Shape-Preserving Pattern Matching*

Domenico Cantone[†], Simone Faro[†] and M.Oguzhan Külekci[‡]

[†]Università di Catania, Viale A. Doria 6, 95125 Catania, Italy
{domenico.cantone,simone.faro}@unict.it

[‡]Istanbul Technical University
kulekci@itu.edu.tr

Abstract. Two sequences of integers x and z of the same length $m \geq 2$ are *shape-isomorphic* if, up to a positive proportional factor, the sequences of the distances between consecutive elements of x and z are the same, i.e., if for some $\rho > 0$ one has $x[i+1] - x[i] = \rho(z[i+1] - z[i])$, for each $1 \leq i \leq m-1$. In this paper we present two linear-time algorithms which, given a text y and a pattern x over an integer alphabet, finds all the factors of y that are shape-isomorphic to x. Our first solution is a two steps algorithm based on a reduction to the standard exact string matching problem, while our second solution is an online algorithm based on the well-known Knuth-Morris-Pratt string matching algorithm. We call this problem *shape-preserving pattern-matching problem*.

Keywords: Approximate text analysis, non-standard string matching, text processing.

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. Algorithmic solutions for it, both in its exact and approximation versions, are basic components of the implementations of practical softwares available under most operating systems.

Among the different approximation variants of the string matching problem, the *order-preserving pattern-matching problem* [12, 7, 6, 4, 10, 5, 3] (OPPM, for short) has recently gained attention. In this variant, the characters of the pattern and of the text are drawn from a linearly ordered alphabet Σ, so that each string z in $\Sigma^\star$ can naturally be mapped into the sequence of the ranks (within the ordered sequence of the characters occurring in z) of its characters, which we call *rank sequence*. For instance, if the alphabetic order is used for characters,

* Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0). This work was supported by the University of Catania "programma ricerca di ateneo UNICT 2020-22 linea 2"

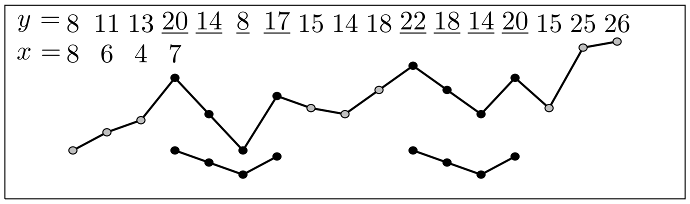

Fig. 1. Example of a pattern x of length 4 over an integer alphabet with two shape preserving occurrences in a text y of length 17, at positions 3 and 10.

the rank sequence of the string $x =$ "*gdich*" is the sequence $\langle 3, 2, 5, 1, 4\rangle$, since g has rank 3 in x, d has rank 2 in x, and so on. Then, given a text y and a pattern x, the OPPM problem consists in finding all the *order-occurrences* of x in y, namely the factors of y that have the same rank sequence as x.

The first solution to the OPPM problem was presented by Kubica et al. [12] in 2013. They provided a $\mathcal{O}(n+m \log m)$ solution over generic ordered alphabets based on the Knuth-Morris-Pratt algorithm [13] and also a $\mathcal{O}(n)$ solution in the case of integer alphabets. In the same year, Cho et al. [7] showed how the Boyer-Moore approach [2] can be applied to the OPPM problem, and Belazzougui et al. [1] showed that the Aho-Corasik approach can be applied to the OPPM problem for searching a set of patterns. More recently, Chhabra and Tarhio [6] have proposed a more practical solution based on the filtration method.

However, for applications such as time series analysis, weather data analysis, music melody matching, etc., OPPM is not adequate, as order-occurrences do not retain enough information to catch up the shape features of interest. Consider for instance the sequences shown in Fig. 1, where y may represent the price variation of some goods throughout a period of time. In this context, the pattern $x = \langle 8, 6, 4, 7\rangle$ could be interpreted as follows: when the price decreases twice the same amount δ, then one expects a subsequent increase of the price of $\frac{3}{2}\delta$. Observe that the given pattern has two order-occurrences in y, the first one at position 4 and the second one at position 11. However, despite of the fact that x and $\langle 18, 12, 11, 13\rangle$ share the same relative order, the two sequences are far from being similar. On the other hand, the second occurrence, $\langle 22, 18, 14, 20\rangle$, perfectly catches up the features of the pattern x.

In this paper we shall consider a restricted variant of the OPPM problem, that we call *shape-preserving pattern-matching problem.*

We say that two non-constant strings x and z of the same length $m \geq 2$, over an integer alphabet $\Sigma = \{1, 2, \dots, \sigma\}$,[1] are *shape-isomorphic* if, for some positive factor $\rho > 0$, we have $x[i+1]-x[i] = \rho(z[i+1]-z[i])$, for $1 \leq i \leq m-1$. Concerning constant strings of the same length, we agree that they are shape-isomorphic with proportionality factor $\rho = 0$. Then, the shape-preserving pattern-matching

[1] For notational convenience, we shall restrict our presentation to integer alphabets of the form $\Sigma = \{1, 2, \dots, \sigma\}$ only. However, all our considerations can be immediately generalized to any finite linearly ordered alphabet, by just identifying each character with its position in the alphabet.

problem (SPPM, for short) is the problem of finding all the factors of a given text y that are shape-isomorphic to a given pattern x, where x and y are strings over an integer alphabet.

We observe that in many practical cases a fixed ratio of proportionality is rare, thus a more practical approach would be to consider an approximate version of the problem or allowing the ratios to belong to some bounded interval. However the restricted problem accounted in this paper is a first step towards a more suitable method for comparing two numeric strings.

Specifically we shall provide two linear-time algorithms for the SPPM problem, based on (1) a reduction to the exact string matching problem and (2) on a simple, yet non trivial, modification of the Knuth-Morris-Pratt algorithm.

The paper is organized as follows. In Section 2, after some preliminary notions, we formally define the concept of shape-isomorphism, and state some of its properties. Then in Section 3 we present a linear-time algorithm for the SPPM problem based on a simple reduction to the exact string matching problem, while in Section 4 we present a linear-time KMP based algorithm, proving also its correctness. We draw our conclusions in Section 5.

2 Preliminary Notions and Definitions

Let $\Sigma = \{1, 2, \ldots, \sigma\}$ be a finite integer alphabet of size σ. A string x over Σ is a sequence of elements in Σ. We denote by $|x|$ the length of x and by $x[i]$, for $1 \leq i \leq |x|$, the i-th element of x. In addition, for $1 \leq i \leq j \leq |x|$, we denote by $x[i \mathinner{..} j]$ the substring of x of length $(j - i + 1)$, starting with the element of x at position i and ending with the element of x at position j. By $x.y$ we denote the concatenation of x and y. We write Σ^+ for the collection of all nonnull finite strings over Σ.

Two sequences $x, y \in \Sigma^+$ of the same length are said to be *order-isomorphic* if their elements have the same relative order. More formally, we have:

Definition 1 (Order-Isomorphism). *Two sequences $x, y \in \Sigma^+$ of the same length are* order-isomorphic*, and we write $x \sim y$, if the following condition holds:*

$$x[i] \leq x[j] \quad \text{if and only if} \quad y[i] \leq y[j]\,, \quad \text{for } 1 \leq i, j \leq |x|\,.$$

It is immediate to check that order-isomorphism is an equivalence relation.

We say that two sequences $x, y \in \Sigma^+$ of the same length $m \geq 2$ are *shape-isomorphic* if, up to a positive factor, the sequences of the distances between consecutive elements of x and y are the same. We also agree that any two sequences of length 1 are always regarded as shape-isomorphic. More formally, we have:

Definition 2 (Shape-Isomorphism). *Two non-constant sequences $x, y \in \Sigma^+$ of the same length $m \geq 2$ are said to be* shape-isomorphic with proportionality factor $\rho > 0$ *(or, more simply,* ρ-isomorphic*), and we write $x \approx_\rho y$, if the following condition holds: $x[i+1] - x[i] = \rho(y[i+1] - y[i])$, for all $1 \leq i \leq m - 1$.*

If $x, y \in \Sigma^+$ are constant sequences of the same length $m \geq 1$, we agree that they are shape-isomorphic with proportionality factor $\rho = 0$, and write $x \approx_0 y$. We say that two sequences $x, y \in \Sigma^+$ are shape-isomorphic*, and write $x \approx y$, if they are ρ-isomorphic, for some factor $\rho \geq 0$.*

The following lemma lists some very elementary facts concerning shape- and order-isomorphism. In particular, it states that shape-isomorphism is a hereditary equivalence relation which is finer than order-isomorphism.

Lemma 1. *Let $x, y, z \in \Sigma^+$, where $\Sigma = \{1, 2, \dots, \sigma\}$, and let $\rho > 0$ and $\overline{\rho}_1, \overline{\rho}_2 \geq 0$. Then the following properties hold:*

(a) either $x \approx_0 x$ or $x \approx_1 x$
(b) if $x \approx_\rho y$, then $y \approx_{\frac{1}{\rho}} x$;
(c) if $x \approx_{\overline{\rho}_1} y$ and $y \approx_{\overline{\rho}_2} z$, then $x \approx_{\overline{\rho}_1 \overline{\rho}_2} z$;
(d) if $x \approx_{\overline{\rho}_1} y$, then either $x[i\,..\,j] \approx_{\overline{\rho}_1} y[i\,..\,j]$ or $x[i\,..\,j] \approx_0 y[i\,..\,j]$, for all $1 \leq i \leq j \leq |x| - 1$;
(e) $\approx$ is an equivalence relation over Σ^+;
(f) if $x \approx y$, then $x \sim y$, i.e., shape-isomorphism is finer than order-isomorphism;
(g) if $x \approx y$, then $x[i\,..\,j] \approx y[i\,..\,j]$, for all $1 \leq i \leq j \leq |x| - 1$, i.e., shape-isomorphism is hereditary on substrings.

By exploiting the characterization contained in the following straightforward lemma, one can easily test in linear time whether two given sequences of the same length are shape-isomorphic.

Lemma 2. *Let $x, z \in \Sigma^+$ be two sequences of the same length m. Then $x \approx z$ if and only if either x and z are both constant sequences, or $x \approx_\rho z$, where $\rho = \frac{x[i+1]-x[i]}{z[i+1]-z[i]}$, for any $1 \leq i \leq m-1$ such that $z[i] \neq z[i+1]$.*

2.1 The Shape-Preserving Pattern-Matching Problem

Next, we formally define the shape-preserving pattern-matching problem.

Definition 3 (Shape-Preserving Pattern-Matching Problem). *Let $x, y \in \Sigma^+$, where $\Sigma = \{1, 2, \dots, \sigma\}$, be sequences of length m and n, respectively, such that $m \leq n$. The* shape-preserving pattern-matching problem *consists in finding all* shape-occurrences *of x in y, namely all positions $1 \leq i \leq n - m + 1$ such that $y[i\,..\,i + m - 1] \approx x$. In this context, x is the* pattern *and y is the* text.

By Lemma 1(f), any algorithm for the OPPM problem can be used as a filter to locate all candidate shape-occurrences of a pattern x in a text y. When an order-occurrence of x is found in y, a $\mathcal{O}(|x|)$-time verification procedure based on Lemma 2 can be run to check whether such an order-occurrence is shape-isomorphic to x. Since, for a pattern of length m and a text of length n, the OPPM problem can be solved in time $\mathcal{O}(n)$, the algorithm just outlined for the SPPM problem will run in $\mathcal{O}(n + km)$-time, where k is the number of order-occurrences of x in y.

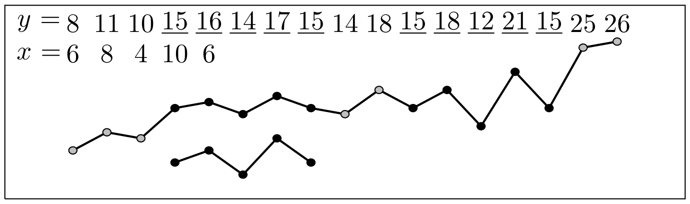
$y = 8\ 11\ 10\ \underline{15}\ \underline{16}\ \underline{14}\ \underline{17}\ \underline{15}\ 14\ 18\ \underline{15}\ \underline{18}\ \underline{12}\ \underline{21}\ \underline{15}\ 25\ 26$
$x = 6\ 8\ 4\ 10\ 6$

Fig. 2. A pattern x of length 5 with two shape-occurrences in a text y of length 17: at positions 4 and 11, with proportionality factors $\frac{1}{2}$ and $\frac{3}{2}$, respectively.

In the next sections we present two linear algorithms for the SPPM problem. Our first solution, presented in Section 3, is a two steps algorithm based on a reduction of the SPPM problem to the standard exact string matching problem. Our second solution, presented in Section 4, is a single step algorithm modeled after the well-known Knuth-Morris-Pratt algorithm, whose running time does not dependent on the number of order-occurrences.

3 Reducing SPPM to Exact Pattern Matching

In this section we present a first linear algorithms for solving the SPPM problem. Our solution is straightforward and assume that the proportionality ratio between the pattern and its occurrence in the text is a fixed constant $\rho \geq 0$. We observe that in this case the problem can be easily reduced, after some suitable transformations of the input strings, to the ordinary string matching problem. To begin we give some additional definitions.

Definition 4 (Delta Function). *The Delta Function $\delta()$ is the fucntion which associates a given input string x with the corresponding sequence of the differences between adjacent characters in x. Formally, for a given string $x \in \Sigma^*$, with $|x| = m$, we define $\delta(x)$ as the numeric sequence of length $|x| - 1$ such that, for $1 \leq i \leq m-1$, $\delta(x)[i] := x[i+1] - x[i]$.*

Definition 5 (Last Non-Zero Function). *The last non-zero function $\gamma()$ is the function which associates a given input string x with the sequence of the last non zero element up to each position of x. Formally, for a given string $x \in \Sigma^*$, with $|x| = m$, we have that $\gamma(x)$ is a sequence of length $m-1$, such that, for $1 \leq i \leq m-1$,*

$$\gamma(x)[i] := x[j] \text{ where } j = \max(\{1 \leq h < i+1 \ : \ x[h] \neq 0\} \cup \{0\})$$

Observe that $\gamma(x)[1] = 0$ if and only if $x[1] = 0$.

Definition 6 (Ratios Function). *The ratios-function $\psi()$ is the function which associates a given input string x with the sequence of the ratios between its (almost) adjacent characters. Formally, for a string $x \in \Sigma^*$ of length $|x| = m$ we have, for $1 \leq i \leq m-1$*

$$\psi(x)[i] := \frac{x[i+1]}{\gamma(x)[i]}$$

Our algorithm is based on a reduction of the SPPM problem to the exact string matching problem. Such reduction is inspired by the following straightforward technical lemmas. Specifically the following elementary Lemma 3 describes how to compute shape-isomorphic occurrences for a constant string, while Lemma 5 describes how to compute shape-isomorphic occurrences for a non-constant string.

Lemma 3. *Let x be a constant string of length m and let y be a text of length n, both strings over the same alphabet Σ. Then we have that x has a shape-isomorphic occurrence at position i of the text, i.e. $x \approx y[i..i+m-1]$, if and only if $\delta(x)[j] = \delta(y)[i+j]$ for $1 \leq j \leq m-1$.*

Lemma 4. *Let x and y be two non constant strings over the same alphabet Σ, with $|x| = |y| = m$ and such that $x \approx_\rho y$ with a proportionality factor $\rho > 0$. Then we have that $\gamma(\delta(x))[i] = \rho \cdot \gamma(\delta(y))[i]$, for all $1 \leq i < m-1$.*

Lemma 5. *Le x and y be two strings of length $m > 2$ over the same alphabet Σ, and assume $x[1] \neq x[2]$ and $y[1] \neq y[2]$. Then $x \approx y$ if and only if $\psi(\delta(x)) = \psi(\delta(y))$ and $\delta(x)[1] \times \delta(y)[1] > 0$.*

Based on Lemma 5, Algorithm1 depicted in Fig.4 finds all shape-isomorphic occurrences of a given pattern x of length m in a given text y of length n. During the preprocessing phase the algorithm performs a partition of the pattern x into two strings, x_1 and x_2, where x_1 is the constant prefix of the pattern with maximal length, while x_2 is the suffix of the pattern with maximal length such that $x_2[1] \neq x_2[2]$.

More formally we compute an index k, such that

$$k = \max(\{j \ : \ j > 0 \text{ and } x[j] \neq x[j+1]\} \cup \{0\}),$$

and set $x_1 = x[1 \ldots k]$ and $x_2 = x[k \ldots m]$. The value of k can be computed in linear time in the size of x.

In a first step the algorithm computes all shape-isomorphic occurrences of x_1 in y. Since x_1 is a constant sequence all such occurrences are simply computed (based on Lemma 3) by running a linear exact string matching algorithm in order to search $\delta(y)$ for all occurrences of $\delta(x_1)$. We indicate with Γ_1 the set of such occurrences. In the event that $|x_1| = 1$ we skip this step and set $\Gamma_1 = \{i \ : \ 1 \leq i \leq n-m\}$.

In a second step the algorithm searches for all shape-isomorphic occurrences of x_2 in y by running (based on Lemma 5) a linear exact string matching algorithm in order to search $\gamma(\delta(y))$ for all occurrences of $\gamma(\delta(x_2))$. We indicate with Γ_2 the set of such occurrences. In the event that $|x_2| < 2$ we skip this step and set $\Gamma_2 = \{i \ : \ 1 \leq i \leq n-m\}$.

The last step of the algorithm the two sets of occurrences Γ_1 and Γ_2 are combined in order to find the set Γ of all positions i such that $x \approx y[i..i+m-1]$. We keep out from Γ_2 all values i such that $\delta(x)[1] \times \delta(y)[i] < 0$ (this is due to the condition $\rho >= 0$ in the shape-isomorphic Definition 2). Specifically we have $\Gamma = \{i-k+2 \ : \ i \in \Gamma_2 \text{ and } (i-k+2) \in \Gamma_1 \text{ and } \delta(x)[1] \times \delta(y[i]) > 0\}$.

```
ALGORITHM1($x, y$)
1.    $m \leftarrow |x|$;
2.    $k \leftarrow \min(\{j \ : \ 1 \leq j \leq m-1$ and $x[j] \neq x[j+1]\} \cup \{m\})$;
3.    $x_1 \leftarrow x[1..k]$;
4.    $x_2 \leftarrow x[k..m]$;
5.    $\mathit{\Gamma}_1 \leftarrow \mathit{\Gamma}_2 \leftarrow \{i \ : \ 1 \leq i \leq n-m\}$;
6.    if($|x_1| > 1$) then $\mathit{\Gamma}_1 \leftarrow$ occurrences of $\delta(x_1)$ in $\delta(y)$;
7.    if($|x_2| > 2$) then $\mathit{\Gamma}_2 \leftarrow$ occurrences of $\psi(\delta(x_2))$ in $\psi(\delta(y))$;
8.    for each $i \in \mathit{\Gamma}_2$ do;
9.        if($\delta(x)[1] \times \delta(y[i]) > 0$ and $(i-k+2) \in \mathit{\Gamma}_1$) then output($i-k+2$);
```

Fig. 3. A linear-time algorithm for the SPPM problem based on a reduction to the exact string matching problem.

If we use a linear worst case time exact string matching algorithm for the two steps, then it is trivial to observe that ALGORITHM1 achieves an $\mathcal{O}(n+m)$ worst case time complexity. Observe, however, that the last step of the algorithm depends on the number of occurrences of x_1 and x_2, which is always bounded by $\mathcal{O}(n)$.

4 A KMP based Algorithm for the SPPM Problem

The Knuth-Morris-Pratt algorithm [13] has been the first algorithm to achieve a linear worst-case time complexity for the exact pattern matching problem. It uses a *prefix table*, also called *border table* or *prefix function*, to carry the information which allows one to compute in constant time the length of the longest safe shift when a mismatch occurs or a match is found, thus keeping the number of character comparisons linear in the size of the text.

Much in the same way, for the SPPM problem of our interest we shall use a *shape-border table*, which associates to each position i of a given pattern x the length of the longest proper suffix of $x[1\,..\,i]$ that is shape-isomorphic to a prefix of $x[1\,..\,i]$.

The *shape-border table* for a finite integer sequence x is defined as follows.

Definition 7 (Shape-Border Table). *Let x be an integer sequence of length $m \geq 1$. The* shape-border table *for x is the map $\pi_x \colon \{0, 1, \ldots, m\} \to \mathbb{N}$ defined by*

$$\pi_x[i] \ =_{\scriptscriptstyle Def} \max\Big(\big\{j \ : \ 1 \leq j < i \ \textit{and} \ x[1\,..\,j] \approx x[i-j+1\,..\,i]\big\} \cup \{0\}\Big)\,.$$

Plainly, for $2 \leq i \leq m$, we have $1 \leq \pi_x[i] < i$ and $x\big[1\,..\,\pi_x[i]\big] \approx x\big[i-\pi_x[i]+1\,..\,i\big]$. The latter relationship allows us to readily define a related *proportionality factor map* $\varrho_x \colon \{2, 3, \ldots, m\} \to \mathbb{N}$ such that, for $2 \leq i \leq m$,

$$x\big[1\,..\,\pi_x[i]\big] \approx_{\varrho_x[i]} x\big[i - \pi_x[i] + 1\,..\,i\big]. \qquad (1)$$

When the sequence x is understood from the context, we shall simply write π and ϱ in place of π_x and ϱ_x.

Using the notation $(\cdot)^j$ for map iteration,[2] by induction (1) generalizes to

$$x\big[1\,..\,\pi^j[i]\big] \approx_{\varrho[\pi^{j-1}[i]]} x\big[i-\pi^j[i]+1\,..\,i\big], \tag{2}$$

for all $j \geq 1$ such that $\pi^{j-1}[i] \geq 2$. We also notice that since $\pi[i] < i$ (for $2 \leq i \leq m$) and $\pi[1] = \pi[0] = 0$, any sequence of the form $\big\langle \pi^1[i], \pi^2[i], \pi^3[i], \dots \big\rangle$ starts with a (possibly empty) strictly decreasing substring of positive integers and then continues indefinitely with a sequence of 0s.

Next we show that the shape-border table π satisfies the following recursive relation, for $1 \leq i < m$, which, together with (2), will yield to a linear algorithm for computing π:

$$\begin{cases} \pi[0] = \pi[1] = 0 \\ \pi[i+1] = \max\limits_{j\geq 1}\Big\{\pi^j[i]+1 \,:\, x\big[1\,..\,\pi^j[i]+1\big] \approx x\big[i-\pi^j[i]+1\,..\,i+1\big]\Big\}. \end{cases} \tag{3}$$

Let $1 \leq i < m$ and let

$$A_i =_{\text{Def}} \big\{\pi^j[i]+1 \,:\, j \geq 1 \text{ and } x\big[1\,..\,\pi^j[i]+1\big] \approx x\big[i-\pi^j[i]+1\,..\,i+1\big]\big\}.$$

By the very definition of π, it follows easily that $\max A_i \leq \pi[i+1] \leq \pi[i]+1$. Thus, to prove the correctness of (3), it is enough to show that $\pi[i+1] \in A_i$, which we do as follows. If $\pi[i+1] = 1$, then we plainly have $\pi[i+1] \in A_i$. Thus, let us assume that $\pi[i+1] \geq 2$. If $\pi[i+1] = \pi[i]+1$, we are done; otherwise, let $\overline{j}$ be the largest index $j \geq 1$ such that $\pi[i+1]-1 < \pi^j[i]$, so that

$$\pi^{\overline{j}+1}[i] \leq \pi[i+1]-1 < \pi^{\overline{j}}[i]. \tag{4}$$

By (2) $x\big[1\,..\,\pi^{\overline{j}}[i]\big] \approx x\big[i-\pi^j[i]+1\,..\,i\big]$, and since $x\big[1\,..\,\pi[i+1]-1\big] \approx x\big[i-\pi[i+1]+2\,..\,i\big]$ and we have $\pi[i+1]-1 < \pi^{\overline{j}}[i]$, by the hereditarity and transitivity of shape-isomorphism it follows that $x[1\,..\,\pi[i+1]-1]$ is shape-isomorphic to a proper suffix of $x[1\,..\,\pi^{\overline{j}}[i]]$. Hence, $\pi[i+1]-1 \leq \pi^{\overline{j}+1}[i]$ which, by (4), implies $\pi[i+1] = \pi^{\overline{j}+1}[i]+1$, proving that $\pi[i+1] \in A_i$, and in turn completing the proof of correctness of (3).

From (3), $\pi[i+1] = \pi^{j_i}[i]+1 \leq \pi[i]-j_i+2$, where j_i is the smallest $j \geq 1$ such that $x\big[1\,..\,\pi^j[i]+1\big] \approx x\big[i-\pi^j[i]+1\,..\,i+1\big]$. Thus, to compute $\pi[i+1]$, it is enough to test the latter condition j_i times, yielding a linear number of tests in $m = |x|$, since

$$\sum_{i=1}^{m-1} j_i \leq \sum_{i=1}^{m-1} \big(\pi[i]-\pi[i+1]+2\big) = \pi[1]-\pi[m]+2m-2 \leq m-3.$$

[2] We recall that the operator $(\cdot)^j$ for map iteration is defined as follows. Given a map $f : A \to A$, for $x \in A$ we put: $f^0(x) = x$ and, recursively, $f^{j+1}(x) = f(f^j(x))$.

i	1	2	3	4	5	6	7	8
$x[i]$	4	2	10	6	22	14	13	17
$\pi[i]$	0	1	1	2	3	4	2	3
$\varrho[i]$	-	0	0	2	2	2	0.5	0.5

Table 1. The shape-border table and the proportionality factor map for the pattern $x = \langle 4, 2, 10, 6, 22, 14, 13, 17\rangle$.

```
Compute-Shape-Border-Table($x$)
1.   $\pi[1] \leftarrow 0$;   $\pi[2] \leftarrow 1$;   $\varrho[2] \leftarrow 0$;
2.   for $i \leftarrow 3$ to $|x|$ do
3.       $k \leftarrow i - 1$;
4.       repeat
5.           $t \leftarrow \pi[k]$;
6.           $\varrho[i] \leftarrow \textsc{ExtShapeIso}\big(x[1 \mathinner{..} t], x[i-t \mathinner{..} i-1], x[t+1], x[i], \varrho[k]\big)$;
7.           $k \leftarrow \pi[k]$;
8.       until $\varrho[i] \geq 0$;
9.       $\pi[i] \leftarrow t + 1$;
10.  return $\pi, \varrho$;

ExtShapeIso($x, y, a, b, \rho$)
1.   $m \leftarrow |x|$;
2.   return if $\rho = 0$ and $x[m] = a$ and $y[m] = b$ then $0$
         elseif $\rho = 0$ and $(a - x[m]) \cdot (b - y[m]) > 0$ then $\frac{a-x[m]}{b-y[m]}$
         elseif $\rho > 0$ and $(a - x[m]) = \rho \cdot (b - y[m])$ then $\rho$
         else $-1$;
```

Fig. 4. The procedure Compute-Shape-Border-Table for computing the shape-border table and the proportional factor map for a pattern x and its subroutine ExtShapeIso.

If the values of the proportionality factor map ϱ are also tabulated during the computation of the shape-border table π, it is possible to make each of the tests

$$x\big[1 \mathinner{..} \pi^j[i] + 1\big] \approx x\big[i - \pi^j[i] + 1 \mathinner{..} i + 1\big], \tag{5}$$

for $j = 1, 2, \ldots, j_i$, in constant time by means of the following procedure call

$$\textsc{ExtShapeIso}\big(x\big[1 \mathinner{..} \pi^j[i]\big], x\big[i - \pi^j[i] + 1 \mathinner{..} i\big], x\big[\pi^j[i] + 1\big], x[i+1], \varrho[\pi^{j-1}[i]]\big)$$

(cf. Fig. 4). If the condition (5) is true, then such a call will return the proportionality factor $\varrho[i+1]$ of (5), otherwise it will return the value -1.

The above considerations leads to the $\mathcal{O}(m)$-time procedure Compute-Shape-Border-Table reported in Fig. 4 for computing the shape-border table and the proportional factor map for a pattern x of length m. Table 1 reports the shape-border table and the proportionality factor map for the pattern $x = \langle 4, 2, 10, 6, 22, 14, 13, 17\rangle$.

Much in the style of the Knuth-Morris-Pratt algorithm, we show next how to find all shape-occurrences of a pattern x (of length m) in a text y (of length n, with $n \geq m$) in time $\mathcal{O}(n)$, using the shape-border table and the proportionality factor map for x. The complete algorithm, called Algorithm2, is reported in Fig. 5. Line references in the following discussion are intended to point out to its pseudo-code, even if not explicitly stated.

Let us assume that it is known that $x[1\,..\,j] \approx_\rho y[i\,..\,i+j-1]$ holds, for some $\rho \geq 0$, $i \leq n-m$, and $j \leq m$, and that no further progress in the search for a shape-occurrence of x at position i in y is possible. This means that either $j = m$, in which case a shape-occurrence of x at position i in y has been found (cf. line 9), or $j < m$ and $x[1\,..\,j+1] \not\approx y[i\,..\,i+j]$. In any case, one needs to examine a new position $i' > i$ to find a (new) shape-occurrence of x. We claim that if $\ell > i$ is any position in y of a shape-occurrence of x, then $\ell \geq i+j-\pi[j]$, so that it is safe to move to position $i' = i+j-\pi[j]$ (cf. line 11). Indeed, if $\ell < i+j-\pi[j]$, then $x[1\,..\,i+j-\ell] \approx y[\ell\,..\,i+j-1]$, so that, by the hereditarity and transitivity of shape-isomorphism, we would have $x[1\,..\,i+j-\ell] \approx x[\ell-i+1\,..\,j]$. Hence, by the very definition of $\pi[j]$, it would follow that $i+j-\ell \leq \pi[j]$, a contradiction.

Notice that once we move to position $i' = i+j-\pi[j]$ in y, we already know that $x[1\,..\,\pi[j]] \approx y[i+j-\pi[j]\,..\,i+j-1]$ (for some proportionality factor; see below). Hence, the matching phase relative to position i' does not need to reconsider again the positions from 1 to $\pi[j]$ of the pattern x and related positions from i' to $i'+\pi[j]-1$ of the text y (cf. line 11). However, to execute efficiently the matching phase by repeated calls to the procedure ExtShapeIso in Fig. 4 (cf. line 5), at each step (even at the first one) one needs to know the proportionality factor of the shape-isomorphic substrings which have been matched so far. From the initial assumption $x[1\,..\,j] \approx_\rho y[i\,..\,i+j-1]$, we can infer that $x[1\,..\,\pi[j]] \approx_\rho y[i+j-\pi[j]\,..\,i+j-1]$ holds only when $x[1\,..\,\pi[j]]$ is not a constant sequence. Otherwise, we would have $x[1\,..\,\pi[j]] \approx_0 y[i+j-\pi[j]\,..\,i+j-1]$. Plainly, $x[1\,..\,\pi[j]]$ is a constant sequence if and only if $\varrho[j] = 0$ (cf. line 10). Finally, we point out that the procedure ExtShapeIso, called at line 5, not only allows one to make progress in the matching phase, but it also takes care of updating, if needed, the proportionality factor ρ.

The above discussion highlighted the key points needed in a more formal proof of the correctness of the Algorithm2 in Fig. 5 for the SPPM problem.

Complexity issues

Next we show that the overall time complexity of the while-loop at lines 3–11 of Algorithm2 is $\mathcal{O}(n)$, where, as usual, n is the size of the text. Since, as already shown, the call to procedure Compute-Shape-Border-Table at line 1 takes $\mathcal{O}(m)$ time, where, again, m is the size of the pattern, and since $m \leq n$, it will follow that the total time complexity of Algorithm2 is $\mathcal{O}(n)$.

Plainly, the time complexity of the while-loop at lines 3–11 is dominated by the number N of calls to procedure ExtShapeIso at line 5 (each of which takes constant time). Let us associate to each such a call

$$\text{ExtShapeIso}\big(x[1\,..\,j], y[i\,..\,i+j-1], x[j+1], y[i+j], \rho\big),$$

```
ALGORITHM2$(x, y)$
1.  $(\pi, \varrho) \leftarrow$ COMPUTE-SHAPE-BORDER-TABLE$(x)$
2.  $i \leftarrow 1$;   $j \leftarrow 1$;   $\rho \leftarrow 0$;
3.  while $i \leq |y| - |x|$ do
4.      repeat
5.          $\rho' \leftarrow$ EXTSHAPEISO$(x[1\,..\,j], y[i\,..\,i+j-1], x[j+1], y[i+j], \rho)$;
6.          if $\rho' \geq 0$ then
7.              $j \leftarrow j+1$;   $\rho \leftarrow \rho'$;
8.      until $j = |x|$ or $\rho' = -1$;
9.      if $j = |x|$ then write $i$;  //a shape-occurrence has been found
10.     if $\varrho[j] = 0$ then  $\rho \leftarrow 0$;
11.     $i \leftarrow i + j - \pi[j]$;   $j \leftarrow \pi[j]$;
```

Fig. 5. A linear-time KMP based algorithm for the SPPM problem.

the pair (i, j) of the values of the parameters i and j when the call is made, and form their sequence

$$(i_1, j_1),\ (i_2, j_2), \ldots,\ (i_N, j_N), \tag{6}$$

following the same ordering of the calls (so that $(i_1, j_1) = (1, 1)$).

By a simple inspection of the pseudo-code of ALGORITHM2, it is easy to see that the sequence (6) enjoys the following two properties:
(a) $i_1 + j_1 \leq i_2 + j_2 \leq \cdots \leq i_N + j_N \leq n$; (b) $j_\ell \geq 1$, for all $1 \leq \ell \leq N$.

To ease presentation, let us refer to any pair $\langle(i, j), (i', j')\rangle$ of consecutive pairs (i, j), (i', j') in (6) as a *transition*, and distinguish between *increasing transitions*, when $j' = j + 1$, and *decreasing transitions*, when $j' < j$. Let I and D be, respectively, the number of increasing and of decreasing transitions in (6). We plainly have $N = I + D + 1$, as any transition is either increasing or decreasing. In addition, since $j_1 = 1$, we have $D \leq I$, so that $N \leq 2D+1$. Finally, for any increasing transition $\langle(i, j), (i', j')\rangle$, we have $2 \leq i + j < i' + j' \leq n$, so that $I \leq n - 2$. From the latter inequality, we obtain $N \leq 2n - 3$, yielding the linear bound $\mathcal{O}(n)$ seeked for for the time complexity of our ALGORITHM2.

5 Conclusions

In this paper we introduced a restricted variant of the Order Preserving Pattern Matching problem, called Shape Preserving Pattern Matching. Specifically we say that two non-constant strings x and z of the same length $m \geq 2$ are shape-isomorphic if, for some positive factor $\rho > 0$, we have $x[i+1] - x[i] = \rho(z[i+1) - z[i])$, for $1 \leq i \leq m-1$. In most practical applications this restricted variant turns out to be more effective then the original problem. We also provided two linear-time algorithms for such problem, based on a simple, yet non trivial, modification of the Knuth-Morris-Pratt algorithm.

Although a fixed ratio between two sequences is rare in practice, this paper presents a first step towards a more suitable way to compare two numeric sequences. A more practical approach would be considering a k-approximate version of the problem or allowing the ratios to belong to some bounded interval. Moreover, we are also interested to extend the dependency of differences in the occurrence from the differences in the pattern not only for a linear function but also for an arbitrary function.

References

1. Belazzougui, D., Pierrot, A., Raffinot, M., Vialette, S.: Single and Multiple Consecutive Permutation Motif Search, In Proc. of ISAAC 2013, LNCS, vol. 8283, 66–77. Springer (2013)
2. Boyer, R.S., Moore, J.S.: A fast string searching algorithm. Communications of the ACM 20(10), 762–772 (1977)
3. Cantone, D. Faro, S., Külekci, M.O., The order-preserving pattern matching problem in practice, Discret. Appl. Math., vol.274, pp.11–25 (2020)
4. Cantone, D., Faro, S., Külekci, M.O., An Efficient Skip-Search Approach to the Order-Preserving Pattern Matching Problem, in Proc. of Stringology 2015, pp. 22-35 (2015)
5. Chhabra, T., Faro, S., Külekci, M.O., Tarhio, J., Engineering order-preserving pattern matching with SIMD parallelism, Softw. Pract. Exp., vol.47 (5), pp. 731–739 (2017)
6. Chhabra, T., Tarhio, J.: Order-preserving matching with filtration. In: Proc. SEA '14, 13th International Symposium on Experimental Algorithms. LNCS, vol. 8504, 307–314. Springer (2014)
7. Cho, S., Na, J.C., Park, K., Sim, J.S.: Fast order-preserving pattern matching. In: Widmayer, P., Xu, Y., Zhu, B. (eds.) COCOA 2013. LNCS, vol. 8287, 295–305. Springer (2013)
8. 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, 84–95. Springer (2013)
9. Durian, B., Holub, J., Peltola, H., Tarhio, J.: Improving practical exact string matching. Information Processing Letters 110(4): 148–152 (2010)
10. Faro, S., Külekci, M.O., Efficient Algorithms for the Order Preserving Pattern Matching Problem, In Proc. of Algorithmic Aspects in Information and Management - 11th International Conference, AAIM 2016, Lecture Notes in Computer Science, vol.9778, pp.185–196, Springer (2016)
11. 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)
12. 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)
13. Knuth, D.E., Morris, J.M., Pratt, V.R.: Fast pattern matching in strings. SIAM Journal on Computing 6(2), 323–350 (1977)
14. 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