ELSEVIER

Theoretical Computer Science 163 (1996) 117–144

Theoretical Computer Science

How the character comparison order shapes the shift function of on-line pattern matching algorithms

Livio Colussi, Laura Toniolo *

Università di Padova, Dipartimento di Matematica Pura ed Applicata, Via Belzoni 7, 35131 Padova, Italy

Received November 1994; revised June 1995
Communicated by G. Ausiello

Abstract

String matching is the problem of finding all occurrences of a string $\mathcal{W}[0\ldots m-1]$ of length m called a *pattern*, in a longer string $\mathcal{T}[0\ldots n-1]$ of length n called a *text*. Several string matching algorithms have been designed to solve the problem in linear time; most of them work in two steps, called pattern preprocessing and text search step.

The paper addresses the definition and computation of the shift function in the pattern preprocessing step of on-line string matching algorithms. The shift function depends essentially on the order the pattern characters are compared with the corresponding text characters.

We consider a family $\mathcal{F}$ of algorithms that do not change the character comparison order J during execution and we present a uniform definition of shift function δ_J for such algorithms via a function $imin_J$. The definition allows one to compute δ_J in $O(m \log\log m)$ time in the worst case, given $imin_J$, but sufficient conditions to compute δ_J in $O(m)$ time are provided. Computing $imin_J$ requires $O(m^2)$ comparisons in general. We introduce the class of compact orders (which is the generalization of Knuth–Morris–Pratt, Boyer–Moore and Crochemore–Perrin character comparison orders) and we give algorithms to compute both function $imin_J$ and shift function δ_J in $O(m)$ time for all compact orders.

We show that given the order J and the pattern $\mathcal{W}$ there exists a set C of equivalent orders such that the function $imin_K$ can be computed in linear time given $imin_J$ for all orders $K \in C$. Moreover, we characterize two orders in the set C that respectively minimize and maximize the values of the shift function and we show that for both those orders the shift function can be computed in linear time given $imin_J$.

1. Introduction

String matching is the problem of finding all occurrences of a string $\mathcal{W}[0\ldots m-1]$ of length m called a *pattern*, in a longer string $\mathcal{T}[0\ldots n-1]$ of length n called a *text*.

* Corresponding author. E-mail: {colussi, laura}@euler.math.unipd.it.

SSDI 0304-3975(95)00137-9

A naive algorithm to solve the problem considers each text position as a potential occurrence of the pattern, compares corresponding symbols from left to right and finally shifts the pattern along the text of one position as soon as a mismatch is encountered. Clearly such an algorithm takes $O(mn)$ time in the worst case (think of $\mathscr{T} = a^n$, $\mathscr{W} = a^{m-1}b$).

Several string matching algorithms have been designed to solve the problem in linear time. For a survey on the subject see Aho's paper [1]. From a theoretical point of view the main operation in string matching algorithms is considered that of comparing symbols; thus their complexity is often expressed by the number of character comparisons performed. Efficient implementations on a conventional machine are incidentally addressed or just sketched. Most of these algorithms work in two steps: in the first step some information about the pattern is computed, stored and used later in the second step or the text search step.

One of the best known string matching algorithms is that of Knuth et al. [23] (KMP algorithm for short) that in the worst case makes $2m - 4$ comparisons in the pattern preprocessing step and at most $2n - m$ comparisons in the text search step. We shall assume in the sequel that the reader is familiar with this algorithm.

In order to find all the occurrences of a pattern in a text, KMP algorithm aligns the pattern $\mathscr{W}[0 \ldots m-1]$ with the text $\mathscr{T}[0 \ldots n-1]$ at the leftend side of the strings and compares characters from left to right. Suppose that at the current situation a prefix $\mathscr{W}[0 \ldots i-1], 0 \leqslant i \leqslant m$ has been discovered in the text $\mathscr{T}$ starting at position l, i.e. $\mathscr{T}[l \ldots l+i-1] = \mathscr{W}[0 \ldots i-1]$; if a mismatch is found in comparing the ith pattern character ($0 \leqslant i < m$) with $\mathscr{T}[l+i]$, a shift of the pattern to the right follows. The shift can be defined as $shift_{\mathscr{W}}(i) = min\{j \geqslant 1 \mid j = i+1 \text{ or } (\mathscr{W}[j \ldots i-1] = \mathscr{W}[0 \ldots i-j-1] \text{ and } \mathscr{W}[i] \neq \mathscr{W}[i-j])\}$. (In order to deal uniformly the case of a whole occurrence ($i = m$) of the pattern in the text, it is customary to add a character $\mathscr{W}[m]$ different from all characters in $\mathscr{W}[0 \ldots m-1]$.) The information about the shifts is computed in the pattern preprocessing step of string matching algorithms. We refer to it as *Shift Function* δ.

KMP algorithm keeps comparing the character $\mathscr{T}[l+i]$ of the text (where the mismatch has been detected) with the pattern character aligned with it in the new position of the pattern ($\mathscr{W}[i - shift_{\mathscr{W}}(i)]$ will be the pattern character chosen after the first mismatch), until a match is found, or the pattern is shifted after position $l+i$ in the text.

In a first phase KMP algorithm compares the characters of the text with the characters of the pattern from left to right while they are found to be equal; then in a second phase, as soon as a mismatch is found in position $\mathscr{T}[j]$ of the text, it compares the same text character $\mathscr{T}[j]$ with a set of selected characters of the pattern in order from right to left. The cardinality of the set is at most $\log_{\Phi}(m+1)$ as shown in [24]. During this second phase the first position where the pattern and the text can be aligned matching corresponding symbols (up to position $\mathscr{T}[j]$ included) is computed. The KMP algorithm is strictly on-line since text characters are processed in order from left to right never reconsidering previous characters.

There are two ways to modify KMP algorithm. One can modify the second phase by choosing a different order to compare the text character $\mathcal{T}[j]$ with the set of selected pattern characters. This approach has been followed in Simon's algorithm [25] and in some extensions of the string matching problem [6, 21]. In [6] the authors consider the more general problem of computing the length of the longest prefix of $\mathcal{W}$ for each position in $\mathcal{T}$. They show how to choose the optimal order of character comparisons, achieving a tight bound of $\lfloor((2m-1)/m)n\rfloor$ comparisons in the text search step. Note that all strictly on-line algorithms that solve the string matching problem, solve also the more general problem of prefix-matching and therefore the results mentioned hold for the string matching problem as well.

A different way of modifying KMP algorithm consists in changing the order of comparing text characters with the aligned pattern characters in the first phase. The algorithms obtained by this approach are on-line in a wider sense since they need to have access to the text by a window of m characters (the ones aligned with the pattern in the current position).

The Boyer–Moore algorithm [5] (BM algorithm for short) can be considered as a first example of this approach since it compares text characters with the aligned pattern characters from right to left. It makes $O(m)$ comparisons in the pattern preprocessing step and about $3n$ comparisons in the text search step, as shown recently by Cole [9]. A variant of the BM algorithm that was designed by Apostolico and Giancarlo [3] achieves the $2n-m$ comparison bound in the text search step, with $O(m)$ time pattern preprocessing.

The approach to the string matching problem by on-line algorithms that can access the text by a window of size m allows to get better bounds in the text search step. Indeed, Crochemore and Perrin [16] presented a linear time, constant space string matching algorithm that takes at most $5m$ comparisons in the pattern preprocessing step and at most $2n-m$ comparisons in the text search step. Other algorithms [15, 17, 20] use constant space, but make more than $2n-m$ comparisons in the text search step.

Colussi [11] improved KMP algorithm to make at most $n+\frac{1}{2}(n-m)$ character comparisons in the text search step. The improvement is achieved by computing some more information in the pattern preprocessing step, still using at most $2m-4$ character comparisons. Colussi's algorithm has been furthermore improved by Galil and Giancarlo [19] to make at most $n+\frac{1}{3}(n-m)$ character comparisons in the text search step; Cole and Hariharan [10] discovered an algorithm that makes at most $n+[8/3(m+1)](n-m)$ character comparisons in the text search step. However, Cole and Hariharan's algorithm requires $O(m^2)$ pattern preprocessing time and it uses $O(m)$ space. Independently, Breslauer and Galil [8] gave a linear time algorithm that makes at most $n+\lceil((4\log m+2)/m)(n-m)\rceil$ character comparisons in the text search phase. The pattern preprocessing phase takes linear time and makes at most $2m$ character comparisons.

If we consider two of the most efficient string matching algorithms, namely Crochemore and Perrin's algorithm and Colussi's algorithm, we realize that they carefully choose how to compare characters as a result of some information gathered from the

structure of the pattern $\mathscr{W}$ (mainly from the periodicity structure of $\mathscr{W}$). This kind of computation in the preprocessing step of the algorithms allows to obtain "good" shift functions, i.e. shift functions that move the pattern along the text faster. From this point of view we are given a clear hint to look at the relations, if any, between character comparison orders and their shift functions. Another reason of interest in the shift functions can be linked to the following observation: all string matching algorithms mentioned above compute the information about their shift function δ independently, using different properties of strings and different techniques; but many string matching algorithms use $O(m)$ comparisons in the pattern preprocessing step. Was that a mere coincidence or were there relations between the shift functions not yet studied? Recall the basic idea underlying the definition of a shift: it tells us how many positions we are allowed to move the pattern along the text, without skipping any possible occurrence and satisfying all the constraints (i.e. matches) from previous iterations.

Thus the fundamental feature of the shift function δ is indeed that it depends on the *order* in which the pattern characters are compared with the corresponding text characters; from left to right in case of KMP algorithm, but in general any order is possible!

In this paper we study, for any given pattern, the relation between the order of comparing pattern characters with corresponding text characters and the relative shift function. We are interested in describing the different shift functions to highlight the most desirable ones from the point of view of time efficiency and/or maximization of their values.

Although studying the relations among the shift functions is clearly important, the approach to the problem has been limited so far to practical and experimental results, at least to these authors' knowledge. Researchers modified the shift functions in order to achieve a better speedup in the following text search step [4, 22, 26].

Our interest is instead that of a theoretical approach to the shift functions themselves related to the character comparison orders from which they derive. The goal is to abstract from specific algorithms and ad hoc techniques. Nevertheless we will show how to efficiently choose character comparison orders that maximize the values of the shift functions, therefore suggesting a useful heuristic for the text search phase of string matching algorithms. This heuristic has been successfully used in [11] where J_{Col} is the character comparison order that maximizes the shift function in the equivalence class of J_{KMP} (as defined in Section 4) and in [12] where the character comparison order used to improve BM algorithm is the one that maximizes the shift function in the equivalence class of J_{BM}.

The possible steps towards a whole understanding of the relations between the shift functions can be described by answering the following questions:

1. Is it possible to give a *uniform* definition of shift function for string matching algorithms?
2. Is there any characterization of character comparison orders such that the shift function can be computed in $O(m)$ time?

3. Are there sets of equivalent orders, namely orders such that the computation of the shift function is strictly related and requires the same character comparisons?
4. Is it possible to choose for each equivalent set of character comparison orders an order to maximize the values of the shift function? [1]

We consider a family $\mathcal{F}$ of *on-line* string matching algorithms in the deterministic sequential comparison model. An on-line algorithm can access the text through a window of length m: suppose the pattern $\mathcal{W}[0 \ldots m-1]$ is aligned with $\mathcal{T}[l \ldots l+m-1]$, $0 \leqslant l \leqslant n-m$, the on-line algorithm can inspect the characters in $\mathcal{T}[l \ldots l+m-1]$ of the text. The algorithms can access the input only by pairwise symbol comparisons that result in equal or unequal answers. Algorithms in the family $\mathcal{F}$ may differ only in the order in which pattern characters are compared with the corresponding text characters. We assume that the character comparison order does not change during the execution of the algorithm.

We like to stress that in our approach, for any fixed pattern we are given an order of comparing pattern characters to the *corresponding* text characters. Thus, when a mismatch occurs, the pattern is shifted to the right and the following iteration of the algorithm will resume the same order of character comparisons.

We do not account for the computation of the character comparison order which is considered as a piece of input data. Recalling the definition of shift in the example of the KMP algorithm, note that if $j < i$, then there is an overlap between the pattern instances before and after the shift: $\mathcal{W}[0 \ldots i-j-1] = \mathcal{W}[j \ldots i-1]$. KMP algorithm resumes the character comparisons from position $i-j$ in $\mathcal{W}$. In general it is possible to keep track of the characters already matched by a boolean array of size about m. This technique, being independent of the character comparison order, has been used in [10] and, in a slightly different fashion in [8] as indexes of potential occurrences of the pattern.

Finally, the algorithms in the family $\mathcal{F}$ can be indexed by permutations $J = (j_0, j_1, \ldots, j_{m-1})$ of the set $\{0, 1, \ldots, m-1\}$ of pattern positions. Since the permutation J represents the character comparison order, throughout the paper we ambiguously use J to denote both the permutation and the character comparison order it represents; moreover, we might use the expressions "comparison order" or simply "order" to mean character comparison order, unless specified otherwise.

Thus, the character comparison order of KMP algorithm [23] is given by $J_{\mathrm{KMP}} = (0, 1, \ldots, m-1)$, that of BM [5] by $J_{\mathrm{BM}} = (m-1, m-2, \ldots, 1, 0)$. In the Crochemore–Perrin algorithm [16] the pattern is divided into two substrings $\mathcal{W}[0 \ldots \xi]$ and $\mathcal{W}[\xi+1 \ldots m-1]$ by the Critical Factorization Theorem and the order of comparison is given by $J_{\mathrm{CP}} = (\xi+1, \ldots, m-1, \xi, \ldots, 1, 0)$. In Colussi's algorithm [11] the character comparison order is $J_{\mathrm{Col}} = (j_0, j_1, \ldots, j_{m-1})$, such that $j_i < j_{i+1}$ for $0 \leqslant i < t$ and $j_i > j_{i+1}$ for $t \leqslant i < m$, where t is the number of "noholes", i.e. the number of pattern

[1] Note that the performance of the text search step of string matching algorithms depends both on the size of the shifts and on the nonobliviousness of the algorithms, in the sense that they do not "forget" previous comparisons.

positions j_i that terminate at least a period of $\mathcal{W}[0..j_i - 1]$ (see Definition 3.3). We remind the reader that a string $\mathcal{W}[0..m-1]$ has a period of length p if $\mathcal{W}[i] = \mathcal{W}[i+p]$ for $i = 0..m-p-1$. Algorithms [19] and [10] do not belong to the family $\mathcal{F}$ because the character comparison order may change according to the history of previous comparisons. However, since there is a finite number of orders, it is possible to compute the shift function for each order using our approach.

In order to study the relations among the character comparison orders and the relative shift functions a general framework to "represent" the pattern $\mathcal{W}$ was needed. Therefore we introduce the notion of *Autocorrelation Matrix* $\mathcal{M}_{\mathcal{W}}$ of a string $\mathcal{W}$. Each entry $a_{i,j}$ in $\mathcal{M}_{\mathcal{W}}$ is given value zero if $\mathcal{W}[i] = \mathcal{W}[j]$, one if $\mathcal{W}[i] \neq \mathcal{W}[j]$. We believe that the autocorrelation matrix can be a useful and powerful tool to study combinatorial properties of strings. It can be used, for example, to detect periods, squares, palindromes and so on.

In this paper we show the following results:

1. A uniform definition (i.e. depending only on the order of character comparisons) of shift function δ_J for all algorithms in $\mathcal{F}$ via a function $imin_J$ based on the autocorrelation matrix $\mathcal{M}_{\mathcal{W}}$. The meaning of $imin_J(k) = i$ is that the ith character of the pattern (with respect to the order J) is the first one that witnesses that the pattern has not period k (i.e. such that $\mathcal{W}[j_i] \neq \mathcal{W}[j_i - k]$).
2. An algorithm to compute δ_J in $O(m \log\log m)$ time in the worst case, given $imin_J$ and sufficient conditions to compute δ_J in $O(m)$ time.
3. Computing $imin_J$ requires at most $O(m^2)$ comparisons. We show that if $imin_J$ can be computed in $O(g(m))$ time for an order J and a fixed pattern $\mathcal{W}$, then there exists a set C of equivalent orders (as defined in Section 4) on $\mathcal{W}$ such that $imin_K$ can be computed in $O(g(m))$ time for all orders $K \in C$. Moreover, we characterize orders that respectively minimize and maximize the values of the shift function in the set C and we show that for both those orders the function δ_K can be computed in linear time given $imin_K$.
4. Algorithms to compute both function $imin_J$ and shift function δ_J in $O(m)$ time for the wide class of compact orders as defined in Section 8 and, by the above point 3, for all the orders that maximize or minimize the shift function in the equivalence classes of the compact orders. (Note that J_{CP} is compact and J_{KMP} and J_{BM} can be considered as degenerate cases of compact orders.)

The paper is organized as follows: in Section 2 we give the definition and characterization of autocorrelation matrix $\mathcal{M}_{\mathcal{W}}$ of a string $\mathcal{W}$; in Section 3 we give a uniform definition of shift function for algorithms in $\mathcal{F}$ and we state sufficient conditions to compute δ_J in linear time, given function $imin_J$. In Section 4 we define equivalent comparison orders for any given pattern $\mathcal{W}$ and we characterize them by a partial order relation on the set of positions of the pattern. The links between the computation of the shift function and equivalent character comparison orders are studied. In Section 5 we describe some basic properties of the autocorrelation matrix $\mathcal{M}_{\mathcal{W}}$ that are used in the computation of the shift function for KMP comparison order in Section 6. Although it is well known that δ_{KMP} can be computed in $O(m)$ time [23], the approach by $\mathcal{M}_{\mathcal{W}}$

allows to extend the results to the computation of the shift function for BM comparison order in Section 7 and for general compact orders in Section 8. Indeed, both KMP and BM comparison orders can be seen as particular cases of compact orders.

2. The autocorrelation matrix $\mathcal{M}_{\mathcal{W}}$

In the comparison model algorithms can access the input string by pairwise symbol comparisons that test for equality. Let $\mathcal{W}[0 \ldots m-1]$ be a string on some alphabet Σ. The naive approach of comparing all pairs of symbols in $\mathcal{W}$ requires $O(m^2)$ comparisons. We introduce a binary matrix as a model to represent all the comparisons between pairs of symbols in a string.

Definition 2.1. The Autocorrelation Matrix $\mathcal{M}_{\mathcal{W}}$ of a string $\mathcal{W}[0 \ldots m-1]$ of length m is a $m \times m$ matrix whose entries are defined as follows:

$$a_{i,j} = \begin{cases} 0 & \text{if } \mathcal{W}[i] = \mathcal{W}[j], \\ 1 & \text{if } \mathcal{W}[i] \neq \mathcal{W}[j]. \end{cases}$$

It follows immediately from Definition 2.1 that $\mathcal{M}_{\mathcal{W}}$ is symmetric and that two columns (two rows) in $\mathcal{M}_{\mathcal{W}}$ having a 0 in the same position are equal. We call *k*th-*downward diagonal* the sequence of entries $a_{0,k}, a_{1,k+1}, \ldots, a_{m-k-1,m-1}$ in $\mathcal{M}_{\mathcal{W}}$. The *main diagonal* is the 0th-downward diagonal.

Obviously not all $\{0,1\}$ matrices are autocorrelation matrices. The following Lemma 2.2 states necessary and sufficient conditions to have an autocorrelation matrix of a string $\mathcal{W}$.

Lemma 2.2. *A* $\{0,1\}$ *matrix* $\mathcal{M}$ *of size* $m \times m$ *is the autocorrelation matrix* $\mathcal{M}_{\mathcal{W}}$ *of some string* $\mathcal{W}$ *of length* m *if and only if the main diagonal is 0-filled and for all* i,j,k,l *such that* $0 \leqslant i \leqslant k < m$ *and* $0 \leqslant j \leqslant l < m$ *the inequality* $a_{i,j} + a_{i,l} + a_{k,j} + a_{k,l} \neq 1$ *holds (i.e. there is no* 2×2 *submatrix of* $\mathcal{M}$ *having exactly one entry equal to* 1*).*

Proof. See Section 5. □

Properties of a string $\mathcal{W}$ can be described in terms of properties of its autocorrelation matrix. Of course, there can be more than one string $\mathcal{W}$ having the same autocorrelation matrix $\mathcal{M}_{\mathcal{W}}$; however, such strings cannot be distinguished by pairwise character comparisons. Thus, autocorrelation matrices allow to abstract from properties of strings that are not detectable by comparisons. The definition of the autocorrelation matrix $\mathcal{M}_{\mathcal{W}}$ suffices to obtain our first goal, that is a uniform definition of shift function for algorithms in the family $\mathcal{F}$, as presented in the following section. The properties of $\mathcal{M}_{\mathcal{W}}$ will be used later to develop the computation of the shift function for the wide class of compact orders defined in Section 8. Therefore we delay the proof of

Lemma 2.2, the investigation on the main features of $\mathcal{M}_{\mathcal{W}}$ and on the relations between periods of substrings of string $\mathcal{W}$ and $\mathcal{M}_{\mathcal{W}}$ to Section 5.

3. Shift function

Consider an on-line string matching algorithm $\mathcal{A} \in \mathcal{F}$. Let $\mathcal{W}[0 \ldots m-1]$ be a pattern and $\mathcal{M}_{\mathcal{W}}$ its autocorrelation matrix. Algorithm $\mathcal{A}$ compares pattern characters to the corresponding text characters in an order $J = (j_0, j_1, \ldots, j_{m-1})$ and we write $\mathcal{A}_J$ to indicate the relation between the on-line algorithm and the order of character comparisons. We shall use the subscript also everytime a function depends on the character comparison order J. When a mismatch or an occurrence of the pattern is found, the pattern is shifted to the right along the text.

Suppose that algorithm $\mathcal{A}_J$ gets unequal answer when it compares $\mathcal{W}[j]$ with $\mathcal{T}[l+j]$ (i.e. $\mathcal{W}[j] \neq \mathcal{T}[l+j]$) and that $a_{i,j} = 0$ in $\mathcal{M}_{\mathcal{W}}$ for some $i \leqslant j$. Then $\mathcal{W}[i] = \mathcal{W}[j] \neq \mathcal{T}[l+j]$: the pattern cannot match the text when it is shifted of $j-i$ positions (this holds also for $i = j$, since $a_{i,i} = 0$ for $0 \leqslant i \leqslant m-1$). We say that the shift $j-i$ is not successful. Therefore, if a mismatch $\mathcal{W}[j] \neq \mathcal{T}[l+j]$ occurs, then all shifts $j-i$ such that there is a zero in row i and column j of $\mathcal{M}_{\mathcal{W}}$ are not successful. Similarly, if $\mathcal{W}[j] = \mathcal{T}[l+j]$ and $a_{i,j} = 1$, then all shifts $j-i$ such that there is a one in row i and column j of $\mathcal{M}_{\mathcal{W}}$ are not successful. Define two sets of integers for all positions j of the pattern $\mathcal{W}$:

$$P(j) = \{j-i \mid 0 \leqslant i \leqslant j,\ a_{i,j} = 0\} \tag{1}$$

and

$$N(j) = \{j-i \mid 0 \leqslant i \leqslant j,\ a_{i,j} = 1\}. \tag{2}$$

$P(j)$ is the set of shifts which are not successful due to a mismatch between the pattern character $\mathcal{W}[j]$ and the corresponding text character; similarly for $N(j)$ in case of a match with the text. Consider a general execution of algorithm $\mathcal{A}_J$: the pattern is aligned with the text starting at some position l and we find that $\mathcal{W}[j_0] = \mathcal{T}[l+j_0], \mathcal{W}[j_1] = \mathcal{T}[l+j_1], \ldots, \mathcal{W}[j_{i-1}] = \mathcal{T}[l+j_{i-1}]$, and $\mathcal{W}[j_i] \neq \mathcal{T}[l+j_i]$; then all integers in the sets $N(j_0), N(j_1), \ldots, N(j_{i-1})$ and $P(j_i)$ are not successful shifts: the pattern should be shifted of the minimum positive integer that does not belong to any one of the previous sets. On the other hand, if an occurrence of the pattern is found starting at position l in the text $\mathcal{T}$ (i.e. $\mathcal{W}[j_0] = \mathcal{T}[l+j_0], \mathcal{W}[j_1] = \mathcal{T}[l+j_1], \ldots, \mathcal{W}[j_{m-1}] = \mathcal{T}[l+j_{m-1}]$), then all integers in the sets $N(j_0)$, $N(j_1), \ldots, N(j_{m-1})$ are not successful shifts. Also shift zero should not be considered as a possible shift; in order to treat shift zero in a uniform way, we put a sentinel (a character \$ different from all other pattern characters) at the end of the pattern in position $j_m = m$; the permutation J, the autocorrelation matrix $\mathcal{M}_{\mathcal{W}}$ and all previous definitions are naturally expanded. Then the set $P(j_m) = \{0\}$ contains only shift zero.

Thus, the shift function δ_J is defined as follows for algorithms $\mathcal{A}_J \in \mathcal{F}$:

$$\delta_J(i) = min\left\{k | k \notin P(j_i) \cup \bigcup_{t=0}^{i-1} N(j_t)\right\} \tag{3}$$

for all indexes i such that $0 \leqslant i \leqslant m$.

This definition of shift function depends only on the order of character comparisons. The computation of the shift function requires at most $O(m^2)$ comparisons; given the pattern $\mathcal{W}$ and the order J the naive approach is as follows: compute all sets $P(j_i)$ and $N(j_i)$ and use the definition (3) of δ_J. The computation of the sets $P(j_i)$ and $N(j_i), 0 \leqslant i \leqslant m$ is equivalent to the computation of all the $m(m-1)/2$ unknown entries in the upper triangle of $\mathcal{M}_{\mathcal{W}}$. (Recall that $\mathcal{M}_{\mathcal{W}}$ is symmetric and that $a_{i,i} = 0$ for $0 \leqslant i \leqslant m$ and $a_{i,m} = 1$ for $0 \leqslant i < m$.) However we do not need to know all the sets $N(j_t)$ in order to decide if an integer k belongs to the set $P(j_i) \cup \bigcup_{t=0}^{i-1} N(j_t)$. Indeed, the case $k = 0$ can be decided immediately since $0 \in P(j_i)$ for all i; for $1 \leqslant k \leqslant m$ we need only to know the first index t such that $k \in N(j_t)$; such a t always exists since $N(j_m)$ contains all k such that $1 \leqslant k \leqslant m$.

Define, for all k such that $1 \leqslant k \leqslant m$, the following function:

$$imin_J(k) = min\{i | k \in N(j_i)\}. \tag{4}$$

In order to explain the concept underlying the definition of function $imin_J$ we start by recalling the notion of period of a string.

Definition 3.1. A string $\mathcal{W}[0..m-1]$ has a period of length p if $\mathcal{W}[i] = \mathcal{W}[i+p]$ for $i = 0..m-p-1$.

Intuitively, the value of the function $imin_J(k)$ for a fixed k is the index of the first position in $\mathcal{W}$ with respect to the order J that witnesses that $\mathcal{W}$ has not period k. Similar information, restricted to the identical permutation $J = (0, 1, \ldots, m-1)$, is given by the *Failure Function* that is computed in the preprocessing step of the KMP algorithm and used in several string matching algorithms. The function $imin_J$ can be thought of as a generalization of the failure function to any character comparison order J on $\mathcal{W}$. The exact complexity of the failure function, i.e. the exact number of character comparisons needed to compute it, has been recently established in a joint work by these authors and Dany Breslauer [6, 7]. The computation of function $imin_J(k)$ clearly requires at most $O(m^2)$ comparisons. However, it is an open problem whether the quadratic bound is tight for general permutations J.

We shall show in Section 4 that there are families of orders such that the computation of function $imin_J$ requires only $O(m)$ comparisons. Among these orders there are those of KMP (Section 6), BM (Section 7) and compact orders as defined in Section 8 (compact orders are a generalization of Crochemore–Perrin comparison order).

The function $imin_J$ leads to a definition of shift function δ_J that can be easily implemented as we show next.

Lemma 3.2. *The two assertions*

$$- \ k \notin P(j_i) \cup \bigcup_{t=0}^{i-1} N(j_t), \tag{5}$$

$$- \ \mathit{imin}_J(k) = i \quad \text{or} \quad (k > j_i \quad \text{and} \quad \mathit{imin}_J(k) > i) \tag{6}$$

are equivalent for all k such that $1 \leqslant k \leqslant m$.

Proof. (5) $\Rightarrow$ (6): If $1 \leqslant k \leqslant j_i$ and $k \notin P(j_i)$, then $k \in N(j_i)$ and, since $k \notin N(j_t)$ for all t such that $0 \leqslant t < i$, then $\mathit{imin}_J(k) = i$. If $j_i < k \leqslant m$ then $k \notin N(j_i)$ and, since $k \notin N(j_t)$ for all t such that $0 \leqslant t < i$, then $\mathit{imin}_J(k) > i$.

(6) $\Rightarrow$ (5): If $\mathit{imin}_J(k) = i$, then $k \in N(j_i)$, and so $k \notin P(j_i)$, and $k \notin N(j_t)$ for all t such that $0 \leqslant t < i$. If $k > j_i$ and $\mathit{imin}(k) > i$ then $k \notin P(j_i)$, and $k \notin N(j_t)$ for all t such that $0 \leqslant t < i$. Then (5) holds. □

The shift function δ_J can be described in terms of function imin_J by Lemma 3.2:

$$\delta_J(i) = \mathit{min}\{k | \mathit{imin}_J(k) = i \text{ or } (k > j_i \text{ and } \mathit{imin}_J(k) > i)\}. \tag{7}$$

Given function imin_J, the shift function δ_J can be computed in $O(m \log m)$ time in the worst case, without any extra character comparison (except those to compute imin_J). The data structure used are a B-tree S and priority queues $T(i)$ for $i = 0..m$. The code in a Pascal-like notation is the following:

```
begin
    for i := 0 to m do T(i) := ∅;
    for k := 1 to m do T(imin_J(k)) := insert(k, T(imin_J(k)));
    {T(i) is the set of all k such that imin_J(k) = i}
    S := ∅;
    for i := m downto 0 do
        begin
            {S is the set of all k such that imin_J(k) > i}
            if T(i) = ∅ then
                    δ_J(i) = min(split_2(j_i, S))
                else
                    δ_J(i) = min(T(i));
            while T(i) ≠ ∅ do
                begin
                    k := min(T(i));
                    T(i) := delete(k, T(i));
                    S := insert(k, S)
                end
            {S is the set of all k such that imin_J(k) ⩾ i}
        end
end
```

See [2, 14] for implementation details of functions $\mathit{insert}(k, S)$ (the set $S \cup \{k\}$),

$split_2(k,S)$ (the set $\{x | x \in S \text{ and } x \geqslant k\}$), $min(S)$ (the minimum element of the set S) and $delete(k,S)$ (the set $S \setminus \{k\}$). Note that the bound $O(m \log m)$ can be further improved to $O(m \log\log m)$ time by using flat tree integer priority queue van Emde Boas et al. [27].

Function δ_J can be computed in linear time from function $imin_J$ under some conditions without extra character comparisons, as we show next. In particular for the character comparison orders of KMP (Section 6), BM (Section 7) and compact orders (Section 8) the complete computation of the shift function can be done in linear time with a linear number of character comparisons. In the next definition we shall adopt the same language used by [19].

Definition 3.3. We say that $\mathcal{W}[j_i]$ is a hole if there is no k such that $imin_J(k) = i$; $\mathcal{W}[j_i]$ is a nohole if $imin_J(k) = i$ for at least one k.

The following Theorem 3.4 gives sufficient conditions to compute δ_J in linear time, given function $imin_J$.

Theorem 3.4. *The function δ_J can be computed in $O(m)$ time via function $imin_J$ in case of:*

(i) $j_t > j_i \Longrightarrow t > i$ *for all noholes $\mathcal{W}[j_t]$ and holes $\mathcal{W}[j_i]$ (i.e. all noholes $\mathcal{W}[j_t]$ that follow a hole $\mathcal{W}[j_i]$ in the pattern also follow $\mathcal{W}[j_i]$ according to order J).*

(ii) $j_t > j_i \Longrightarrow t < i$ *for all noholes $\mathcal{W}[j_t]$ and holes $\mathcal{W}[j_i]$ (i.e. all noholes $\mathcal{W}[j_t]$ that follow a hole $\mathcal{W}[j_i]$ in the pattern precede $\mathcal{W}[j_i]$ according to order J).*

Proof. Consider any nohole $\mathcal{W}[j_i]$. Since $imin_J(k) = i$ implies $k \leqslant j_i$, then $\delta_J(i) = min\{k \mid imin_J(k) = i\}$. δ_J can be computed on the noholes by the following linear time code:

```
begin
    for i := 0 to m do δ_J(i) := 0;
    for k := m downto 1 do δ_J(imin_J(k)) := k;
end
```

After the execution of the previous code, the correct values of $\delta_J(i)$ are computed on all noholes (note that $\mathcal{W}[m] = \$$ is always a nohole since $j_m = m$ and $imin_J(m) = m$ for all orders J). On holes the value of $\delta_J(i)$ is still zero.

Consider any hole $\mathcal{W}[j_i]$; by definition $\delta_J(i) = min\{k \mid k > j_i \text{ and } imin_J(k) > i)\}$. Therefore the value of $\delta_J(i)$ can vary in the range $[j_i+1 \ldots p]$, where p is the minimum integer such that $k > j_i$ and $imin_J(p) = m$.

In case of (i), $imin_J(j_i + 1) > i$ since noholes follow holes in the order J; $\delta_J(i)$ can be computed on holes as follows:

```
for i := 0 to m do
    if δ_J(i) = 0 then δ_J(i) := j_i + 1
```

In case of (ii) the computation of $\delta_J(i)$ is not immediate. For all positions $j \leqslant m$ in the pattern $\mathscr{W}$, let $pmin(j)$ be the minimum period of $\mathscr{W}$ greater than j. Note that $pmin(j) = min\{p | p > j \text{ and } imin_J(p) = m\}$. For all holes $\mathscr{W}[j_i]$ the shift function $\delta_J(i) = pmin(j)$. Indeed if k is not a period of $\mathscr{W}$, then $k \in N(j_t)$ for some nohole $\mathscr{W}[j_t]$. The following linear time code computes δ_J on holes:

```
begin
    for j := m − 1 downto 0 do
        begin
            if imin_J(j + 1) = m then p := j + 1;
            {p is the minimum period of 𝒲 greater than j}
            pmin(j) := p
        end;
    for i := 0 to m do
        if δ_J(i) = 0 then δ_J(i) := pmin(j_i)
end □
```

In the next section we show that if there exists an order J such that $imin_J$ is computed in linear time (and therefore $O(m)$ character comparisons), then there exists a whole set of orders with the same property.

4. Character comparison orders

The definition (3) in the previous section shows once again that the fundamental feature of the shift function relative to a pattern $\mathscr{W}$ depends on the character comparison order J defined on it. We have seen that there exists a strong relation between δ_J and function $imin_J$ which is still depending on the permutation J.

The role of function $imin_J$ is very important since, besides leading to a fast computation of δ_J (in linear time if Theorem 3.4 holds and in $O(m \log\log m)$ time in the worst case), it contains the basic information on the periodicity structure of $\mathscr{W}$ relatively to the order J. It is natural to ask if such structure, on the fixed pattern $\mathscr{W}$, can be maintained also for other character comparison orders. An affermative answer would lead to the computation of new shift functions closely related to δ_J.

In this section the pattern $\mathscr{W}$ is considered fixed; we show that if we can compute the function $imin_J$ for $\mathscr{W}$ in $O(g(m))$ time for a character comparison order J, then we can compute it in $O(g(m))$ time for a whole set of orders. We shall start by defining equivalent character comparison orders and computing the corresponding *imin* functions using a linear-time transformation that does not require any character comparison. Recall that $imin_J$ can be always computed in $O(m^2)$ time; on the other hand $g(m) = \Omega(m)$ since m values have to be computed. The interesting sets of equivalent character comparison orders are those such that $g(m) = O(m)$.

Definition 4.1. Let $J = (j_0, j_1, \ldots, j_{m-1}, m)$ and $H = (h_0, h_1, \ldots, h_{m-1}, m)$ be two character comparison orders on a fixed pattern $\mathcal{W}$. We say that J is equivalent to H, $J \equiv H$, if $h_{imin_H(k)} = j_{imin_J(k)}$ for all $k, 1 \leqslant k \leqslant m$.

Lemma 4.2. *Given two equivalent orders $J \equiv H$ and function $imin_J$, let $\Pi = (\pi_0, \pi_1, \ldots, \pi_{m-1}, m)$ be the inverse permutation of H, i.e. $\pi_{h_i} = i$ for all i. Then $imin_H(k) = \pi_{j_{imin_J(k)}}$ for all k such that $1 \leqslant k \leqslant m$.*

Proof. Immediately from definitions. □

Given function $imin_J$ for the order J on $\mathcal{W}$, the following lines of code compute function $imin_H$ for all orders H on $\mathcal{W}$ such that $H \equiv J$:

```
begin
    for i := 0 to m do π_{h_i} := i;
    for k := 1 to m do
        begin
            t := j_{imin_J(k)};
            imin_H(k) := π_t
        end
end
```

The test for $H \equiv J$ cannot generally be done in linear time. However, in Lemma 4.5 we shall provide sufficient conditions for $H \equiv J$.

The idea underlying the definition of equivalent character comparison orders is that of grouping in the same class all the comparison orders such that noholes (defined for a given comparison order J) are defined for the same periodicity values and are kept in the same positions of the pattern although not necessarily checked in the same order. It might happen that, although noholes are placed in the same positions of the given pattern, they refer to different values of the periodicity k. For example, consider the pattern $\mathcal{W}[0..4] = aabba$, $J = (0,1,2,3,4)$ and $H = (0,1,3,2,4)$; then for $k = 2, imin_J(2) = 2$ and $imin_H(2) = 2$ but $j_2 = 2$ and $h_2 = 3$. Note that in this case $J \not\equiv H$ according to Definition 4.1. Since the definition of noholes depends on function $imin_J$ and on the permutation J considered, there seems to be a relation of precedence among some pattern positions for any character comparison order in the same equivalent class of J. Formally we show that for any order $J = (j_0, j_1, \ldots, j_{m-1}, m)$, the set of equivalent orders $\{H | H \equiv J\}$ can be characterized by the following relation $\preceq_J$ defined on the set of positions $\{0, 1, \ldots, m-1, m\}$ of the pattern.

Definition 4.3. We say that pattern position l dominates pattern position l' with respect to the character comparison order J, $l \preceq_J l'$, if there exists $k, 0 < k \leqslant m$ such that $l = j_{imin_J(k)}$ and $k \in N(l')$.

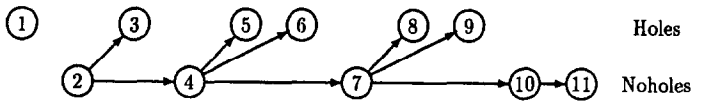

Fig. 1. The transitive closure of the relation $\preceq_{\mathrm{KMP}}$ for pattern $\mathscr{W}[0\ldots m] = aabacaacaab\$$.

The transitive closure of the relation $\preceq_J$ is a partial order embedded in the linear order $J = (j_0, j_1, \ldots, j_{m-1}, m)$ of indexes: if $j_i \preceq_J j_t$, then there exist $k, 0 < k \leqslant m$ such that $k \in N(j_t)$ and $i = imin_J(k)$. Therefore $i \leqslant t$.

In the next example we consider the permutation J_{KMP} of KMP algorithm on a pattern $\mathscr{W}$ and we show how function $imin_{\mathrm{KMP}}$ imposes a partial order relation on the positions of $\mathscr{W}$. Moreover, we show that the permutation J_{Col} of Colussi's algorithm is in the same equivalence class of J_{KMP}.

Example. Consider the comparison order in KMP algorithm, $J_{\mathrm{KMP}} = (0, 1, \ldots, m-1, m)$ and the pattern $\mathscr{W}[0\ldots m] = aabacaacaab\$$. The values of function $imin_{\mathrm{KMP}}(k)$ for $0 < k \leqslant m$ are listed below:

k	1	2	3	4	5	6	7	8	9	10	11
$imin_{\mathrm{KMP}}(k)$	2	2	4	4	7	7	7	11	10	10	11

The set of pattern positions $\{2, 4, 7, 10, 11\}$ corresponds to noholes (in other words to the set of values of $imin_{\mathrm{KMP}}$), while the set $\{0, 1, 3, 5, 6, 8, 9\}$ corresponds to holes. The transitive closure of the relation $\preceq_{\mathrm{KMP}}$ is shown in Fig. 1. A directed arc between two pattern positions represents the "dominate" relation.

Recall that, as we suggested in the introduction, one way of modifying KMP algorithm consists in changing the order of comparing pattern characters to the corresponding text characters while in a "match" phase.

Colussi's algorithm behaves exactly this way. It computes the values of function $imin_{\mathrm{Col}}$ as KMP algorithm does, therefore obtaining the same set of noholes and holes; the key difference is that Colussi's algorithm checks first noholes in increasing order of position, then holes in decreasing order (except the sentinel that is always checked last). Therefore $J_{\mathrm{Col}} = (2, 4, 7, 10, 9, 8, 6, 5, 3, 1, 0, 11)$ for the pattern of this example. It is clear that, by Definition 4.1, $J_{\mathrm{KMP}} \equiv J_{\mathrm{Col}}$. Note also that the precedence relation of Fig. 1 still holds; it follows that $\preceq_{\mathrm{KMP}} = \preceq_{\mathrm{Col}}$, too. Similar reasoning shows that if we consider $J_{\mathrm{BM}} = (m-1, m-2, \ldots, 1, 0, m)$, then $\preceq_{\mathrm{KMP}} \neq \preceq_{\mathrm{BM}}$ and $\preceq_{\mathrm{Col}} \neq \preceq_{\mathrm{BM}}$ (this is true for most patterns, not only for the one of this example).

The above example suggests that the notion of equivalence between character comparison orders and that of equality of the dominate relations are linked. The following Theorem 4.4 shows that such link is the equivalence of the two notions.

Theorem 4.4. *Let J and H be two character comparison orders. Then assertions* (a)–(c) *below are equivalent*:

(a) $H \equiv J$;

(b) *the relation* $\preceq_J$ *(respectively* $\preceq_H$*) embeds in* H (J)*, i.e. if* $h_i \preceq_J h_t$*, then* $i \leqslant t$ *(if* $j_i \preceq_H j_t$*, then* $i \leqslant t$*);*
(c) $\preceq_J = \preceq_H$.

Proof. (a) $\Rightarrow$ (b): Let J and H be such that $H \equiv J$ and let i and t be such that $h_i \preceq_J h_t$. Then, there exists k such that $k \in N(h_t)$ and $h_i = j_{imin_J(k)}$. Since $H \equiv J$, then $h_i = j_{imin_J(k)} = h_{imin_H(k)}$ and so $i = imin_H(k)$. Thus $i \leqslant t$ since $k \in N(h_t)$.

(b) $\Rightarrow$ (a): Let $l = j_{imin_J(k)}$ and let i be such that $h_i = l$. Then $k \notin P(h_i)$. Moreover, if $k \in N(h_t)$ then $h_i \preceq_J h_t$ and so $i \leqslant t$. Thus, $k \notin N(h_t)$ for all t such that $t < i$ and then $imin_H(k) = i$ and $h_{imin_H(k)} = h_i = l = j_{imin_J(k)}$. (Similarly for $\preceq_H$)

(b) $\Rightarrow$ (c): From definitions.

(c) $\Rightarrow$ (b): Obvious, since $\preceq_J = \preceq_H$ embeds in H.

(a) $\Rightarrow$ (c): $j_{imin_J(k)} = h_{imin_H(k)}$ for all k, by (a). Thus $l \preceq_J l'$ if and only if there exists k such that $k \in N(l')$ and $l = j_{imin_J(k)}$ if and only if $k \in N(l')$ and $l = h_{imin_H(k)}$ if and only if $l \preceq_H l'$.

(c) $\Rightarrow$ (a): From definitions. □

Given a character comparison order J, the Definition 4.3 of dominate relation $\preceq_J$ involves function $imin_J$ and so it imposes to the noholes of the given pattern $\mathscr{W}$ to dominate some other pattern positions; on the other hand holes do not have to meet any constraint (refer to Fig. 1 as an example). This observation implies immediately that all character comparison orders obtained by postponing the checking of some holes are equivalent. Note that there might be an exponential number of such orders. The following Lemma 4.5 states that the simple operation of moving a hole after a nohole creates an equivalent character comparison order.

Lemma 4.5. *If* $\mathscr{W}[j_t]$ *is a hole for the character comparison order* $J = (j_0, \ldots, j_t, j_{t+1}, \ldots, j_{m-1}, m)$*, then the character comparison order* $H = (h_0, \ldots, h_t, h_{t+1}, \ldots, h_{m-1}, m)$ *such that* $h_t = j_{t+1}$, $h_{t+1} = j_t$ *and* $h_i = j_i$ *otherwise is equivalent to* J.

Proof. Since $\mathscr{W}[j_t]$ is a hole, then for any $k \in N(j_t)$ there exists $s < t$ such that $k \in N(j_s)$. Moreover, $N(j_t) = N(h_{t+1})$ and so for any $k \in N(h_{t+1})$ there exists $s < t$ such that $k \in N(j_s) = N(h_s)$. Thus, $\mathscr{W}[h_{t+1}]$ is still a hole for order H. Therefore $\preceq_H = \preceq_J$, and by Theorem 4.4 $H \equiv J$. □

There are examples in the literature of character comparison orders J such that $imin_J$ is computed in linear time (we just mention $J_{KMP}, J_{BM}, J_{CP}, J_{Col}$) and we shall introduce more examples in Section 8. So, consider a character comparison order J such that $imin_J$ is computed in linear time. Since any character comparison order H that is obtained from J by moving forward the holes is equivalent to J, function $imin_H$ is also computed in linear time by Lemma 4.2. In particular, if the equivalent order H is obtained from J by moving all holes after all noholes, then point (ii) of Theorem 3.4 holds for H. So, not only can the shift function δ_H be computed in linear time,

but also the condition for δ_H to take its maximum value $\delta_H(i) = pmin(h_i)$ on holes is satisfied. (Similarly, if the equivalent order H obtained from J satisfies point (i) of Theorem 3.4, then the shift function δ_H can still be computed in linear time and it takes its minimum value $\delta_H(i) = j_i + 1$ on the holes. However, it is not so easy to obtain from J an equivalent order H that satisfies point (i) of Theorem 3.4 since in general a hole cannot be moved freely before a nohole.)

This strategy can be applied to each equivalence class of character comparison orders: given an order J it is possible to choose a character comparison order $H \equiv J$ such that the shift function δ_H has maximum value $\delta_H(i)$ on hole h_i. Note that there is a trade-off between maximizing the values of the shift function on holes and maximizing the number of comparisons that can be saved in the next round of the execution of the algorithm $\mathcal{A}_H$. However, maximizing the values of the shift function is a useful euristic to obtain efficient string matching algorithms. This strategy has been applied in Colussi's algorithm [11]. We have shown in the Example of this section that J_{Col} is in the same equivalence class of J_{KMP}. Indeed, in Colussi's algorithm both functions $imin_{\mathrm{Col}}$ and δ_{Col} are computed in linear time, using $O(m)$ comparisons. The permutation J_{Col} is such that noholes are inspected from left to right before holes which are inspected in the reverse order. This choice of comparing characters is such that if a mismatch occurs in comparing hole h_i with the corresponding text character, then $\delta_{\mathrm{Col}}(i) = pmin(j_i)$ and Colussi's algorithm does not need to check any of the $\mathcal{W}[0 \ldots m - pmin(j_i) - 1]$ characters in the new alignment with the text. The algorithm gets a better performance in the text processing stage: at most $\frac{3}{2}n - \frac{1}{2}m$ comparisons versus $2n - m$ comparisons of the KMP algorithm are made. The same strategy has been also applied to BM algorithm in [12] obtaining an improvement of the worst-case bound from $3n$ to $2n$. Last, in [18] it was proved that this strategy applied to CP algorithm allows to save character comparisons, but does not improve the worst-case bound.

Finally, string matching algorithms that perform less than $\frac{3}{2}n$ comparisons in the text processing step [10, 19] do change the character comparison order during execution to use all the information gathered in previous rounds. It is an open problem if $\frac{3}{2}n$ is a tight bound for algorithms that do not change character comparison order, i.e. for algorithms in the family $\mathcal{F}$. The lower bound of $\frac{3}{2}n$ has been proved to hold for a large subclass of $\mathcal{F}$ in [18].

5. More about $\mathcal{M}_{\mathcal{W}}$

In this section we shall prove Lemma 2.2 and study some basic properties of the autocorrelation matrix $\mathcal{M}_{\mathcal{W}}$ of a string $\mathcal{W}[0 \ldots m-1]$. These properties will be used in the following sections devoted to the computation of the shift function for the character comparison orders $J_{\mathrm{KMP}}, J_{\mathrm{BM}}$ and J_C, where C is a compact character comparison order.

Proof of Lemma 2.2. Let $\mathcal{M} = \mathcal{M}_{\mathcal{W}}$ be the autocorrelation matrix of string $\mathcal{W}$; then $a_{i,i} = 0$ since $\mathcal{W}[i] = \mathcal{W}[i]$ for all i. Assume $a_{i,j} = a_{i,l} = a_{k,j} = 0$. Then,

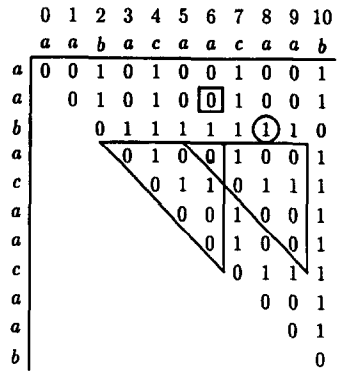

Fig. 2. The autocorrelation matrix $\mathcal{M}_{\mathcal{W}}$ for pattern $\mathcal{W}[0\ldots m-1] = aabacaacaab$, the relations between entries and the periodicity induced by substring $\mathcal{W}[3\ldots 9]$.

$\mathcal{W}[i] = \mathcal{W}[j], \mathcal{W}[i] = \mathcal{W}[l]$ and $\mathcal{W}[k] = \mathcal{W}[j]$ hold. By the transitivity property of equality $\mathcal{W}[k] = \mathcal{W}[l]$ and $a_{k,l} = 0$.

Suppose $\mathcal{M}$ is a $\{0,1\}$ matrix of size $m \times m$, such that $a_{i,j} + a_{i,l} + a_{k,j} + a_{k,l} \neq 1$ and $a_{i,i} = 0$. Let $\mathcal{W}$ be any string such that $\mathcal{W}[i] = \mathcal{W}[j]$ if and only if columns i and j are equal in $\mathcal{M}$.

If $\mathcal{W}[i] = \mathcal{W}[j]$ then $a_{i,j} = a_{i,i} = 0$. If $\mathcal{W}[i] \neq \mathcal{W}[j]$ then there exists l such that $a_{l,i} \neq a_{l,j}$. Assume $a_{l,i} = 0$ and $a_{l,j} = 1$; since $a_{i,i} = 0$ then $a_{i,j} = 1$. □

We shall consider only entries $a_{i,j}$ such that $0 \leqslant i \leqslant j < m$ (i.e. the upper triangle of $\mathcal{M}_{\mathcal{W}}$) since $\mathcal{M}_{\mathcal{W}}$ is symmetric with respect to the main diagonal. Refer to Fig. 2 as an example of autocorrelation matrix.

The following two lemmas describe some relations between entries of $\mathcal{M}_{\mathcal{W}}$.

Lemma 5.1. *Let $a_{i,j}$ be any entry in $\mathcal{M}_{\mathcal{W}}$. Then $a_{i,j} = 0$ if and only if $a_{p,j} = a_{p,i}$ for all p such that $0 \leqslant p < m$.*

Proof. By Definition 2.1 $a_{i,j} = 0$ if and only if $\mathcal{W}[i] = \mathcal{W}[j]$. Let p be such that $0 \leqslant p < m$; then $a_{p,j} = 0$ if and only if $\mathcal{W}[p] = \mathcal{W}[j]$ if and only if $a_{p,i} = 0$. Viceversa, take $p = i$; then $a_{i,j} = a_{i,i} = 0$. □

Intuitively Lemma 5.1 states that $a_{i,j} = 0$ if and only if columns i and j are equal. $\mathcal{M}_{\mathcal{W}}$ being symmetric the same holds for rows i and j. Consider as an example the entry $a_{1,6} = 0$ in Fig. 2 (the entry is highlighted by a square); then columns one and six are equal (similarly for rows one and six).

Lemma 5.2. *Suppose entry $a_{i,j} = 1, 0 \leqslant i \leqslant j < m$ in the autocorrelation matrix $\mathcal{M}_{\mathcal{W}}$ of string $\mathcal{W}$. Then either $a_{p,j} = 1$ or $a_{p,i} = 1$ (or both) for all p such that $0 \leqslant p < m$.*

Proof. By definition $a_{i,j} = 1$ if and only if $\mathcal{W}[i] \neq \mathcal{W}[j]$. Let p be such that $0 \leqslant p < m$; then $a_{p,j} = 0$ if and only if $\mathcal{W}[p] = \mathcal{W}[j] \neq \mathcal{W}[i]$ and so $a_{p,i} = 1$. If $a_{p,i} = 0$, then $\mathcal{W}[p] = \mathcal{W}[i] \neq \mathcal{W}[j]$ and so $a_{p,j} = 1$. □

Intuitively, Lemma 5.2 states that if $a_{i,j} = 1$ then columns i and j cannot have a zero in the same position. Consider as an example the entry $a_{2,8} = 1$ in Fig. 2 (the entry is highlighted by a circle); then there is no pair of corresponding entries in columns two and eight which is equal to zero.

We shall indicate by $\mathcal{W}[i \ldots j], 0 \leqslant i \leqslant j \leqslant m-1$ the substring of $\mathcal{W}$ starting at position i and ending at position j. The periodicity of $\mathcal{W}[i \ldots j]$ is equivalent to sequences of consecutive zeros in a diagonal of $\mathcal{M}_{\mathcal{W}}$ as follows:

Lemma 5.3. *The substring $\mathcal{W}[i \ldots j]$ is p-periodic if and only if all the entries $a_{i,i+p}, a_{i+1,i+p+1}, \ldots, a_{j-p,j}$ in $\mathcal{M}_{\mathcal{W}}$ are equal to zero.*

Proof. $\mathcal{W}[i \ldots j]$ is p-periodic if and only if $\mathcal{W}[k] = \mathcal{W}[k+p]$ for all k such that $i \leqslant k \leqslant j-p$ if and only if $a_{k,k+p} = 0$ for all k such that $i \leqslant k \leqslant j-p$. □

The periodicity of $\mathcal{W}[i \ldots j]$ induces a periodicity of a whole region of $\mathcal{M}_{\mathcal{W}}$ as the next Lemma 5.4 shows.

Lemma 5.4. *Let $\mathcal{W}[i \ldots j]$ be p-periodic; then $a_{k,q} = a_{k,q-p}$ and $a_{q,k} = a_{q-p,k}$ for all k and q such that $0 \leqslant k < m$ and $i + p \leqslant q \leqslant j$.*

Proof. Since $\mathcal{W}[i \ldots j]$ is p-periodic, then the entries $a_{i,i+p} \ldots a_{j-p,j}$ are equal to zero by Lemma 5.3. Let q be such that $i + p \leqslant q \leqslant j$. Moreover, $a_{q,q} = 0$ and $a_{q-p,q-p} = 0$ since the main diagonal of $\mathcal{M}_{\mathcal{W}}$ is zero filled. Then by Lemma 5.1 it follows that $a_{k,q} = a_{k,q-p}$ and $a_{q,k} = a_{q-p,k}$ for all k such that $0 \leqslant k < m$. □

Refer to the substring $\mathcal{W}[3 \ldots 9]$ in Fig. 2; the substring is 3-periodic and the entries in the same positions of the two triangular areas are equal.

6. Shift function for KMP character comparison order and equivalent orders

In this section we show how to compute the shift function δ_{KMP} for KMP character comparison order using the autocorrelation matrix $\mathcal{M}_{\mathcal{W}}$ of a string $\mathcal{W} = \mathcal{W}[0 \ldots m]$ containing the sentinel as last character.

The preprocessing step of KMP algorithm (refer to [23]) is solved by computing the *Failure Function* in linear time and at most $2m-4$ character comparisons. According to the general definition of shift function given in the introduction, it follows that δ_{KMP} is related to the failure function of KMP algorithm (that we call ff function) by the relation $\delta_{\mathrm{KMP}}(i) = i - ff(i+1) + 1$.[2] Indeed, it is well known that also δ_{KMP} can be computed in $O(m)$ time and at most $2m-4$ character comparisons. However the computation of ff in [23] cannot be easily generalized to compute shift functions δ_J for other orders J, not even for an equivalent order like J_{Col}. Therefore we shall adopt

[2] Note that in [23] the pattern positions are indexed starting from one instead of zero.

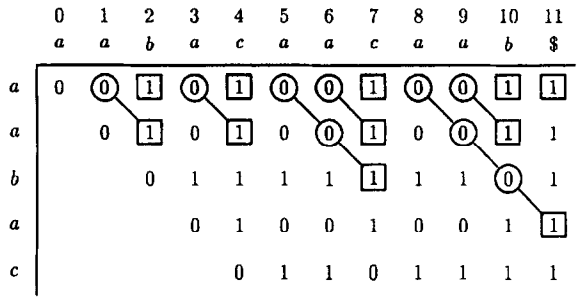

Fig. 3. Pattern $\mathscr{W}[0\ldots m] = aabacaacaab\$$ and entries $a_{i-k,i} = 1$ such that $i = imin_{\mathrm{KMP}}(k)$.

a different strategy to compute δ_{KMP}; our strategy will be based on the definitions of Section 3 and the autocorrelation matrix $\mathscr{M}_{\mathscr{W}}$.

According to Eq. (7) of Section 3 the computation of the shift function δ_{KMP} requires the computation of the values $imin_{\mathrm{KMP}}(k)$, $0 < k \leqslant m$. Since KMP comparison order is $J = (0, 1, \ldots, m-1, m)$, it trivially satisfies point (i) of Theorem 3.4; therefore δ_{KMP} will be computed in linear time from function $imin_{\mathrm{KMP}}$, without extra character comparisons, as the theorem shows. Moreover, it is easy to transform the values of function $imin_{\mathrm{KMP}}$ into the values of function $imin_J$ for equivalent orders J, following the lines of code after Lemma 4.2. Finally, the relative shift functions can be computed in $O(m \log\log m)$ time in the worst case or in $O(m)$ time if Theorem 3.4 holds. In particular it is immediate to obtain the values of δ_{Col}.

So, we are reduced to the problem of efficiently computing function $imin_{\mathrm{KMP}}$.

In terms of entries of $\mathscr{M}_{\mathscr{W}}$, the value of $imin_{\mathrm{KMP}}(k)$, $0 < k \leqslant m$, is the index i of the first column in $\mathscr{M}_{\mathscr{W}}$ such that $a_{i-k,i} = 1$. We can equivalently compute the length $pref(k)$ of the maximal 0-filled prefix of the kth-downward diagonal in $\mathscr{M}_{\mathscr{W}}$. In the sequel we shall refer to the downward diagonal simply as the diagonals of $\mathscr{M}_{\mathscr{W}}$, unless specified otherwise. An example is drawn in Fig. 3, where the prefixes of the given pattern are highlighted by circles, while squares correspond to the entries $a_{i-k,i} = 1$ such that $i = imin_{\mathrm{KMP}}(k)$.

We formally define function $pref$ as follows:

Definition 6.1. We say that $pref(k)$, $0 < k \leqslant m$ is the length of the maximal 0-filled prefix of the sequence of entries $a_{0,k} \ldots a_{m-k,m}$ in $\mathscr{M}_{\mathscr{W}}$.

We can easily and efficiently recover the values of function $imin_{\mathrm{KMP}}$ by observing that $imin_{\mathrm{KMP}}(k) = k + pref(k)$ for $0 < k \leqslant m$.

The problem of computing function $pref$ has been addressed also in [6] and recently in [7]. The authors refer to it as string self-prefix problem; they give a linear-time algorithm (in [7]) that requires at most $2m - \lceil 2\sqrt{m} \rceil$ character comparisons, matching an equal lower bound previously provided in [6]. The algorithm given in [7] is of great

theoretical interest since it determines the exact complexity (i.e. the exact number of character comparisons) of the problem. The string self-prefix problem is similar to the computation of the failure function ff in KMP algorithm: ff can be easily computed from $pref$ in linear time without additional comparisons by using Eq. (8) below. Therefore, the tight bounds on the string self-prefix problem apply also to the computation of the failure function (improving the previous bound of at most $2m-4$ character comparisons of the KMP algorithm).

However, the algorithm designed in [7] is rather complicated and for any practical purpose an algorithm that performs at most $2m$ character comparisons is fairly efficient.

The relations between function $pref$ and ff can be more precisely outlined in terms of the correlation matrix; $ff(i)$ is the length of the longest 0-filled prefix of a downward diagonal that ends in column i. Thus, while function $pref$ aims to record all the maximal 0-filled prefixes of the downward diagonals, the function ff records only the longest one that ends in each column. Then the following relation holds:

$$ff(i) = \max\{h \mid pref(i-h) = h-1\} \cup \{0\}. \tag{8}$$

The computation of ff in KMP algorithm is such that the lengths of all the 0-filled prefixes of the downward diagonals are indirectly calculated, but only the lengths of the longest ones are stored in ff. It is possible to obtain an algorithm to compute function $pref$ by inserting statements that store the values of $pref$ in the right places of the KMP algorithm. The algorithm obtained this way uses twice the memory of KMP (it stores two functions ff and $pref$). On the other end we can design an algorithm that works similar to KMP algorithm, but uses only function $pref$, still using at most $2m-4$ character comparisons. The advantages of computing directly function $pref$ are rooted both in the extension of the computation of the shift function for equivalent character comparison orders and in an efficient and conceptually easy way of computing the shift function for BM character comparison order (Section 7) and, more importantly, for all compact orders (Section 8). Note that BM and compact orders are not equivalent in general to KMP character comparison order.

The remaining part of this section is organized as follows: we start by stating some properties of function $pref$ in the following two lemmas. Such properties will be translated in our linear-time algorithm. Finally we show that the algorithm performs at most as many comparisons as KMP algorithm.

Lemma 6.2. *Let* $pref(k) = s$. *Then* $pref((l+1)k) = s - lk$ *for all* l *such that* $0 \leqslant l \leq \lfloor s/k \rfloor$.

Proof. By Lemma 5.3, the string $\mathcal{W}[0..k+s-1]$ is k periodic. By Lemma 5.4, entries $a_{0,(l+1)k} \ldots a_{s-lk,k+s}$ are equal to entries $a_{lk,(l+1)k} \ldots a_{s,k+s}$. Since $pref(k) = s$, then $a_{lk,(l+1)k} \ldots a_{s-1,k+s-1}$ are zero and $a_{s,k+s} = 1$. Thus $a_{0,(l+1)k} \ldots a_{s-lk-1,k+s-1}$ are zero and $a_{s-lk,k+s} = 1$ and so $pref((l+1)k) = s - lk$. □

Suppose we computed the value $pref(k_0) = s$; then we can compute almost all values of $pref(k_0 + p)$ for $0 < p \leqslant s$ without any extra character comparison, looking

at values of function *pref* in a previous region of $\mathcal{M}_{\mathcal{W}}$. Lemma 6.3 formalizes this intuitive idea.

Lemma 6.3. *Let* $pref(k_0) = s$ *and let* p *be such that* $0 < p \leqslant s$. *Then*:
(a) *if* $pref(p) < s - p$ *then* $pref(k_0 + p) = pref(p)$,
(b) *if* $pref(p) > s - p$ *then* $pref(k_0 + p) = s - p$,
(c) *if* $pref(p) = s - p$ *then* $pref(k_0 + p) \geqslant s - p$.

Proof. Since $pref(k_0) = s$, then $\mathcal{W}[0 \ldots k_0 + s - 1]$ is k_0 periodic. By Lemma 5.4, $a_{t,k_0+p} = a_{t,p}$ for all t and p such that $0 \leqslant t < m$ and $0 \leqslant p < s$.

(a) Since $pref(p) < s - p$, then entries $a_{0,p}, a_{1,p+1}, \ldots, a_{pref(p)-1,p+pref(p)-1}$ in the pth-diagonal are zero and $a_{pref(p),p+pref(p)} = 1$. By the k_0 periodicity of $\mathcal{W}[0 \ldots k_0 + s - 1]$ the corresponding entries $a_{0,k_0+p}, a_{1,k_0+p+1}, \ldots, a_{pref(p)-1,k_0+p+pref(p)-1}$ in the (k_0+p)th-diagonal are zero and $a_{pref(p),k_0+p+pref(p)} = 1$. Thus $pref(k_0+p) = pref(p)$.

(b) Since $pref(p) > s - p$, then entries $a_{0,p}, a_{1,p+1}, \ldots, a_{s-p,s}$ in the pth-diagonal are zero. By the k_0 periodicity of $\mathcal{W}[0 \ldots k_0 + s - 1]$ the entries $a_{0,k_0+p}, a_{1,k_0+p+1}, \ldots, a_{s-p-1,k_0+s-1}$ in the $(k_0 + p)$th-diagonal are zero. Since $a_{s,k_0+s} = 1, a_{s,s} = 0$ and $a_{s-p,s} = 0$, then $a_{s-p,k_0+s} = 1$ by Lemma 5.2. Thus $pref(k_0 + p) = s - p$.

(c) Since $pref(p) = s - p$, then entries $a_{0,p}, a_{1,p+1}, \ldots, a_{s-p-1,s-1}$ in the pth-diagonal are zero and $a_{s-p,s} = 1$. By the k_0 periodicity of $\mathcal{W}[0 \ldots k_0 + s - 1]$ the corresponding entries $a_{0,k_0+p}, a_{1,k_0+p+1}, \ldots, a_{s-p-1,k_0+s-1}$ in the $(k_0 + p)$th-diagonal are zero. Since $a_{s,k_0+s} = 1, a_{s,s} = 0$ and $a_{s-p,s} = 1$, we cannot determine by Lemma 5.2 if $a_{s-p,k_0+s} = 1$ or $a_{s-p,k_0+s} = 0$. □

If points (a) or (b) of Lemma 6.3 hold, then we can assign the right value to function *pref*. If point (c) holds, only a lower bound for the value of the function *pref* is given by Lemma 6.3.

Lemma 6.3 can be translated into a linear time algorithm that computes function *pref* (and consequently function $imin_{\mathrm{KMP}}$).

The algorithm inspects entries of $\mathcal{M}_{\mathcal{W}}$ starting from the left upper corner and moving along two different directions: down along a diagonal and up along a column. We refer to it as pref-algorithm. The pref-algorithm is as follows:

Step 0 (*Inizialization.*) Set $j = 1, s = 0$ and go to Step 1.

Step 1 (*Move down along diagonal j.*) If $j > m$ then stop. Otherwise, we know that the entries $a_{0,j}, a_{1,j+1}, \ldots, a_{s-1,j+s-1}$ are equal to zero. Starting from $a_{s,j+s}$ move down along the jth-diagonal by incrementing s until the first entry $a_{s,j+s} = 1$ is found. Set $pref(j) = s$ and go to Step 2.

Step 2 (*Move up along column* $j + s$.) We know that $pref(j) = s$. Starting from position $k = j + 1$ assign the right value to $pref(k)$ for all k such that cases (a) or (b) of Lemma 6.3 applies and until either $k = j + s + 1$ or $pref(k - j) = j + s - k$ is found (i.e. case (c) of Lemma 6.3 holds). In the former case:

(a) set $j = j + s + 1, s = 0$ and go to Step 1.

In the latter case inspect entry $a_{j+s-k,j+s}$.

If $a_{j+s-k,j+s} = 0$ then:

(b) set $s = j + s - k + 1, j = k$, and go to Step 1.

If $a_{j+s-k,j+s} = 1$ then:

(c) set $s = j + s - k, j = k$ and set $pref(j) = s$ and repeat Step 2.

Pref-algorithm behaves like KMP algorithm for the computation of function ff. Indeed, Step 1 represents the matching phase when corresponding characters are found to be equal, proceeding from left to right. In Step 2 the list of characters to compare when a mismatch occurs is given; the order of comparing these characters is, like in KMP algorithm, by decreasing positions in the pattern. Therefore the bound of at most $2m - 4$ character comparisons applies to pref-algorithm as well.

We show how to transform the output of pref-algorithm into the output of the failure function in KMP algorithm, for which the pattern positions are indexed from 1 to $m+1$. In Step 0 we inizialize the failure function to $ff(1) = 0$; in Step 1 we add the statement that assigns the value $ff(j+s+1) = ff(s+1)$ while $a_{s,j+s} = 0$, i.e. while we are moving down along diagonal j and the statement that assigns the value $ff(j + s + 1) = s + 1$ when the first entry $a_{s,j+s} = 1$ is found; in Step 2 we substitute the loop that starts from $k = j+1$ and looks for the first value k such that $k = j+s+1$ or $pref(k-j) = j+s-k$ by the statement $k := j+s+1-ff(s+1)$. Indeed, by Eq. (8), $k := j+s+1-ff(s+1)$ is just the first k such that $k = j+s+1$ or $pref(k-j) = j+s-k$. Last, we eliminate all assignments to function $pref$.

As a remark note that the algorithms in [11, 13] have a better performance than $2m - 4$ character comparisons, but they cannot be used to compute function $pref$ since they do not compare the characters from left to right.

7. Shift function for BM character comparison order

In this section we show how to compute the shift function for BM character comparison order by using function $pref$ that is computed by the pref-algorithm of Section 6. We consider a string $\mathscr{W}[0 \ldots m]$ containing the sentinel as last character and we proceed following the strategy outlined in the previous section.

BM character comparison order is given by $J = (j_0, \ldots, j_{m-1}, m) = (m - 1, m - 2, \ldots, 1, 0, m)$, and so $j_i = m - 1 - i$ for i in the range $0 \leqslant i < m$ and $j_m = m$. According to our strategy, we first compute the values $imin_{\mathrm{BM}}(k)$, for $1 \leqslant k \leqslant m$. The value of $imin_{\mathrm{BM}}(k)$ is given by $m - 1 - l$, where l is the maximum column index in the autocorrelation matrix $\mathscr{M}_{\mathscr{W}}$, such that $0 \leqslant l < m$ and $a_{l-k,l} = 1$. In particular if no such l exists, then only the column of the sentinel contains one and $imin_{\mathrm{BM}}(k)$ is set to m. We can equivalently compute the length $postf(k)$ of the maximal 0-filled postfixes of the downward diagonals in $\mathscr{M}_{\mathscr{W}}$ (with the last column removed).

We formally define function $postf$ as follows:

Definition 7.1. We say that $postf(k)$, $0 < k \leqslant m$, is the length of the maximal 0-filled postfix of the sequence of entries $a_{m-1-k,m-1}\, a_{m-2-k,m-2} \ldots a_{0,k}\, a_{m-k,m}$ in $\mathcal{M}_{\mathcal{W}}$.

We can then recover the values of function $imin_{\mathrm{BM}}$ by observing that for $0 < k \leqslant m$ $imin_{\mathrm{BM}}(k) = m - 1 - postf(k)$ if $postf(k) \neq m - k$ and $imin_{\mathrm{BM}}(k) = m$ if $postf(k) = m - k$.

The computation of function $postf$ can be performed in $O(m)$ time from function $pref$; moreover, no extra character comparisons are required except those to compute function $pref$. The stages of the computation can be described as follows:

(1) Let $\overline{\mathcal{W}} = \mathcal{W}_{m-1}\mathcal{W}_{m-2}\ldots\mathcal{W}_0\$$ be the reverse string of $\mathcal{W}$, leaving the sentinel at the end.
(2) Compute the values of function $pref$ on $\overline{\mathcal{W}}$.
(3) Compute the values of function $postf$ on $\mathcal{W}$ from $pref$ on $\overline{\mathcal{W}}$ by using the relation $postf(k) = pref(m-k-1), 0 < k < m$ and $postf(m) = pref(m)$.

The above stages can be clearly implemented in linear time and the upper bound of $2m - 4$ character comparisons still holds for the computation of function $postf$ since character comparisons are performed only in stage (2); thus, also function $imin_{\mathrm{BM}}$ can be computed in linear time. Moreover, since BM character comparison order satisfies point (ii) of Theorem 3.4, the shift function δ_{BM} is also computed in linear time as the code in the theorem mentioned shows. Finally, we recall that KMP and BM comparisons orders are not equivalent according to Definition 4.1, but it is possible to compute $imin_{\mathrm{KMP}}, imin_{\mathrm{BM}}, \delta_{\mathrm{KMP}}$ and δ_{BM} in linear time.

Functions $pref$ and $postf$ turn out to be useful tools in the computation of the shift function for compact orders as we show in the following section.

8. Shift function for compact orders

In this section we define compact orders and we present a linear-time algorithm to compute the shift function δ_C for a compact order C on a string $\mathcal{W}[0\ldots m]$ containing the sentinel as last character. The autocorrelation matrix $\mathcal{M}_{\mathcal{W}}$ will again play an important role to describe the different steps of the computation.

Definition 8.1. The character comparison order $C = (j_0, \ldots, j_{m-1}, m)$ is compact if the characters $\mathcal{W}[j_0], \mathcal{W}[j_1], \ldots, \mathcal{W}[j_i]$ cover a segment of pattern $\mathcal{W}[0\ldots m]$ for all $i, 0 \leqslant i \leqslant m$.

The following is a natural definition describing how the pattern positions extend the compact order C.

Definition 8.2. We say that position $j_i, 0 \leqslant i \leqslant m$, is a left (respectively right) position in the compact order $C = (j_0, \ldots, j_{m-1}, m)$ if it extends the segment covered by the characters $\mathcal{W}[j_0]$, $\mathcal{W}[j_1], \ldots, \mathcal{W}[j_{i-1}]$ to the left (respectively right).

Note that, by Definition 8.2, position j_0 is both a left and a right position. The class of compact orders is wide enough to include some of the best known character comparison orders; in particular KMP and BM character comparison orders can be considered as extreme cases of compact orders: in KMP order all positions are right positions, while in BM order all but position $j_m = m$ are left positions; CP character comparison order is a prototype of compact order.

Based on the theory developed in Section 3 we know that the computation of the shift function δ_C requires first the computation of the values $imin_C(k), 0 < k \leqslant m$. We can characterize function $imin_C$ by the autocorrelation matrix $\mathcal{M}_{\mathcal{W}}$ as follows:

Lemma 8.3. *Let $C = (j_0, \ldots, j_{m-1}, m)$ be a compact order. Then*

(a) *let j_i be a left position. Then $imin_C(k) = i$ if and only if $a_{j_i-k,j_i} = 1$ and $a_{j_i-k+p,j_i+p} = 0$ for all p such that $1 \leqslant p \leqslant i$;*

(b) *let j_i be a right position. Then $imin_C(k) = i$ if and only if $a_{j_i-k,j_i} = 1$ and $a_{j_i-k-p,j_i-p} = 0$ for all p such that $1 \leqslant p \leqslant min\{j_i - k, i\}$.*

Proof. (a) By Definition (3) of Section 3, $imin_C(k) = i$ if and only if $k \in N(j_i)$ and $k \notin N(j_t)$ for all t such that $0 \leqslant t < i$, i.e. if and only if $a_{j_i-k,j_i} = 1$. Since j_i is a left position in the compact order C, then the set of positions $j_0, \ldots, j_{i-1}$ covers a segment of pattern from position j_i to position $j_i + i$;

(b) The proof is similar to point (a), recalling that since j_i is a right position in the compact order C, then the set of positions $j_0, \ldots, j_i - 1$ covers a segment of the pattern from position $j_i - i$ to position $j_i - 1$. Note that we should consider only positions between $\max(k, j_i - i)$ and $j_i - 1$ since $k \notin N(j)$ for all j such that $j < k$. □

Lemma 8.3 shows that, in order to compute function $imin_C$, we need to compute 0-filled sequences in the downward diagonals of $\mathcal{M}_{\mathcal{W}}$. Functions *pref* and *postf* defined in the previous sections seem to fit well in such a contest.

We shall proceed by stages: we first describe which 0-filled sequences of $\mathcal{M}_{\mathcal{W}}$ need to be computed, then we implement such computations by using functions *pref* and *postf* and finally we show how to assign the correct values to function $imin_C$ according to the compact order $C = (j_0, \ldots, j_{m-1}, m)$. We describe the stages of the computation by indicating which operations need to be performed, leaving to the reader the implementation details.

The first stage can be described as follows:

(1) consider column j_0 in the upper triangle of $\mathcal{M}_{\mathcal{W}}$. Compute the length $F_1(k)$ of the maximal 0-filled postfix of the kth-downward diagonal ending in column j_0, for $k = 1$ to j_0;

(2) compute the length $F_2(k)$ of the maximal 0-filled prefix of the kth-downward diagonal for $k = j_0 + 1$ to m;

(3) compute the length $F_3(k)$ of the maximal 0-filled segment of the kth-downward diagonal for $k = 1$ to j_0 starting from column j_0.

In order to implement the above points, it is convenient to divide string $\mathscr{W}[0\ldots m-1]$ in two substrings $\sigma_1 = \mathscr{W}[0\ldots j_0]$ and $\sigma_2 = \mathscr{W}[j_0+1\ldots m-1]$. Then the implementation of (1)–(3) proceeds as follows:

(1) Compute function $postf_{\mathrm{BM}}$ on $\sigma_1\$$ for $k = 1$ to j_0 and set $F_1(k) = postf_{\mathrm{BM}}(k)$

(2) Compute function $pref_{\mathrm{KMP}}$ on $\mathscr{W}[0\ldots m]$ and for $k = j_0 + 1$ to m set $F_2(k) = pref_{\mathrm{KMP}}(k)$

(3) Compute function $pref_{\mathrm{KMP}}$ on $\sigma_2\sigma_1\$$ for $k = 1$ to m. Assign values to function F_3 for $k = 1$ to j_0 as follows: $F_3(k) = pref_{\mathrm{KMP}}(m-k)$ if $pref_{\mathrm{KMP}}(m-k) \neq k$; suppose $pref_{\mathrm{KMP}}(m-k) = k$: the maximal 0-filled segment of the kth-downward diagonal starting from column j_0 might be longer than $pref_{\mathrm{KMP}}(m-k)$. It implies that the 0-filled segment covers a segment of σ_2 and $F_3(k) = pref_{\mathrm{KMP}}(m-k) + pref_{\mathrm{KMP}}(k)$.

Finally we have to assign the correct values to function $imin_C$. We consider the permutation $\Pi = (\pi_0, \pi_1, \ldots, \pi_{m-1}, m)$ such that $\pi_{j_i} = i$ for all $i, 0 \leqslant i \leqslant m$ and we distinguish the following two cases:

- $k \in [1\ldots j_0]$. Then
  - (i) Consider function F_1; if $F_1(k) = j_0 - k + 1$, then the kth postfix of σ_1 is complete (i.e. we reached row zero) and the value of $imin_C(k)$ is the value given by point (ii) and (iii); otherwise $F_1(k) < j_0 - k + 1$. Consider the quantity $j_0 - F_1(k)$ and compute the index $\pi_{j_0-F_1(k)}$.
  - (ii) Consider the quantity $j_0 + F_3(k)$ and compute the index $\pi_{j_0+F_3(k)}$.
  - (iii) If $\pi_{j_0-F_1(k)} < \pi_{j_0+F_3(k)}$ then $imin_C(k) = \pi_{j_0-F_1(k)}$ else $imin_C(k) = \pi_{j_0+F_3(k)}$.
- $k \in [j_0 + 1\ldots m]$. Consider the quantity $k + F_2(k)$ and set $imin_C(k) = \pi_{k+F_2(k)}$.

The computation of function $imin_C$, as described above, requires $O(m)$ time since it uses the permutation Π and functions $pref$ and $postf$; note that the entries of the autocorrelation matrix $\mathscr{M}_{\mathscr{W}}$ are accessed only in the computation of the latter functions.

The last step to be performed consists in computing the shift function δ_C from function $imin_C$. We show how this can be done in $O(m)$ time. Theorem 3.4 shows how to compute δ_C on noholes. Let $\mathscr{W}[j_i]$ be a hole for the compact order C. If j_i is a right position, then $\delta_C(i) = j_i + 1$, i.e. the same approach of KMP character comparison order holds; if j_i is a left position, let $\mathscr{W}[l_i\ldots r_i]$ be the segment of the pattern covered by characters $\mathscr{W}[j_0], \ldots, \mathscr{W}[j_i]$ and let $pmin(i)$ be the minimum period of $\mathscr{W}[0\ldots r_i]$ such that $pmin(i) > l_i$. Then $\delta_C(i) = pmin(i)$.

The only concern is to compute the values of function $pmin$ in linear time. We show how to compute the values $pmin(i)$ for all $i = 0, \ldots, m-1$, irrespective of j_i be a left or right position by the following lines of code.

begin

$l := j_0$; $r := j_0$; $p := j_0 + 1$; $t := j_0 + 1$;

$\{p = pmin(0)$; $\mathscr{W}[l\ldots r] = \mathscr{W}[j_0]$; k is not period of $\mathscr{W}[0\ldots r]$ for all k such that $p < k \leqslant t\}$

```
for i := 0 to m − 1 do
        begin
        {p = pmin(i); W[l...r] is covered by W[j0],...,W[ji]; k is not period
         of W[l...r] for all k such that p < k ⩽ t}
        pmin[i] := p;
        if j_{i+1} "is a right position" then
            begin
            {j_{i+1} = r + 1}
            r := r + 1
            if pref_KMP[p] + p ⩽ r {otherwise pmin(i + 1) = pmin(i)}
                then begin
                        t := t + 1;
                        while pref_KMP[t] + t ⩽ r do t := t + 1;
                        p := t;
                        end
            end
        else
        {j_{i+1} = l − 1 is a left position}
            begin
            if pref_KMP[l] + l > r {otherwise pmin(i + 1) = pmin(i)}
            then p := l;
            l := l − 1
            end
        end
end
```

It is easy to verify that this algorithm is linear. The only trouble might be the nested *while* loop. However, the while loop increases the value of variable t and t cannot be greater than m.

After we computed the shift function δ_C for a specific compact order C we can compute the shift functions for many other character comparison orders in linear time and we can maximize or minimize the values of the shift functions, too. Indeed, the equivalent class defined by C includes, for instance, all the orders obtained moving forward one or more holes of C, as proved in Lemma 4.5. Moreover, for all the orders in the equivalent class that satisfy Theorem 3.4, we can compute the relative shift function in $O(m)$ time and maximize or minimize its values according to which situation described in Theorem 3.4 holds. Note that the linear work performed does not require any extra character comparison, except those to compute $imin_C$. These procedure can be applied to each compact order C, since compact orders are not equivalent in general.

Finally, note that the character comparison orders in the equivalent class of a compact order C are not necessarily compact; for example J_{KMP} is compact, but the equivalent order J_{Col} is not.

9. Concluding remarks and open problems

String matching algorithms use shift functions to slide the pattern along the text.

In this paper we studied the relations between the order J of comparing corresponding characters in the pattern and text strings and the relative shift function δ_J for a class $\mathscr{F}$ of on-line string matching algorithms. We gave a uniform definition of δ_J depending only on the character comparison order J and we characterized sets of equivalent character comparison orders for which the computation of the shift function is strictly related.

By introducing the class of compact orders we generalized some of the best known string matching algorithms and we provided numerous other character comparison orders for which the computation of the shift function is efficient.

There are few open problems that we like to mention:

1. Is $\frac{3}{2}n$ a tight bound on the number of character comparisons in the text search step for algorithms in the family $\mathscr{F}$?
2. What is the exact comparison complexity of function $imin_J$? How is the comparison complexity related to the character comparison order J?
3. Which are the classes, besides the compact orders, of comparison orders such that the computation of the shift function requires $O(m)$ time?
4. Is it possible to describe the classes in 3. (if any) in a uniform way?
5. Is it possible to reduce the number of character comparisons performed in the computation of function $imin_C, C$ being a compact order?

Acknowledgements

We are grateful to the anonymous referees for their comments which helped to improve the bound in Lemma 3.2 and the presentation of the paper.

References

[1] A.V. Aho, Algorithms for finding patterns in strings, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science* (Elsevier, Amsterdam, 1990) 257–300.

[2] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA., 1974).

[3] A. Apostolico and R. Giancarlo, The Boyer–Moore–Galil string searching strategies revisited, *SIAM J. Comput.* **15** (1986) 98–105.

[4] R. Baeza-Yates, Improved string searching, *Software Practice and Experience* **19**(3) (1989) 257–271.

[5] R.S. Boyer and J.S. Moore, A fast string searching algorithm, *Comm. ACM* **20** (1977) 762–772.

[6] D. Breslauer, L. Colussi and L. Toniolo, Tight comparison bounds for the string prefix-matching problem, *Inform. Process. Lett.* **47** (1993) 51–57.

[7] D. Breslauer, L. Colussi and L. Toniolo, On the exact complexity of the string prefix-matching problem, in: *Proc. European Symp. on Algorithms*, Vol. 855 (Springer, Berlin, Verlag, 1994) 483–494.

[8] D. Breslauer and Z. Galil, Efficient comparison based string matching, *J. Complexity* **32** (1993) 339–365.

[9] R. Cole, Tight bounds on the complexity of the Boyer–Moore pattern matching algorithm, In: *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, (1991) 224–233.
[10] R. Cole and R. Hariharan, Tighter bounds on the exact complexity of string matching, in: *Proc. 33rd IEEE Symp. on Foundations of Comput. Sci.*, (1992) 600–609.
[11] L. Colussi, Correctness and efficiency of string matching algorithms, *Inform. and Control* **95** (1991) 225–251.
[12] L. Colussi, Fastest pattern matching in strings, *J. Algorithms* **16** (1994) 163–189.
[13] L. Colussi, Z. Galil and R. Giancarlo, On the exact complexity of string matching, in: *Proc. 31st IEEE Symp. on Foundations of Computer Science* (1990) 135–143.
[14] T.H. Cormen, C.E. Leiserson and R.L. Rivest, *Introduction to Algorithms* (McGraw-Hill, New York, 1990).
[15] M. Crochemore, String-matching on ordered alphabets, *Theoret. Comput. Sci.* **92** (1992) 33–47.
[16] M. Crochemore and D. Perrin, Two-way string-matching, *J. Assoc. Comput. Mach.* **38**(3) (1991) 651–675.
[17] M. Crochemore and W. Rytter, Periodic prefixes in texts, in: R. Capocelli, A. De Santis and U. Vaccaro, eds., *Proc. of the Sequences '91 Workshop: Sequences II: Methods in Communication, Security and Computer Science* (Springer, Berlin, 1993) 153–165.
[18] M.C. Favaretto, Limiti superiori e inferiori per la complessità di una classe di algoritmi di string matching, Master's thesis, Dipartimento di Matematica Pura ed Applicata, Università di Padova, Padova, Italy, 1994.
[19] Z. Galil and R. Giancarlo, The exact complexity of string matching: upper bounds, *SIAM J. Comput.* **21**(3) (1992) 407–437.
[20] Z. Galil and J. Seiferas, Time-space-optimal string matching, *J. Comput. System Sci.* **26** (1983) 280–294.
[21] C. Hancart. On Simon string searching algorithm, *Inform. Process. Lett.* **47** (1993) 95–99.
[22] A. Hume and D. Sunday, Fast string searching, *Software Practice and Experience*, **21**(11) (1991) 1221–1248.
[23] D.E. Knuth, J.H. Morris and V.R. Pratt, Fast pattern matching in strings, *SIAM J. Comput.* **6** (1977) 322–350.
[24] M. Lothaire, *Combinatorics on Words* (Addison-Wesley, Reading, MA, 1983).
[25] I. Simon, String Matching Algorithms and Automata, in: Baeza-Yates and Ziviani, eds., *1st South-American Workshop on String Processing* (Belo Horizonte, Brasil, 1993).
[26] G.D.V. Smit, A Comparison of Three string matching algorithms, *Software Practice and Experience* **12**(1) (1982) 57–66.
[27] P. van Emde Boas, R. Kaas and E. Zijlstra, Design and implementation of an efficient priority queue, *Math. Systems Theory* **10** (1977) 99–127.