.08	0.96	0.87	0.87
	20	35.41	$1.11^{(4,6)}$	0.96	1.10	1.09	1.21	1.21	1.07	0.97	0.89	0.89
	24	35.15	$1.18^{(3,7)}$	0.89	1.31	1.51	1.67	1.60	1.14	1.15	1.10	1.18
	28	32.23	$1.24^{(3,7)}$	0.81	1.23	1.40	1.56	1.36	1.07	1.04	1.08	1.15
	32	30.34	$1.36^{(3,7)}$	0.75	1.43	1.60	1.53	1.43	1.22	1.19	1.11	1.07
40	8	45.07	$1.10^{(3,4)}$	1.15	0.93	1.18	0.94	0.81	0.89	1.12	0.91	0.78
	12	37.91	$1.14^{(4,5)}$	1.08	1.08	1.12	1.03	0.93	1.03	1.13	1.03	1.08
	16	32.41	$1.19^{(4,5)}$	1.04	1.11	1.04	1.06	1.13	1.07	1.07	1.02	1.10
	20	28.63	$1.22^{(4,6)}$	1.00	1.05	1.09	1.24	1.35	1.08	1.04	1.04	1.15
	24	27.25	$1.28^{(4,6)}$	0.97	1.18	1.39	1.59	1.53	1.10	1.12	1.14	1.40
	28	24.91	$1.32^{(4,6)}$	0.95	1.20	1.51	1.67	1.41	1.05	1.17	1.30	1.50
	32	23.63	$1.38^{(4,6)}$	0.91	1.39	1.63	1.55	1.31	1.20	1.27	1.41	1.41

- our filtering solution NRq based on the Neighborhood Ranking approach, for $2 \leqslant q \leqslant 6$;
- our filtering solution NOq based on the Neighborhood Ordering approach, for $2 \leqslant q \leqslant 4$;
- the SkSop(k, q) algorithm [4], using k hash functions and q-grams, for $1 \leqslant k \leqslant 5$ and $3 \leqslant q \leqslant 8$;
- the algorithm based on SSE instructions presented in [6].

As in the experimental evaluations in [7] and in [6], in all cases in our evaluations we used the SBNDM2 algorithm [13]. In our dataset we use the following acronyms to identify the tested algorithms:

- FCT, for the SBNDM2 based algorithm by Chhabra and Jorma Tarhio [7];
- NRq, for the SBNDM2 algorithm based on the NR approach (Section 3.1); and
- NOq, for the SBNDM2 algorithm based on the NO approach (Section 3.2).

All algorithms have been evaluated in terms of their efficiency, i.e., of their running times. In particular, for the FCT algorithm, the average running times (in milliseconds) are reported. Instead, for the remaining algorithms, the speed-up of the running times obtained when compared with the time used by the FCT algorithm is reported. In the case of the SkSop(k, q) algorithm, only the best speed-up among all different implementations is reported, indicating in brackets the best values of k and q.

Please cite this article in press as: D. Cantone, et al., The order-preserving pattern matching problem in practice, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.10.023.

ARTICLE IN PRESS

We tested our solutions on sequences of short integer values (where each element is an integer in the range [0 .. 256]), long integer values (where each element is an integer in the range [0 .. 10.000]), and floating point values (where each element is a floating point in the range [0.0, 10000.99]). However, we did not observe sensible differences in the results; thus, for brevity, in the following tables only the results obtained on short integer sequences are reported. All texts have length 10^6. In particular, we tested our algorithm on two sequences: a Rand-δ sequence of random integer values varying around a fixed mean equal to 100 with a variability of δ; a Period-δ sequence of random integer values varying around a periodic function with a period of 10 elements and a variability of δ.

For each text in the set, we randomly selected 100 patterns extracted from the text and computed the average running time over 100 runs. We also computed the average number of false positives detected by the algorithms during the search. All algorithms have been implemented in the C programming language and have been compiled on a MacBook Pro with 8Gb Ram using the gcc compiler Apple LLVM version 5.1 (based on LLVM 3.4svn). Compilations have been carried out with the -O3 optimization option. In Tables 1 and 2, running times are expressed in milliseconds and the best results have been underlined.

Experimental tests on the Rand-δ numeric sequences have been carried out with values of $\delta = 5, 20$, and 40 (see Table 1). The results show that 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 between 1.15 and 1.50. In addition, we can observe that in all cases, the best speed-up achieved by the new algorithms is greater than that achieved by the SkSop and SSE algorithms. For the sake of completeness, we mention also that in more than 90% of the cases the gain in the number of detected false positives lies between 90% and 100%.

Experimental tests on the Period-δ problem have been carried out on a periodic sequence with a period equal to 10 and with $\delta = 5$, 20 and 40 (see Table 2). The results show that 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 when patterns are short. The no approach is always less efficient than the Fct algorithm. When the size of δ 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 solution with a speed up close to 1.7 in the case of long patterns. Also in these cases, the best speed-up achieved by the new algorithms is greater than that achieved by the SkSop and SSE algorithms.

References

[1] R. Baeza-Yates, G.H. Gonnet, A new approach to text searching, Commun. ACM 35 (10) (1992) 74–82, http://doi.acm.org/101145/135239135243.
[2] D. Belazzougui, A. Pierrot, M. Raffinot, S. Vialette, Single and multiple consecutive permutation motif search, in: L. Cai, S. Cheng, T.W. Lam (Eds.), Algorithms and Computation—24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings, in: Lecture Notes in Computer Science, vol. 8283, Springer, 2013, pp. 66–77.
[3] R.S. Boyer, J.S. Moore, A fast string searching algorithm, Commun. ACM 20 (10) (1977) 762–772.
[4] D. Cantone, S. Faro, M.O. Külekci, An efficient skip-search approach to the order-preserving pattern matching problem, in: Holub and Zdárek [14], pp. 22–35.
[5] C. Charras, T. Lecroq, J.D. Pehoushek, A very fast string matching algorithm for small alphabeths and long patterns (extended abstract), in: Combinatorial Pattern Matching, 9th Annual Symposium, CPM 98, Piscataway, New Jersey, USA, July 20-22, 1998, Proceedings, 1998, pp. 55–64. http://dx.doi.org/10.1007/BFb0030780.
[6] T. Chhabra, M.O. Külekci, J. Tarhio, Alternative algorithms for order-preserving matching, in: Holub and Zdárek [14], pp. 36–46.
[7] T. Chhabra, J. Tarhio, Order-preserving matching with filtration, in: J. Gudmundsson, J. Katajainen (Eds.), Experimental Algorithms—13th International Symposium, SEA 2014, Copenhagen, Denmark, June 29–July 1, 2014 Proceedings, in: Lecture Notes in Computer Science, vol. 8504, Springer, 2014, pp. 307–314.
[8] T. Chhabra, J. Tarhio, A filtration method for order-preserving matching, Inf. Process. Lett. 116 (2) (2016) 71–74.
[9] S. Cho, J.C. Na, K. Park, J.S. Sim, Fast order-preserving pattern matching, in: P. Widmayer, Y. Xu, B. Zhu (Eds.), Combinatorial Optimization and Applications—7th International Conference, COCOA 2013, Chengdu, China, December 12-14, 2013, Proceedings, in: Lecture Notes in Computer Science, vol. 8287, Springer, 2013, pp. 295–305.
[10] S. Faro, M.O. Külekci, Fast packed string matching for short patterns, in: Proceedings of the 15th Meeting on Algorithm Engineering and Experiments, 2013, pp. 113–121.
[11] S. Faro, M.O. Külekci, Fast and flexible packed string matching, J. Discrete Algorithms vol. 28 (2014) 61–72.
[12] S. Faro, M.O. Külekci, Efficient algorithms for the order preserving pattern matching problem, in: Proceedings of AAIM 2016, in: Lecture Notes in Computer Science, vol. 9778, 2016, pp. 185–196.
[13] J. Holub, B. Durian, 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.
[14] J. Holub, J. Zdárek (Eds.), Proceedings of the Prague Stringology Conference 2015, Prague, Czech Republic, August 24-26, 2015, Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2015.
[15] Intel, Intel (R) 64 and IA-32 Architectures Optimization Reference Manual, Intel Corporation, 2011.
[16] J. Kim, P. Eades, R. Fleischer, S. Hong, C.S. Iliopoulos, K. Park, S.J. Puglisi, T. Tokuyama, Order-preserving matching, Theoret. Comput. Sci. 525 (2014) 68–79.
[17] D.E. Knuth, J.H. Morris Jr, V.R. Pratt, Fast pattern matching in strings, SIAM J. Comput. 6 (1) (1977) 323–350.
[18] M. Kubica, T. Kulczynski, J. Radoszewski, W. Rytter, T. Walen, A linear time algorithm for consecutive permutation pattern matching, Inf. Process. Lett. 113 (12) (2013) 430–433.

Please cite this article in press as: D. Cantone, et al., The order-preserving pattern matching problem in practice, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.10.023.