BRICS

Basic Research in Computer Science

On the Comparison Complexity of the String Prefix-Matching Problem

Dany Breslauer
Livio Colussi
Laura Toniolo

BRICS Report Series **RS-95-46**

ISSN 0909-0878 **August 1995**

See back inner page for a list of recent publications in the BRICS Report Series. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark

Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@daimi.aau.dk

BRICS publications are in general accessible through WWW and anonymous FTP:

`http://www.brics.aau.dk/BRICS/`
`ftp ftp.brics.aau.dk (cd pub/BRICS)`

On the Comparison Complexity of the String Prefix-Matching Problem

Dany Breslauer* Livio Colussi† Laura Toniolo‡

Abstract

In this paper we study the exact comparison complexity of the string prefix-matching problem in the deterministic sequential comparison model with equality tests. We derive almost tight lower and upper bounds on the number of symbol comparisons required in the worst case by on-line prefix-matching algorithms for any fixed pattern and variable text. Unlike previous results on the comparison complexity of string-matching and prefix-matching algorithms, our bounds are almost tight for any particular pattern.

We also consider the special case where the pattern and the text are the same string. This problem, which we call the *string self-prefix* problem, is similar to the pattern preprocessing step of the Knuth-Morris-Pratt string-matching algorithm that is used in several comparison efficient string-matching and prefix-matching algorithms, including in our new algorithm. We obtain roughly tight lower and upper bounds on the number of symbol comparisons required in the worst case by on-line self-prefix algorithms.

Our algorithms can be implemented in linear time and space in the standard uniform-cost random-access-machine model.

*BRICS – Basic Research in Computer Science, Centre of the Danish National Research Foundation, Department of Computer Science, University of Aarhus, DK-8000 Aarhus C, Denmark. Partially supported by the ESPRIT Basic Research Action Program of the EC under contract #7141 (ALCOM II). Part of the research reported in the paper was carried out while this author was visiting at the Istituto di Elaborazione dell'Informazione, Consiglio Nazionale delle Ricerche, Pisa, Italy, with the support of the European Research Consortium for Informatics and Mathematics postdoctoral fellowship.

†Dipartimento di Matematica Pura ed Applicata, Università di Padova, Via Belzoni 7, I-35131 Padova, Italy. Partially supported by "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo" of the Italian National Research Councile under grant number 89.00026.69.

‡Dipartimento di Matematica Pura ed Applicata, Università di Padova, Via Belzoni 7, I-35131 Padova, Italy. Parts of the research reported in this paper were carried out while this author was visiting at the Institut Gaspard Monge, Université de Marne-la-Vallée, Noisy-le-Grand, France, supported by "Borsa di studi per attività di perfeziomento all'estero" from the University of Padua, and at BRICS, Department of Computer Science, University of Aarhus, Aarhus, Denmark, supported by the Gini Foundation of Padua.

1 Introduction

In the *string prefix-matching* problem one is interested in finding the longest prefix of a pattern string $\mathcal{P}[1..m]$ that starts at each position of a text string $\mathcal{T}[1..n]$. The output of the problem is an integer array $\Phi[1..n]$, $0 \leq \Phi[t] \leq \min(m, n-t+1)$, such that for each text position t, $\mathcal{T}[t..t+\Phi[t]-1] = \mathcal{P}[1..\Phi[t]]$ and if $\Phi[t] < m$ and $t + \Phi[t] \leq n$, then $\mathcal{T}[t + \Phi[t]] \neq \mathcal{P}[\Phi[t] + 1]$.

The string prefix-matching problem is a natural generalization of the standard string-matching problem where only complete occurrences of the pattern are sought. The classical linear-time string-matching algorithm of Knuth, Morris and Pratt [32] can be easily adapted to solve the prefix-matching problem in the same time bounds without making additional symbol comparisons[1]. This observation was first made by Main and Lorentz [35] who used a prefix-matching algorithm to detect repetitions in strings. In the parallel setting, Galil's [22] string-matching algorithm also solves the prefix-matching problem. Breslauer [5] and Hariharan and Muthukrishnan [31] gave more efficient parallel algorithms and recently Gąsieniec and Park [27] obtained a time-work optimal parallel prefix-matching algorithm. Prefix-matching algorithms have also been used in the sequential two-dimensional pattern-matching algorithms of Amir, Benson and Farach [2] and Galil and Park [25] and in an early version of the parallel two-dimensional pattern-matching algorithm of Cole et al. [11].

In this paper we study the exact number of comparisons performed by deterministic sequential prefix-matching algorithms that have access to the input strings by pairwise symbol comparisons that test for equality. This work was motivated by recent interest in the exact comparison complexity of the string-matching problem and is continuation of an earlier work by the same authors [7].

In a sequence of papers on the number of symbol comparisons required in string-matching algorithms Colussi [14], Galil and Giancarlo [24], Breslauer and Galil [9] and Cole and Hariharan [12] improved the upper bounds, while Galil and Giancarlo [23], Zwick and Paterson [38], Cole and Hariharan [12] and Cole et al. [13] tightened the lower bounds. Currently, the complexity of the string-matching problem is determined almost exactly with an upper bound of $n + (8/3m)(n - m)$ comparisons and a lower bound of $n + (9/4m)(n - m)$ comparisons[2]. There are numerous other papers that study the exact number of comparisons required in the string-matching problem, in order statistics problems and in various other problems. Most relevant perhaps is the literature on constant-space linear-time string-matching algorithms [6, 17, 20, 21, 26, 28] and

[1]Since complete occurrences of the pattern cannot start at text positions larger than $n - m + 1$, the string-matching algorithm can stop before reaching the end of the text. The prefix-matching algorithm must continue until the end of the text and therefore, it might make at most m extra comparisons.

[2]These bounds are on the number of comparisons made by on-line algorithms in the text processing step, after an unaccounted pattern preprocessing. There is a larger gap between the lower and upper bounds for off-line algorithms.

on variants of the Boyer-Moore string-matching algorithm [3, 4, 10, 15, 18, 19, 29, 32, 33].

Boyer and Moore [4] showed that in the string-matching problem it is not always necessary to examine all n text symbols. On the other hand, Rivest [37] has proved that in the worst case any string-matching algorithm must always examine at least $n - m + 1$ text symbols. Clearly, any algorithm that solves the prefix-matching problem must examine all n text symbols since it must determine if each text symbol is equal to the first pattern symbol. Note that if the input alphabet is known to contain only two symbols, then the prefix-matching problem requires exactly n comparisons, since inequality to one alphabet symbol implies equality to the other.

Breslauer, Colussi and Toniolo [7] defined on-line prefix-matching algorithms and presented a family of algorithms that make at most $\lfloor (2 - 1/m)n \rfloor$ symbol comparisons. They also gave a tight lower bound for any prefix-matching algorithm that has to match the pattern 'ab^{m-1}'. These results imply that the two similar string-matching and prefix-matching problems have intrinsically different asymptotic comparison complexities, approaching n and $2n$, respectively, as m grows. However, on-line prefix-matching algorithms have many similarities to the on-line string-matching algorithms given by Colussi [14] and Breslauer and Galil [9]. The definition of on-line prefix-matching algorithms also coincides with the finite automata approach to string matching. In an analysis of Simon's automata based string-matching algorithm Hancart [30] independently obtained the same bounds that were given in [7] for on-line prefix-matching algorithms (see also [16]). The new results about on-line prefix-matching algorithms presented in this paper apply as well to automata based string-matching, and thus extend also Hancart's work.

Our main effort in this paper is to determine $c^{\mathcal{P}[1..m]}_{on-line}(n)$, the number of symbol comparisons required in the worst case by on-line prefix-matching algorithms to match a fixed pattern $\mathcal{P}[1..m]$ in a variable text of length n. We determine $c^{\mathcal{P}[1..m]}_{on-line}(n)$ almost exactly by showing that for any pattern $\mathcal{P}[1..m]$, there exists a constant $\mathcal{C}^{\mathcal{P}[1..m]}_{on-line}$, $1 \leq \mathcal{C}^{\mathcal{P}[1..m]}_{on-line} \leq 2 - 1/m$, such that,

$$\mathcal{C}^{\mathcal{P}[1..m]}_{on-line} \times (n - m) + m \leq c^{\mathcal{P}[1..m]}_{on-line}(n) \leq \mathcal{C}^{\mathcal{P}[1..m]}_{on-line} \times n.$$

And thus,

$$\lim_{n \to \infty} \frac{c^{\mathcal{P}[1..m]}_{on-line}(n)}{n} = \mathcal{C}^{\mathcal{P}[1..m]}_{on-line}.$$

The upper bound of at most $\mathcal{C}^{\mathcal{P}[1..m]}_{on-line} \times n$ comparisons is achieved by an algorithm that takes linear time and uses linear space. Unlike previous publications on the comparison complexity of string-matching and prefix-matching algorithms that give worst case bounds for a specific pattern, our bounds are almost tight for any given pattern.

We then consider the special case where the pattern and the text are the same string. This problem, which we call the *self-prefix* problem, is similar to the *failure function* that is computed in the preprocessing step of the Knuth-Morris-Pratt [32] string-matching algorithm[3]. The Knuth-Morris-Pratt failure function is used in various string-matching algorithms and also in the pattern preprocessing step of our prefix-matching algorithm. Using the techniques we develop for the prefix-matching problem, we obtain an on-line algorithm for the self-prefix problem that makes at most $2m - \lceil\sqrt{2m}\rceil$ symbol comparisons. We also prove a roughly tight lower bound (up to an additive constant 2) on the number of symbol comparisons required by such an algorithm, and thus, determine the worst case comparison complexity of the on-line self-prefix problem and of computing the Knuth-Morris-Pratt failure function. The self-prefix algorithm and the whole pattern preprocessing step of the prefix-matching algorithm take linear time and use linear space.

Finally, we consider general off-line prefix-matching algorithms. Such algorithms are more difficult to analyze since they have more liberties about the way they might proceed comparing the input symbols. We were unable to obtain tight bounds for off-line algorithms. However, we show that there exist pattern strings for which off-line algorithms require significantly fewer symbol comparisons than on-line algorithms.

The paper is organized as follows. In Section 2 we give the lower and upper bounds for on-line prefix-matching algorithms. In Section 3 we prove that off-line algorithms can be superior to on-line algorithms in some cases. In Section 4 we present the algorithm for the self-prefix problem and in Section 5 we show how to implement the prefix-matching and self-prefix algorithms in the standard random-access-machine model. Conclusions and open problems are given in Section 6.

2 On-line prefix-matching

The discussion below proceeds in the comparison model where only comparisons of input symbols are counted and all other computation is free. We assume that our algorithms can obtain complete information about the pattern $\mathcal{P}[1..m]$ in an unaccounted pattern preprocessing step that might compare even all $\binom{m}{2}$ pairs of pattern symbols. In Section 5 we discuss the efficient implementation of our algorithms, including the pattern preprocessing, in the standard random-access-machine computational model [1].

Recall the definition of on-line string prefix-matching algorithms given by Breslauer, Colussi and Toniolo [7].

[3] In fact, these two problems are equivalent with respect to the number of comparisons they require; the output of either one can be computed from the output of the other in linear-time without any extra comparisons of the input symbols.

Definition 2.1 *A prefix-matching algorithm is* on line *if before comparing the text symbol $\mathcal{T}[t]$ it has determined if the pattern prefixes that start at text positions l, for $1 \le l < t$, terminate before text position t.*

Comparison efficient on-line prefix-matching algorithms are restricted about the choice of comparisons they can make. It is not difficult to verify that on-line algorithms that compare pairs of text symbols are not more efficient than those that compare only pattern symbols to text symbols.

Let $\mathcal{K}^t = \left\{k_i^t \mid t - m < k_0^t < \cdots < k_{q_t}^t = t\right\}$ be the set of all text positions for which $\Phi[k_i^t]$ can not be determined without examining $\mathcal{T}[t]$. Namely, $\mathcal{T}[k_i^t..t-1] = \mathcal{P}[1..t-k_i^t]$ and $\mathcal{T}[t]$ must be compared in order to check whether $\Phi[k_i^t] = t-k_i^t$ or $\Phi[k_i^t] > t-k_i^t$. Then, all comparisons at text position t must be between $\mathcal{T}[t]$ and the pattern symbols $\mathcal{P}[t-k_i^t+1]$ or otherwise can be answered by an adversary as unequal without giving an algorithm any useful information, provided that the text alphabet is large enough.

Clearly, $\mathcal{T}[t]$ has to be compared either until it is found to be equal to some symbol $\mathcal{P}[t-k_i^t+1]$ or until it is known to be different from all these symbols. Thus, the only difference between the comparison efficient on-line prefix-matching algorithms we consider next, is the *order* according to which $\mathcal{T}[t]$ is compared to the pattern symbols $\mathcal{P}[t-k_i^t+1]$.

2.1 Periods in strings

Periods are regularities of strings that are exploited virtually in all efficient string-matching algorithms. In this section we give some basic properties of periods and define the notation that we use throughout the paper. For an extensive treatment of periods and their properties see Lothaire's book [34].

Definition 2.2 *A string $\mathcal{S}[1..h]$ has a period of length π if $\mathcal{S}[g] = \mathcal{S}[g+\pi]$, for $g = 1, \ldots, h-\pi$. We define the set $\Pi^{\mathcal{S}[1..h]} = \left\{\pi_i^{\mathcal{S}} \mid 0 = \pi_0^{\mathcal{S}} < \cdots < \pi_{p_h}^{\mathcal{S}} = h\right\}$ to be the set of all period lengths of $\mathcal{S}[1..h]$. $\pi_1^{\mathcal{S}}$, the smallest non-trivial period length of $\mathcal{S}[1..h]$, is called* the period *of $\mathcal{S}[1..h]$.*

A string $\mathcal{S}[1..l]$, such that $\Pi^{\mathcal{S}[1..l]} = \{0, l\}$, is sometimes called *unbordered*. The simple properties of periods given next follow immediately from the definition.

Proposition 2.3 *The set $\Pi^{\mathcal{S}[1..l]}$ satisfies: $\Pi^{\mathcal{S}[1..l]} \setminus \{l\} \subseteq \Pi^{\mathcal{S}[1..l-1]}$.*

In the last proposition the inclusion in the opposite direction usually does not hold; the periods $\pi \in \Pi^{\mathcal{S}[1..l-1]}$, such that $\pi \notin \Pi^{\mathcal{S}[1..l]}$, are said to *terminate* at position l.

Proposition 2.4 *For any $\pi \in \Pi^{\mathcal{S}[1..l]}$,*

$$\Pi^{\mathcal{S}[1..l]} \cap \{\pi, \ldots, l\} = \left\{\pi + \tau \mid \tau \in \Pi^{\mathcal{S}[1..l-\pi]}\right\}.$$

Periods are connected to on-line prefix-matching algorithms by the following observation.

Lemma 2.5 *The members of the set $\mathcal{K}^t$ correspond to the periods of the pattern prefix $\mathcal{P}[1..t-k_0^t]$ by the following relation:*

$$\mathcal{K}^t = \left\{ k_0^t + \pi \mid \pi \in \Pi^{\mathcal{P}[1..t-k_0^t]} \right\}.$$

Proof. By the definitions, if $k_i^t \in K^t$, then $\mathcal{T}[k_i^t..t-1] = \mathcal{P}[1..t-k_i^t]$. In particular, $\mathcal{T}[k_0^t..t-1] = \mathcal{P}[1..t-k_0^t]$ and therefore $\mathcal{P}[1..t-k_i^t] = \mathcal{P}[k_i^t - k_0^t + 1..t-k_0^t]$, establishing that $k_i^t - k_0^t \in \Pi^{\mathcal{P}[1..t-k_0^t]}$. Similarly, if $\pi \in \Pi^{\mathcal{P}[1..t-k_0^t]}$, then by the definition of periods $\mathcal{P}[1..t-k_0^t-\pi] = \mathcal{P}[\pi+1..t-k_0^t]$. Since $\mathcal{T}[k_0^t..t-1] = \mathcal{P}[1..t-k_0^t]$ we get that $\mathcal{T}[k_0^t+\pi..t-1] = \mathcal{P}[1..t-k_0^t-\pi]$, establishing that $k_0^t + \pi \in \mathcal{K}^t$. $\square$

By Lemma 2.5, $\{\mathcal{P}[t-k_i^t+1] \mid k_i^t \in \mathcal{K}^t\}$, the *set* of all the pattern symbols that $\mathcal{T}[t]$ might be compared to, is equal to $\left\{\mathcal{P}[l-\pi] \mid \pi \in \Pi^{\mathcal{P}[1..l-1]}\right\}$, for $l = t-k_0^t+1$. We define $\Sigma_l^{\mathcal{P}} = \left\{\mathcal{P}[l-\pi] \mid \pi \in \Pi^{\mathcal{P}[1..l-1]}\right\}$, for $l = 1, \ldots, m$. Notice that $\Sigma_l^{\mathcal{P}}$ is a set, thus containing distinct symbols and identifying equal symbols. One can visualize this definition by aligning copies of the pattern $\mathcal{P}$ with the text starting at the positions $k_i^t \in \mathcal{K}^t$; then the elements of $\Sigma_{t-k_0^t+1}^{\mathcal{P}}$ are the pattern symbols that are aligned with the text symbol $\mathcal{T}[t]$.

The following property follows immediately from the definitions.

Proposition 2.6 *For any $\pi \in \Pi^{\mathcal{P}[1..l-1]}$, $\Sigma_{l-\pi}^{\mathcal{P}} \subseteq \Sigma_l^{\mathcal{P}}$. Furthermore, if $\pi = \pi_1^{\mathcal{P}[1..l-1]}$, then $\Sigma_{l-\pi}^{\mathcal{P}} \cup \{\mathcal{P}[l]\} = \Sigma_l^{\mathcal{P}}$. (Notice that $\Sigma_{l-\pi}^{\mathcal{P}} = \Sigma_l^{\mathcal{P}}$ if and only if $\Pi^{\mathcal{P}[1..l]} \neq \{0, l\}$.)*

Given some symbol $\sigma \in \Sigma_l^{\mathcal{P}}$, we define two functions that give the smallest and the largest period lengths of $\mathcal{P}[1..l-1]$ that introduce the symbol σ into the set $\Sigma_l^{\mathcal{P}}$:

$$\pi_{\text{first}}^l(\sigma) \quad = \quad \min\left\{\pi \mid \pi \in \Pi^{\mathcal{P}[1..l-1]} \text{ and } \mathcal{P}[l-\pi] = \sigma\right\},$$

and,

$$\pi_{\text{last}}^l(\sigma) \quad = \quad \max\left\{\pi \mid \pi \in \Pi^{\mathcal{P}[1..l-1]} \text{ and } \mathcal{P}[l-\pi] = \sigma\right\}.$$

The function $\pi_{\text{first}}^l(\sigma)$ is important in on-line algorithms since it determines what will be k_0^{t+1} as a function of k_0^t and $\mathcal{T}[t]$.

Lemma 2.7 *If $\mathcal{T}[t] \in \Sigma_{t-k_0^t+1}^{\mathcal{P}}$, then $k_0^{t+1} = k_0^t + \pi_{\text{first}}^{t-k_0^t+1}(\mathcal{T}[t])$, except if $t - k_0^t + 1 = m$ and $\mathcal{T}[t] = \mathcal{P}[m]$, where $k_0^{t+1} = k_0^t + \pi_1^{\mathcal{P}[1..m]}$.*

Proof. Given k_0^t and $\mathcal{T}[t] \in \Sigma^{\mathcal{P}}_{t-k_0^t+1}$, by Lemma 2.5,

$$k_0^{t+1} = \min\left\{k_i \mid k_i \in \mathcal{K}^t \text{ and } \mathcal{T}[t] = \mathcal{P}[t-k_i^t+1]\right\} = k_0^t + \pi_{\text{first}}^{t-k_0^t+1}(\mathcal{T}[t]),$$

with the exception that if a complete occurrence of the pattern is discovered, namely if $t - k_0^t + 1 = m$ and $\mathcal{T}[t] = \mathcal{P}[m]$, then $k_0^t \notin \mathcal{K}^{t+1}$ and $k_0^{t+1} = k_0^t + \pi_1^{\mathcal{P}[1..m]}$. $\square$

The function $\pi^l_{\text{last}}(\sigma)$ is used in the development of comparison efficient on-line algorithms. Its main properties are summarized in the following lemma.

Lemma 2.8 *For any $l = 1, \ldots, m$, and $\pi \in \Pi^{\mathcal{P}[1..l-1]}$,*

$$\pi^l_{\text{last}}(\sigma) = \pi + \pi^{l-\pi}_{\text{last}}(\sigma) \qquad \textit{for } \sigma \in \Sigma^{\mathcal{P}}_{l-\pi}$$

and

$$\pi^l_{\text{last}}(\sigma) < \pi^l_{\text{last}}(\tau) \qquad \textit{for } \sigma \in \Sigma^{\mathcal{P}}_l \setminus \Sigma^{\mathcal{P}}_{l-\pi} \textit{ and } \tau \in \Sigma^{\mathcal{P}}_{l-\pi}.$$

Proof. By Proposition 2.6, $\pi^l_{\text{last}}(\sigma)$ is defined for $\sigma \in \Sigma^{\mathcal{P}}_{l-\pi}$ and by Proposition 2.4, $\pi^l_{\text{last}}(\sigma) = \pi + \pi^{l-\pi}_{\text{last}}(\sigma) \in \Pi^{\mathcal{P}[1..l-1]}$. By Proposition 2.4, if $\pi^l_{\text{last}}(\sigma) \geq \pi$, then $\sigma \in \Sigma^{\mathcal{P}}_{l-\pi}$. Thus, if $\tau \in \Sigma^{\mathcal{P}}_{l-\pi}$, then $\pi^l_{\text{last}}(\tau) \geq \pi$, and if $\sigma \in \Sigma^{\mathcal{P}}_l \setminus \Sigma^{\mathcal{P}}_{l-\pi}$, then $\pi^l_{\text{last}}(\sigma) < \pi$. $\square$

The following technical lemmas will be used throughout the paper.

Lemma 2.9 *The set $\Pi^{\mathcal{P}[1..l-1]}$ is disjointly partitioned according to $\Sigma^{\mathcal{P}}_l$:*

$$\Pi^{\mathcal{P}[1..l-1]} = \bigcup_{\sigma \in \Sigma^{\mathcal{P}}_l} \left\{\pi + \tau \mid \pi = \pi^l_{\text{first}}(\sigma) \textit{ and } \tau \in \Pi^{\mathcal{P}[1..l-\pi]} \textit{ and } \tau < l - \pi\right\}.$$

Proof. Define for $\sigma \in \Sigma^{\mathcal{P}}_l$,

$$\Pi^l_\sigma = \left\{\pi + \tau \mid \pi = \pi^l_{\text{first}}(\sigma) \text{ and } \tau \in \Pi^{\mathcal{P}[1..l-\pi]} \text{ and } \tau < l - \pi\right\}.$$

By Proposition 2.3 and Proposition 2.4, $\Pi^l_\sigma \subseteq \Pi^{\mathcal{P}[1..l-1]}$, for all $\sigma \in \Sigma^{\mathcal{P}}_l$. In addition, if $\hat{\pi} \in \Pi^l_\sigma$, then $\mathcal{P}[l - \hat{\pi}] = \sigma$, establishing that the sets Π^l_σ are disjoint for different σ. On the other hand, if $\hat{\pi} \in \Pi^{\mathcal{P}[1..l-1]}$, then $\hat{\pi} \in \Pi^l_{\mathcal{P}[l-\hat{\pi}]}$. $\square$

Lemma 2.10 *For any $\pi \in \Pi^{\mathcal{P}[1..l]}$ and $0 \leq h \leq l - \pi$,*

$$\sum_{g=h+1}^{h+\pi} |\Sigma^{\mathcal{P}}_g| \leq 2\pi - 1.$$

Proof. Observe that if $\mathcal{P}[g-\pi] \neq \mathcal{P}[g]$, then $\pi \notin \Pi^{\mathcal{P}[1..g]}$. By Proposition 2.3,

$$\begin{aligned} |\Sigma_g^{\mathcal{P}}| &= |\left\{\mathcal{P}[g-\pi] \mid \pi \in \Pi^{\mathcal{P}[1..g-1]}\right\}| \\ &= 1 + |\left\{\mathcal{P}[g-\pi] \mid \pi \in \Pi^{\mathcal{P}[1..g-1]}\right\} \setminus \{\mathcal{P}[g]\}| \\ &\leq 2 + |\Pi^{\mathcal{P}[1..g-1]}| - |\Pi^{\mathcal{P}[1..g]}|. \end{aligned}$$

Thus, for any π, not necessarily a period length, such that $1 \leq \pi \leq l$ and $0 \leq h \leq l - \pi$,

$$\sum_{g=h+1}^{h+\pi} |\Sigma_g^{\mathcal{P}}| \leq 2\pi + |\Pi^{\mathcal{P}[1..h]}| - |\Pi^{\mathcal{P}[1..h+\pi]}|.$$

If $\pi \in \Pi^{\mathcal{P}[1..l]} \setminus \{0\}$, then by Proposition 2.4, $|\Pi^{\mathcal{P}[1..h]}| \leq |\Pi^{\mathcal{P}[1..h+\pi]}| - 1$, establishing the claim. $\square$

The following two lemmas describe the relations between the lengths of the pattern prefixes and the maximal delay.

Lemma 2.11 *Let* $d_h = \max\left\{|\Sigma_l^{\mathcal{P}}| \mid l = 1, \ldots, h\right\}$. *Then,*

$$2^{d_h} - 1 \leq \sum_{l=1}^{h} |\Sigma_l^{\mathcal{P}}| \leq 2h - 1.$$

Proof. The right side inequality is a special case of Lemma 2.10. The left side inequality is proved by induction on h. By Proposition 2.6, $d_h \leq d_{h-1} + 1$. The basis of the induction for $h = 1$ holds since $|\Sigma_1^{\mathcal{P}}| = 1$ and $d_1 = 1$. The inductive hypothesis clearly holds if $d_h = d_{h-1}$. So it remains to prove the claim if $d_h = d_{h-1}+1$. Let $\pi = \pi_1^{\mathcal{P}[1..h-1]}$. Then, by Proposition 2.6, $d_\pi = d_{h-\pi} = d_{h-1}$, and by the inductive hypothesis,

$$2^{d_h} - 1 = 2^{d_\pi} + 2^{d_{h-\pi}} - 1 \leq \sum_{l=1}^{\pi} |\Sigma_l^{\mathcal{P}}| + \sum_{l=1}^{h-\pi} |\Sigma_l^{\mathcal{P}}| + 1 \leq \sum_{l=1}^{h} |\Sigma_l^{\mathcal{P}}|. \quad \square$$

Lemma 2.12 *Let* $d_h = \max\left\{|\Sigma_l^{\mathcal{P}}| \mid l = 1, \ldots, h\right\}$. *Then, for* $g \geq h$,

$$|\Sigma_g^{\mathcal{P}}| \leq d_h + \lfloor \log_2 \frac{2g}{h+1} \rfloor.$$

Proof. We prove by induction on g, for $g \geq h \geq 1$, that,

$$d_g \leq d_h + \lfloor \log_2 \frac{2g}{h+1} \rfloor.$$

By Proposition 2.6, $d_g \leq d_{g-1}+1$. It suffices to prove the claim above only for those indices $g > h$, such that $d_g = d_{g-1}+1$. Let $g_1, g_2, \ldots,$ be the sequence of these indices. The basis of the induction for g_1 clearly holds since:

$$d_{g_1} = d_h + 1 \leq d_h + \lfloor \log_2 \frac{2g_1}{h+1} \rfloor.$$

Let $\pi = \pi_1^{\mathcal{P}[1..g_i-1]}$. By Proposition 2.6 and by the inductive hypothesis, $d_{g_i-1} = d_{g_{i-1}} = d_{g_i-\pi}$, and thus $g_i - \pi \geq g_{i-1}$. But by Proposition 2.6, also $\pi \geq g_{i-1}$. Therefore, $g_i \geq 2g_{i-1}$, establishing the claim. □

Example. The *delay* at text position t, $|\Sigma^{\mathcal{P}}_{t-k_0^t+1}|$, is the maximal number of comparisons that an algorithm will have to make at this text position. Define the strings $\Delta_h[1..2^h]$ over the alphabet $\{\sigma_0, \ldots, \sigma_h\}$ as:

$$\Delta_h[g] = \sigma_{\max\{k \ \mid \ 2^k \text{ divides } g\}}.$$

An equivalent recursive definition is:

$$\begin{aligned} \Delta_0 &= \sigma_0 \quad \text{and} \\ \Delta_h &= \Delta_{h-1}[1..2^{h-1}]\ \Delta_{h-1}[1..2^{h-1}-1]\ \sigma_h. \end{aligned}$$

These are the only strings for which Lemma 2.11 is satisfied with equality in both parts. They have been used by Hancart [30] and Breslauer and Galil [9] as worst case examples for the proportion between their length 2^h and the delay $|\Sigma^{\Delta_h}_{2^h}| = h+1$.

2.2 Static algorithms

We define a subclass of the on-line algorithms that we call *static* algorithms. These algorithm are restricted enough to be easy to analyze, but still general enough to draw conclusions about the performance of on-line algorithms from their analysis.

Definition 2.13 *An on-line prefix-matching algorithm is said to be* static *if the order according to which the symbols in $\Sigma^{\mathcal{P}}_{t-k_0^t+1}$ are compared to the text symbol $\mathcal{T}[t]$ depends only on $t-k_0^t+1$.*

Since in a static algorithm $\mathcal{A}$ the order of comparisons depends only on $l = t - k_0^t + 1$, it will be defined by the functions:

$$\Lambda_{\mathcal{A},l}(h) : \{1, \ldots, |\Sigma^{\mathcal{P}}_l|\} \longmapsto \Sigma^{\mathcal{P}}_l \qquad \text{for } l = 1, \ldots, m,$$

where the algorithm $\mathcal{A}$ compares $\mathcal{T}[t]$ first to $\Lambda_{\mathcal{A},l}(1)$, then to $\Lambda_{\mathcal{A},l}(2)$ and so on. The number of comparisons that algorithm $\mathcal{A}$ makes to discover that $\mathcal{T}[t] = \sigma$, for some symbol $\sigma \in \Sigma^{\mathcal{P}}_l$, is $\Lambda^{-1}_{\mathcal{A},l}(\sigma)$. As static algorithms depend on the particular pattern, we shall denote by $\mathcal{A}(\mathcal{P}[1..m])$ the static algorithm $\mathcal{A}$ for the pattern string $\mathcal{P}[1..m]$. We ommit the pattern from our notation when it is clear from the context what it is.

Example. The Knuth-Morris-Pratt string-matching algorithm compares the symbols in $\Sigma_l^{\mathcal{P}} = \left\{\mathcal{P}[l-\pi] \mid \pi \in \Pi^{\mathcal{P}[1..l-1]}\right\}$, in *increasing* order of the periods π, sometimes repeating unnecessary comparisons. We define the static prefix-matching algorithm KMP to proceed in the spirit of the Knuth-Morris-Pratt algorithm, comparing the symbols $\mathcal{P}[l-\pi]$ in *increasing* order of the periods π, skipping symbols that were already compared. The order of comparisons $\Lambda_{KMP,l}(h)$ is defined such that,

$$\begin{aligned} &\pi^l_{\text{first}}(\Lambda_{KMP,l}(h)) < \pi^l_{\text{first}}(\Lambda_{KMP,l}(g)) \\ &\qquad\qquad \mathit{for} \quad l = 1, \ldots, m, \quad \text{and} \quad 1 \le h < g \le |\Sigma_l^{\mathcal{P}}|. \end{aligned}$$

2.3 The optimization problem

In this section we describe the method we use to evaluate the performance of static prefix-matching algorithms and define a measure to compare the relative efficiency of algorithms.

Define the *cost function* $\Omega_{\mathcal{A}}(l)$ to reflect the number of comparisons that the static algorithm $\mathcal{A}$ makes to match the pattern prefix $\mathcal{P}[1..l]$:

$$\Omega_{\mathcal{A}}(l) = \begin{cases} 0 & l = 0 \\ \Omega_{\mathcal{A}}(l-1) + \Lambda^{-1}_{\mathcal{A},l}(\mathcal{P}[l]) & l = 1, \ldots, m. \end{cases}$$

The goal is to bound the number of comparisons made by $\mathcal{A}$ by an expression of the form $\mathcal{C}_{\mathcal{A}} \times n$, for some constant $\mathcal{C}_{\mathcal{A}}$ that will be determined later. When the algorithm reaches text position t, just before comparing the text symbol $\mathcal{T}[t]$, we maintain inductively that the number of comparisons made so far is at most $\mathcal{C}_{\mathcal{A}} \times (k_0^t - 1) + \Omega_{\mathcal{A}}(t - k_0^t)$. This bound obviously holds initially at text position $t = 1$. However, when the algorithm advances to the next text position the term $\Omega(t+1-k_0^{t+1})$ might not account for all the comparisons. We shall bound the excess number of comparisons by $\mathcal{C}_{\mathcal{A}} \times (k_0^{t+1} - k_0^t)$, in order to maintain the inductive claim.

Let $l = t - k_0^t + 1$. If the algorithm discovers that $\mathcal{T}[t] = \sigma$, for $\sigma \in \Sigma_l^{\mathcal{P}}$, then it has made $\Lambda^{-1}_{\mathcal{A},l}(\sigma)$ comparisons at this text position. If $\sigma = \mathcal{P}[l]$ and $l < m$, then by Lemma 2.7, $k_0^{t+1} = k_0^t$, and the inductive hypothesis still holds as the cost function accounts for these comparisons.

However, if $\sigma \neq \mathcal{P}[l]$, then by Lemma 2.7, $k_0^{t+1} = k_0^t + \pi^l_{\text{first}}(\sigma)$ and only $\Omega_{\mathcal{A}}(l - \pi^l_{\text{first}}(\sigma))$ comparisons will be accounted by the cost function. To maintain the inductive hypothesis, we require that the remaining $\Omega_{\mathcal{A}}(l-1) + \Lambda^{-1}_{\mathcal{A},l}(\sigma) - \Omega_{\mathcal{A}}(l - \pi^l_{\text{first}}(\sigma))$ comparisons are also accounted by imposing the following constraint on $\mathcal{C}_{\mathcal{A}}$:

$$\frac{\Omega_{\mathcal{A}}(l-1)+\Lambda_{\mathcal{A},l}^{-1}(\sigma)-\Omega_{\mathcal{A}}(l-\pi_{\mathrm{first}}^{l}(\sigma))}{\pi_{\mathrm{first}}^{l}(\sigma)} \leq \mathcal{C}_{\mathcal{A}}. \tag{1}$$

If the algorithm concludes that $\mathcal{T}[t] \neq \sigma$, for all $\sigma \in \Sigma_l^{\mathcal{P}}$, then $k_0^{t+1} = k_0^t + l = t+1$ and there are $\Omega_{\mathcal{A}}(l-1) + |\Sigma_l^{\mathcal{P}}|$ comparisons that will not be accounted by the cost function. To maintain the inductive hypothesis in this case, we make certain that these comparisons are also accounted by requiring that:

$$\frac{\Omega_{\mathcal{A}}(l-1)+|\Sigma_l^{\mathcal{P}}|}{l} \leq \mathcal{C}_{\mathcal{A}}. \tag{2}$$

In addition, if the algorithm discovers that $\mathcal{T}[t] = \mathcal{P}[l]$ and the end of the pattern is reached, that is $l = m$, then by Lemma 2.7, $k_0^{t+1} = k_0^t + \pi_1^{\mathcal{P}}$ and only $\Omega_{\mathcal{A}}(m - \pi_1^{\mathcal{P}})$ comparisons will be accounted by the cost function. To maintain the inductive hypothesis we require also that:

$$\frac{\Omega_{\mathcal{A}}(m)-\Omega_{\mathcal{A}}(m-\pi_1^{\mathcal{P}})}{\pi_1^{\mathcal{P}}} \leq \mathcal{C}_{\mathcal{A}}. \tag{3}$$

Finally, when the end of the text is reached, that is when $t = n+1$, the number of comparisons made is bounded by the inductive hypothesis by $\mathcal{C}_{\mathcal{A}} \times (k_0^t - 1) + \Omega_{\mathcal{A}}(t - k_0^t)$. Since $\Omega_{\mathcal{A}}(h)/h \leq (\Omega_{\mathcal{A}}(h-1) + |\Sigma_h^{\mathcal{P}}|)/h \leq \mathcal{C}_{\mathcal{A}}$, for $h = n - k_0^{n+1} + 1$, by Inequality 2, we get that the number of comparisons made is at most $\mathcal{C}_{\mathcal{A}} \times n$, establishing the inductive claim.

For any static algorithm $\mathcal{A}$ let the characteristic constant $\mathcal{C}_{\mathcal{A}}$ be the smallest constant satisfying the three inequalities above. Inequality 1 has to be satisfied at pattern positions $l = 1, \cdots, m$, and for all $\sigma \in \Sigma_l^{\mathcal{P}} \setminus \{\mathcal{P}[l]\}$; Inequality 2 at pattern positions $l = 1, \cdots, m$; Inequality 3 does not depend on the pattern position. Note that by the same line of reasoning, an adversary can force a static algorithm $\mathcal{A}$ to make at least $\mathcal{C}_{\mathcal{A}} \times (n-m) + m$ comparisons. Thus, as algorithms with smaller characteristic constants make fewer comparisons in the worst case as n grows, the characteristic constant can be used to compare the relative efficiency of static prefix-matching algorithms. The constraints on $\mathcal{C}_{\mathcal{A}}$ are summarized in the following lemma.

Lemma 2.14 *The number of comparisons an algorithm $\mathcal{A}$ makes while scanning the text $\mathcal{T}[1..n]$ is at most $\mathcal{C}_{\mathcal{A}} \times n$, where the characteristic constant*

$$\mathcal{C}_{\mathcal{A}} = \max \left\{ \begin{array}{l} \frac{\Omega_{\mathcal{A}}(l-1)+\Lambda_{\mathcal{A},l}^{-1}(\sigma)-\Omega_{\mathcal{A}}(l-\pi_{\mathrm{first}}^{l}(\sigma))}{\pi_{\mathrm{first}}^{l}(\sigma)} \\ \qquad\qquad \textit{for } l = 1, \ldots, m, \ \textit{and } \sigma \in \Sigma_l^{\mathcal{P}} \setminus \{\mathcal{P}[l]\}\,; \\ \frac{\Omega_{\mathcal{A}}(l-1)+|\Sigma_l^{\mathcal{P}}|}{l} \ \textit{for } l = 1, \ldots, m; \qquad \frac{\Omega_{\mathcal{A}}(m)-\Omega_{\mathcal{A}}(m-\pi_1^{\mathcal{P}})}{\pi_1^{\mathcal{P}}} \end{array} \right\}.$$

Notice that $\mathcal{C}_{\mathcal{A}}$ is always a rational number with denominator between 1 and m. Breslauer, Colussi and Toniolo [7] and Hancart [30] proved that the number of symbol comparisons made by a certain class of on-line prefix-matching algorithms, which include all static algorithms, is at most $2n-1$. We prove next a similar claim about static algorithms using the notation we developed above.

Lemma 2.15 *The characteristic constant of any static algorithm $\mathcal{A}$ satisfies:*

$$1 \leq \mathcal{C}_{\mathcal{A}} \leq 2.$$

Proof. Clearly $1 \leq \mathcal{C}_{\mathcal{A}}$. Recall that $\Omega_{\mathcal{A}}(l) = \sum_{h=1}^{l} \Lambda^{-1}_{\mathcal{A},h}(\mathcal{P}[h])$ and that $\Lambda^{-1}_{\mathcal{A},h}(\sigma) \leq |\Sigma^{\mathcal{P}}_h|$, for any $\sigma \in \Sigma^{\mathcal{P}}_h$. Given some $\sigma \in \Sigma^{\mathcal{P}}_l \setminus \{\mathcal{P}[l]\}$, define $\hat{\mathcal{P}}[1..l] = \mathcal{P}[1..l-1]\ \sigma$. Then $|\Sigma^{\mathcal{P}}_l| \leq |\Sigma^{\hat{\mathcal{P}}}_l| + 1$ and $\hat{\mathcal{P}}[1..l]$ has period length $\pi = \pi^{l}_{\mathrm{first}}(\sigma)$. By Lemma 2.10, Inequality 1 becomes:

$$\begin{aligned}\frac{\Omega_{\mathcal{A}}(l-1) + \Lambda^{-1}_{\mathcal{A},l}(\sigma) - \Omega_{\mathcal{A}}(l-\pi)}{\pi} \quad &\leq \quad \frac{\sum_{h=l-\pi+1}^{l} |\Sigma^{\mathcal{P}}_h|}{\pi} \\ &\leq \quad \frac{1 + \sum_{h=l-\pi+1}^{l} |\Sigma^{\hat{\mathcal{P}}}_h|}{\pi} \quad \leq \quad 2.\end{aligned}$$

By Lemma 2.11, Inequality 2 becomes:

$$\frac{\Omega_{\mathcal{A}}(l-1) + |\Sigma^{\mathcal{P}}_l|}{l} \leq \frac{\sum_{h=1}^{l} |\Sigma^{\mathcal{P}}_h|}{l} \leq 2 - \frac{1}{l}.$$

And by Lemma 2.10, Inequality 3 becomes:

$$\frac{\Omega_{\mathcal{A}}(m) - \Omega_{\mathcal{A}}(m-\pi^{\mathcal{P}}_1)}{\pi^{\mathcal{P}}_1} \leq \frac{\sum_{h=m-\pi^{\mathcal{P}}_1+1}^{m} |\Sigma^{\mathcal{P}}_h|}{\pi^{\mathcal{P}}_1} \leq 2 - \frac{1}{\pi^{\mathcal{P}}_1}. \quad \square$$

In the next sections we address the problem of finding some static prefix-matching algorithm OPT that minimizes the characteristic constant $\mathcal{C}_{OPT}$. Such an algorithm clearly exists as there is only a finite number of static prefix-matching algorithms.

Example. Consider an instance of the prefix-matching problem with the pattern string $\mathcal{P}[1..m] = `a^{\alpha} b^{\beta}\text{'}$, for $\alpha \geq 2$, $\beta \geq 1$ and $m = \alpha + \beta$. If $\alpha = 0$ or $\beta = 0$, then the number of comparisons required is clearly n and if $\alpha = 1$ and $\beta \geq 1$, then the number of comparisons required is $\lfloor (2 - 1/m)n \rfloor$ as shown by Breslauer, Colussi and Toniolo [7] and Hancart [30].

Let us try to find an optimal static algorithm OPT that has the smallest possible characteristic constant $\mathcal{C}_{OPT}$. We define two algorithms. The first, which we call algorithm $\mathcal{AB}$ (it compares first 'a' and then if necessary 'b'), is defined as:

$$\Lambda_{\mathcal{AB},l}(h) = \begin{cases} `a\text{'} & h = 1 \text{ and } 1 \leq l \leq m \\ `b\text{'} & h = 2 \text{ and } \alpha < l \leq m, \end{cases}$$

and the second algorithm, which we call algorithm $\mathcal{X}$ (identical to algorithm $\mathcal{AB}$ up to position $\alpha+1$ and from there on it compares first 'b' and then if necessary 'a'), is defined as:

$$\Lambda_{\mathcal{X},l}(h) = \begin{cases} \text{'}a\text{'} & h=1 \text{ and } 1 \le l \le \alpha+1 \\ \text{'}b\text{'} & h=1 \text{ and } \alpha+1 < l \le m \\ \text{'}a\text{'} & h=2 \text{ and } \alpha+1 < l \le m \\ \text{'}b\text{'} & h=2 \text{ and } l = \alpha+1. \end{cases}$$

It is easy to verify that $\mathcal{C}_{\mathcal{AB}} = 1 + (m-\alpha)/m$ and that for any algorithm $\mathcal{A}$ other than $\mathcal{AB}$ and $\mathcal{X}$, $\mathcal{C}_{\mathcal{A}} > \mathcal{C}_{\mathcal{X}}$. If $\beta = 1$, then the two algorithms are identical and if $\beta \ge 2$, then $\mathcal{C}_{\mathcal{X}} = (\alpha+3)/(\alpha+1)$. Thus, the optimal static algorithm OPT can be chosen as the algorithm that has the smaller characteristic constant $\mathcal{C}_{\mathcal{AB}}$ or $\mathcal{C}_{\mathcal{X}}$, and $\mathcal{C}_{OPT} = \min\{\mathcal{C}_{\mathcal{AB}}, \mathcal{C}_{\mathcal{X}}\}$. Notice that there is a tie $\mathcal{C}_{\mathcal{AB}} = \mathcal{C}_{\mathcal{X}}$ only for the patterns '$a^\alpha b$', '$aabbbb$' and '$aaabbb$'.

In this example we have been able to reduce the number of candidates for an optimal static algorithm from the 2^β different algorithms to the two algorithms $\mathcal{AB}$ and $\mathcal{X}$. We show that in the general case it suffices to consider only few algorithms as candidates for an optimal algorithm. These algorithms are closely related to the generalized form of algorithm $\mathcal{AB}$ that we call algorithm *REVERSE*.

2.4 The algorithm *REVERSE*

In this section we define a static prefix-matching algorithm that has some special properties. This algorithm, which we call *REVERSE*, or *REV* for short, will be the basis for the optimal static algorithm that is developed in Section 2.5.

The order of comparisons $\Lambda_{REV,l}(h)$ in algorithm REV is defined such that,

$$\pi^l_{\text{last}}(\Lambda_{REV,l}(h)) > \pi^l_{\text{last}}(\Lambda_{REV,l}(g)) \qquad \text{for } l = 1,\ldots,m, \text{ and } 1 \le h < g \le |\Sigma^{\mathcal{P}}_l|.$$

More intuitively, if we recall that $\Sigma^{\mathcal{P}}_l = \{\mathcal{P}[l-\pi] \mid \pi \in \Pi^{\mathcal{P}[1..l-1]}\}$, then algorithm REV compares the symbols $\mathcal{P}[l-\pi]$ in *decreasing* order of the periods π, skipping the symbols that were already compared. Notice that algorithm KMP compares the symbols $\mathcal{P}[l-\pi]$ in *increasing* order of the periods π, exactly the opposite of algorithm REV. See Figure 1.

The main property of algorithm REV that is used later in developing the optimal static algorithm is given in the following lemma.

Lemma 2.16 *The cost function of algorithm REV is additive. That is,*

$$\Omega_{REV}(l-1) + \Lambda^{-1}_{REV,l}(\mathcal{P}[l-\pi]) = \Omega_{REV}(\pi) + \Omega_{REV}(l-\pi), \qquad \text{for } l = 1,\ldots,m, \text{ and } \pi \in \Pi^{\mathcal{P}[1..l-1]}.$$

In most places we use this lemma with $\pi = \pi^l_{\text{first}}(\sigma)$, *for some* $\sigma \in \Sigma^{\mathcal{P}}_l$

```
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
a c a b a c a a a c a b a c a c a c a b a c a a a c a b a c a D
                                a c a b a c a a a c a b a c a C
                                                a c a b a c a A
                                                        a c a B
                                                            a C
                                                              A
```

$$\Sigma^{\mathcal{P}}_{32} = \{\text{`}a\text{'}, \text{`}b\text{'}, \text{`}c\text{'}, \text{`}d\text{'}\}$$

Figure 1: Algorithm REV compares first 'a', then 'c', then 'b' and last 'd'. Algorithm KMP compares 'd', 'c', 'a' and 'b'. Note that these comparison orders are not opposite of each other.

Proof. By Proposition 2.6, Lemma 2.8 and the definition of algorithm REV,

$$\Lambda_{REV,h}(g) = \Lambda_{REV,h-\pi}(g) \quad \text{for } h = \pi+1, \ldots, l, \text{ and } g = 1, \ldots, |\Sigma^{\mathcal{P}}_{h-\pi}|.$$

If $\pi = 0$, then by definition $\Omega_{REV}(l) = \Omega_{REV}(l-1) + \Lambda^{-1}_{REV,l}(\mathcal{P}[l])$. If $\pi > 0$, then we prove by induction on h that $\Omega_{REV}(h) = \Omega_{REV}(\pi) + \Omega_{REV}(h-\pi)$, for $h = \pi, \ldots, l-1$. The basis of the induction for $h = \pi$ clearly holds. Observing that $\mathcal{P}[h] = \mathcal{P}[h-\pi]$, for $h = \pi+1, \ldots, l-1$,

$$\begin{aligned}
\Omega_{REV}(h) &= \Omega_{REV}(h-1) + \Lambda^{-1}_{REV,h}(\mathcal{P}[h]) \\
&= \Omega_{REV}(\pi) + \Omega_{REV}(h-\pi-1) + \Lambda^{-1}_{REV,h-\pi}(\mathcal{P}[h-\pi]) \\
&= \Omega_{REV}(\pi) + \Omega_{REV}(h-\pi).
\end{aligned}$$

Finally, $\Lambda^{-1}_{REV,l}(\mathcal{P}[l-\pi])$ is defined since $\mathcal{P}[l-\pi] \in \Sigma^{\mathcal{P}}_l$, and thus,

$$\begin{aligned}
&\Omega_{REV}(l-1) + \Lambda^{-1}_{REV,l}(\mathcal{P}[l-\pi]) = \\
&\Omega_{REV}(\pi) + \Omega_{REV}(l-\pi-1) + \Lambda^{-1}_{REV,l-\pi}(\mathcal{P}[l-\pi]) = \\
&\Omega_{REV}(\pi) + \Omega_{REV}(l-\pi).
\end{aligned}$$

One can also verify that REV is the only algorithm that has this property. □

Intuitively, the last lemma means that in the analysis of algorithm REV, comparisons made at any text position can be charged against the text symbol being compared and the charges do not have to be reassigned later.

The constraints on the characteristic constant in Lemma 2.14 are redundant for algorithm REV.

Lemma 2.17 *The characteristic constant $\mathcal{C}_{REV}$ is given as,*

$$\mathcal{C}_{REV} = \max_{l=1,\ldots,m} \left\{ \frac{\Omega_{REV}(l)}{l} \right\}.$$

It suffices to take the maximum only over those l, such that $\mathcal{P}[1..l]$ is unbordered.

Proof. The proof follows from Lemma 2.16. The constraints imposed by Inequality 1 become:

$$\frac{\Omega_{REV}(l-1)+\Lambda^{-1}_{REV,l}(\sigma)-\Omega_{REV}(l-\pi^{l}_{\text{first}}(\sigma))}{\pi^{l}_{\text{first}}(\sigma)}=\frac{\Omega_{REV}(\pi^{l}_{\text{first}}(\sigma))}{\pi^{l}_{\text{first}}(\sigma)}.$$

The constraints imposed by Inequality 2 satisfy:

$$\frac{\Omega_{REV}(l)}{l}=\frac{\Omega_{REV}(l-1)+\Lambda^{-1}_{REV,l}(\mathcal{P}[l])}{l}\leq\frac{\Omega_{REV}(l-1)+|\Sigma^{\mathcal{P}}_{l}|}{l}\leq\mathcal{C}_{REV}.$$

Taking $\sigma=\Lambda_{REV,l}(|\Sigma^{\mathcal{P}}_{l}|)$, these constraints become:

$$\begin{aligned}\frac{\Omega_{REV}(l-1)+|\Sigma^{\mathcal{P}}_{l}|}{l} &= \frac{\Omega_{REV}(l-1)+\Lambda^{-1}_{REV,l}(\sigma)}{l}\\ &= \frac{\Omega_{REV}(\pi^{l}_{\text{first}}(\sigma))+\Omega_{REV}(l-\pi^{l}_{\text{first}}(\sigma))}{l}.\end{aligned}$$

And if $\pi^{l}_{\text{first}}(\sigma)>0$, then,

$$\frac{\Omega_{REV}(\pi^{l}_{\text{first}}(\sigma))+\Omega_{REV}(l-\pi^{l}_{\text{first}}(\sigma))}{l}\leq \max\left\{\frac{\Omega_{REV}(\pi^{l}_{\text{first}}(\sigma))}{\pi^{l}_{\text{first}}(\sigma))},\frac{\Omega_{REV}(l-\pi^{l}_{\text{first}}(\sigma))}{l-\pi^{l}_{\text{first}}(\sigma)}\right\}.$$

Similarly to the last inequality, one sees that it suffices to take the maximum only over those l, such that $\mathcal{P}[1..l]$ is unbordered. Finally, the constraint imposed by Inequality 3 becomes:

$$\frac{\Omega_{REV}(m)-\Omega_{REV}(m-\pi^{\mathcal{P}}_{1})}{\pi^{\mathcal{P}}_{1}}=\frac{\Omega_{REV}(\pi^{\mathcal{P}}_{1})}{\pi^{\mathcal{P}}_{1}}.\quad\square$$

Breslauer, Colussi and Toniolo [7] and Hancart [30] described a family of algorithms that make at most $\lfloor(2-1/m)n\rfloor$ symbol comparisons. One can verify that algorithm REV belongs to that family. Using the tools we have developed above we prove this bound directly.

Corollary 2.18 *The characteristic constant $\mathcal{C}_{REV}\leq 2-1/m$.*

Proof. By Lemma 2.17 and Lemma 2.11,

$$\mathcal{C}_{REV}=\max_{l=1,\ldots,m}\left\{\frac{\Omega_{REV}(l)}{l}\right\}\leq \max_{l=1,\ldots,m}\left\{\frac{\sum_{h=1}^{l}|\Sigma^{\mathcal{P}}_{h}|}{l}\right\}\leq\max_{l=1,\ldots,m}\left\{2-\frac{1}{l}\right\}=2-\frac{1}{m}.\quad\square$$

The next corollary shows that on two letter alphabets the pattern 'ab^{m-1}' is the only pattern, up to permuting the symbols 'a' and 'b', that requires $\lfloor(2-1/m)n\rfloor$ symbol comparisons. Note that for the strings $\Delta_h[1..2^h]$ defined in Section 2.1, $\mathcal{C}_{REV(\Delta_h)} = 2 - 1/2^h$ similarly to 'ab^{m-1}'.

Corollary 2.19 *If the pattern alphabet contains at most two distinct symbols, then,*

$$\mathcal{C}_{REV} = 1 + \max_{l=1,\ldots,m} \left\{ \frac{|\{h \mid 1 < h \leq l \text{ and } \mathcal{P}[h] \neq \mathcal{P}[1]\}|}{l} \right\}.$$

Proof. By the definition of algorithm REV, if the pattern contains only two distinct symbols, then $\Omega_{REV}(l) = l + |\{h \mid 1 < h \leq l \text{ and } \mathcal{P}[h] \neq \mathcal{P}[1]\}|$ and the claim follows from Lemma 2.17. □

The following lemma describes the relation between the cost function of algorithm REV on the pattern prefixes and the maximal delay.

Lemma 2.20 *Let* $d_l = \max\left\{|\Sigma_h^{\mathcal{P}}| \mid h = 1, \ldots, l\right\}$. *Then,*

$$2^{d_l - 1} - 1 \leq \Omega_{REV}(l) - l.$$

Proof. The claim is proved by induction on l, similarly to the proof of Lemma 2.11. The basis of the induction for $l = 1$ clearly holds. By Proposition 2.6, $d_l \leq d_{l-1} + 1$. The inductive hypothesis holds if $d_l = d_{l-1}$. So it remains to prove the claim if $d_l = d_{l-1} + 1$. Let $\pi = \pi^l_{\text{first}}(\Lambda_{REV,l}(|\Sigma_l^{\mathcal{P}}| - 1))$. Then, by Proposition 2.6 and by Lemma 2.16,

$$\Omega_{REV}(l) = \Omega_{REV}(l-1) + |\Sigma_l^{\mathcal{P}}| = \Omega_{REV}(\pi) + \Omega_{REV}(l-\pi) + 1.$$

By the definition of algorithm REV and by Proposition 2.6, $d_\pi = d_{l-\pi} = d_{l-1}$. By the inductive hypothesis,

$$2^{d_l - 1} - 1 \leq \Omega_{REV}(\pi) - \pi + \Omega_{REV}(l-\pi) - (l-\pi) + 1 = \Omega_{REV}(l) - l. \quad \square$$

2.5 The optimal static algorithm

We define the variants of algorithm REV that we call $\mathcal{R}{:}\theta$, for $2 \le \theta \le m$, such that $\mathcal{P}[\theta] \ne \mathcal{P}[1]$, as:

$$\Lambda_{\mathcal{R}:\theta,l}(h) = \begin{cases} \Lambda_{REV,l}(h) & 1 \le l < \theta \text{ and } 1 \le h \le |\Sigma_l^{\mathcal{P}}| \\ \Lambda_{REV,\theta}(h) & l = \theta,\ 1 \le h \le |\Sigma_\theta^{\mathcal{P}}| \text{ and} \\ & \qquad h \notin \left\{\Lambda_{REV,\theta}^{-1}(P[\theta]), \Lambda_{REV,\theta}^{-1}(\chi_\theta)\right\} \\ \mathcal{P}[\theta] & l = \theta \text{ and } h = \Lambda_{REV,\theta}^{-1}(\chi_\theta) \\ \chi_\theta & l = \theta \text{ and } h = \Lambda_{REV,\theta}^{-1}(\mathcal{P}[\theta]) \\ \mathcal{P}[l] & \theta < l \le m \text{ and } h = 1 \\ \Lambda_{REV,l}(h-1) & \theta < l \le m \text{ and } 2 \le h \le \Lambda_{REV,l}^{-1}(\mathcal{P}[l]) \\ \Lambda_{REV,l}(h) & \theta < l \le m \text{ and } \Lambda_{REV,l}^{-1}(\mathcal{P}[l]) < h \le |\Sigma_l^{\mathcal{P}}|, \end{cases}$$

where the symbol $\chi_\theta \in \left\{\Lambda_{REV,\theta}(h) \mid h = 1, \ldots, \Lambda_{REV,\theta}^{-1}(\mathcal{P}[\theta]) - 1\right\}$ is defined for algorithm $\mathcal{R}{:}\theta$, such that,

$$\frac{\Omega_{REV}(\pi_{\text{first}}^{\theta}(\chi_\theta)) + \Lambda_{REV,\theta}^{-1}(\mathcal{P}[\theta]) - \Lambda_{REV,\theta}^{-1}(\chi_\theta)}{\pi_{\text{first}}^{\theta}(\chi_\theta)} \le \frac{\Omega_{REV}(\pi_{\text{first}}^{\theta}(\sigma)) + \Lambda_{REV,\theta}^{-1}(\mathcal{P}[\theta]) - \Lambda_{REV,\theta}^{-1}(\sigma)}{\pi_{\text{first}}^{\theta}(\sigma)}$$
$$\forall \sigma \in \left\{\Lambda_{REV,\theta}(h) \mid h = 1, \ldots, \Lambda_{REV,\theta}^{-1}(\mathcal{P}[\theta]) - 1\right\}.$$

The algorithms $\mathcal{R}{:}\theta$ are constructed from three parts. The first part is used in positions $l = 1, \ldots, \theta - 1$, and is exactly identical to algorithm REV. The second part is used in position $l = \theta$ and is identical to algorithm REV, except that the order in which $\mathcal{P}[\theta]$ and χ_θ are compared is exchanged. In the third part that is used in positions $l = \theta + 1, \ldots, m$, the relative comparison order of the symbols is the same as in algorithm REV, except that $\mathcal{P}[l]$ is compared first.

We prove next that the algorithms $\mathcal{R}{:}\theta$ defined above are similar enough to algorithm REV to satisfy claims which are closely related to those given in Lemma 2.16 and Lemma 2.17. Unfortunately, in the analysis of the algorithms $\mathcal{R}{:}\theta$, we run across some special cases that require that we slightly change the definition above.

Definition 2.21 *Let* $\hat{\theta} = \min\{\theta \mid \mathcal{P}[\theta] \ne \mathcal{P}[1]\}$. *A position* θ, *such that* $\mathcal{P}[\theta] \ne \mathcal{P}[1]$, *is said to be* $<\pi, l>$ malignant *if the following conditions hold:*

$\pi = \pi_1^{\mathcal{P}[1..l-1]}$*;*

$\pi < \theta \le l - \pi$*;*

$\mathcal{P}[h] = \mathcal{P}[1]$ *for all* $h = 1, \ldots, \hat{\theta} - 1$ *and* $h = \hat{\theta} + 1, \ldots, \pi$*;*

$\mathcal{P}[l] \notin \left\{\mathcal{P}[1], \mathcal{P}[\hat{\theta}]\right\}$; *and*

$\mathcal{P}[l-\pi] = \mathcal{P}[\hat{\theta}]$.

Notice that for a given pattern $\mathcal{P}[1..m]$, if θ_1 is $<\pi_1, l_1>$ malignant and θ_2 is $<\pi_2, l_2>$ malignant, then $\pi_1 = \pi_2$ and $l_1 = l_2$.

If θ is $<\pi, l>$ malignant, then we change the definition of algorithm $\mathcal{R}$:θ at position l, and define $\Lambda_{\mathcal{R}:\theta,l}(h) = \Lambda_{KMP,l}(h)$, for $h = 1, 2, 3$. Namely,

$$\Lambda_{\mathcal{R}:\theta,l}(h) = \begin{cases} \mathcal{P}[l] & h = 1 \\ \mathcal{P}[\hat{\theta}] & h = 2 \\ \mathcal{P}[1] & h = 3. \end{cases}$$

Lemma 2.22 *The cost function of algorithm* $\mathcal{R}$:θ *satisfies for* $l = 1, \ldots, m$, *and* $\pi \in \Pi^{\mathcal{P}[1..l-1]}$:

1. *If* $\pi < \theta$ *and* $l - \pi < \theta$, *then*

$$\Omega_{\mathcal{R}:\theta}(l-1) + \Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l-\pi]) \leq \Omega_{\mathcal{R}:\theta}(\pi) + \Omega_{\mathcal{R}:\theta}(l-\pi),$$

except if $l = \theta$ *and* $\mathcal{P}[l-\pi] = \chi_\theta$, *where*

$$\Omega_{\mathcal{R}:\theta}(\theta-1) + \Lambda^{-1}_{\mathcal{R}:\theta,\theta}(\chi_\theta) = \Omega_{\mathcal{R}:\theta}(\pi) + \Omega_{\mathcal{R}:\theta}(\theta-\pi) + \Lambda^{-1}_{REV,\theta}(\mathcal{P}[\theta]) - \Lambda^{-1}_{REV,\theta}(\chi_\theta).$$

2. *If* $\pi < \theta$ *and* $l - \pi \geq \theta$, *then*

$$\Omega_{\mathcal{R}:\theta}(l-1) + \Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l-\pi]) \leq \Omega_{\mathcal{R}:\theta}(\pi) + \Omega_{\mathcal{R}:\theta}(l-\pi).$$

3. *If* $\pi \geq \theta$ *and* $l - \pi < \theta$, *then*

$$\Omega_{\mathcal{R}:\theta}(l-1) + \Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l-\pi]) \leq \Omega_{REV}(\theta) + \pi - \theta + \Omega_{\mathcal{R}:\theta}(l-\pi),$$

except if θ *is* $<\pi_1^{\mathcal{P}[1..l-1]}, l>$ *malignant and* $\mathcal{P}[l-\pi] = \mathcal{P}[1]$, *where*

$$\Omega_{\mathcal{R}:\theta}(l-1) + \Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l-\pi]) = \Omega_{REV}(\theta) + \pi - \theta + \Omega_{\mathcal{R}:\theta}(l-\pi) + 1.$$

4. *If* $\pi \geq \theta$, *and* $l - \pi \geq \theta > \hat{\theta}$, *then*[4]

$$\Omega_{\mathcal{R}:\theta}(l-1) + \Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l-\pi]) \leq \Omega_{REV}(\theta) + (1 + \frac{1}{\hat{\theta}})(\pi - \theta) + \Omega_{\mathcal{R}:\theta}(l-\pi).$$

[4] We ommit the inequality for the case $\theta = \hat{\theta}$.

Proof. We prove the four inequalities separately. From the definition of algorithm $\mathcal{R}{:}\theta$,

$$\begin{aligned} \Omega_{\mathcal{R}:\theta}(l) &= \Omega_{REV}(l) && \text{for } l = 1, \ldots, \theta - 1, \\ \Omega_{\mathcal{R}:\theta}(\theta) &\leq \Omega_{REV}(\theta) - 1, \\ \Omega_{\mathcal{R}:\theta}(l) &= \Omega_{\mathcal{R}:\theta}(\theta) + l - \theta && \text{for } l = \theta + 1, \ldots, m. \end{aligned}$$

1. If $\pi = 0$, then the claim follows by the definition of the cost function.

   If $l = \theta$ and $\mathcal{P}[l - \pi] = \chi_\theta$, then by Lemma 2.16,

$$\begin{aligned} &\Omega_{\mathcal{R}:\theta}(\theta - 1) + \Lambda^{-1}_{\mathcal{R}:\theta,\theta}(\chi_\theta) &&= \\ &\Omega_{\mathcal{R}:\theta}(\theta - 1) + \Lambda^{-1}_{REV,\theta}(\mathcal{P}[\theta]) &&= \\ &\quad \Omega_{\mathcal{R}:\theta}(\pi) + \Omega_{\mathcal{R}:\theta}(\theta - \pi) &&+ \Lambda^{-1}_{REV,\theta}(\mathcal{P}[\theta]) - \Lambda^{-1}_{REV,\theta}(\chi_\theta). \end{aligned}$$

   Recalling that algorithm $\mathcal{R}{:}\theta$ is identical to algorithm REV in positions $l = 1, \ldots, \theta - 1$, and that at position $l = \theta$,

$$\Lambda^{-1}_{\mathcal{R}:\theta,\theta}(\sigma) \leq \Lambda^{-1}_{REV,\theta}(\sigma) \quad \text{for } \sigma \in \Sigma^{\mathcal{P}}_\theta \setminus \{\chi_\theta\},$$

   the claim follows for $l = 1, \ldots, \theta$, by Lemma 2.16. It remains to prove the claim for $\theta < l < \theta + \pi$. For the positions $h = \theta, \ldots, l - 1$, we prove by induction that:

$$\Omega_{\mathcal{R}:\theta}(h) \leq \Omega_{\mathcal{R}:\theta}(\pi) + \Omega_{\mathcal{R}:\theta}(h - \pi) - 1.$$

   The basis of the induction for $h = \theta$ holds since $\Omega_{\mathcal{R}:\theta}(\theta) \leq \Omega_{REV}(\theta) - 1$, and the inductive step holds since:

$$\Omega_{\mathcal{R}:\theta}(h) = \Omega_{\mathcal{R}:\theta}(h - 1) + 1 \leq \Omega_{\mathcal{R}:\theta}(\pi) + \Omega_{\mathcal{R}:\theta}(h - \pi) - 1.$$

   Finally, since $l - \pi < \theta$ and $\Lambda^{-1}_{\mathcal{R}:\theta,l}(P[l - \pi]) \leq \Lambda^{-1}_{REV,l-\pi}(P[l - \pi]) + 1$, by the inductive hypothesis we get that,

$$\begin{aligned} &\Omega_{\mathcal{R}:\theta}(l - 1) + \Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l - \pi]) \leq \\ &\Omega_{\mathcal{R}:\theta}(l - 1) + \Lambda^{-1}_{REV,l-\pi}(\mathcal{P}[l - \pi]) + 1 \leq \\ &\Omega_{\mathcal{R}:\theta}(\pi) + \Omega_{\mathcal{R}:\theta}(l - \pi - 1) + \Lambda^{-1}_{REV,l-\pi}(\mathcal{P}[l - \pi]) = \\ &\Omega_{\mathcal{R}:\theta}(\pi) + \Omega_{\mathcal{R}:\theta}(l - \pi). \end{aligned}$$

2. If $\pi = 0$, then the claim follows by the definition of the cost function.

   The claim we wish to prove is equivalent to showing that:

$$\pi + \Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l - \pi]) - 1 \leq \Omega_{\mathcal{R}:\theta}(\pi).$$

Let $d_g = \max\left\{|\Sigma_h^{\mathcal{P}}| \mid h = 1, \ldots, g\right\}$. Since $\pi \in \Pi^{\mathcal{P}[1..l-1]}$, by Proposition 2.6,

$$\Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l-\pi]) - 1 \leq d_l - 1 \leq d_\pi.$$

Therefore, if $d_\pi \geq 3$, then by Lemma 2.20,

$$\pi + \Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l-\pi]) - 1 \leq \pi + d_\pi \leq \pi + 2^{d_\pi - 1} - 1 \leq \Omega_{\mathcal{R}:\theta}(\pi).$$

It remains to prove the claim for $d_\pi = 2$. In this case, the inequality

$$\pi + \Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l-\pi]) - 1 \leq \Omega_{\mathcal{R}:\theta}(\pi),$$

might be violated if and only if $\Omega_{REV}(\pi) = \pi + 1$ and $\Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l-\pi]) = d_l = 3$, which is equivalent to saying that θ is $<\pi, l>$ malignant. But by the (modified) definition of algorithm $\mathcal{R}{:}\theta$, $\Lambda_{\mathcal{R}:\theta,l}(3) = \mathcal{P}[1]$, and thus $\mathcal{P}[l-\pi] = \mathcal{P}[1]$, in contradiction to θ being $<\pi, l>$ malignant.

3. The claim we wish to prove is equivalent to showing that:

$$\Omega_{\mathcal{R}:\theta}(\theta) + l - \pi - 1 + \Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l-\pi]) \leq \Omega_{REV}(\theta) + \Omega_{\mathcal{R}:\theta}(l-\pi)$$

If θ is not $<\pi_1^{\mathcal{P}[1..l-1]}, l>$ malignant or if $\mathcal{P}[l-\pi] \neq \mathcal{P}[1]$, then since $l > \theta$:

$$\Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l-\pi]) \leq \Lambda^{-1}_{REV,l-\pi}(\mathcal{P}[l-\pi]) + 1.$$

So it suffices to show that:

$$\Omega_{\mathcal{R}:\theta}(\theta) + l - \pi + \Lambda^{-1}_{REV,l-\pi}(\mathcal{P}[l-\pi]) \leq \Omega_{REV}(\theta) + \Omega_{\mathcal{R}:\theta}(l-\pi).$$

But, by the definition of algorithm $\mathcal{R}{:}\theta$, $\Omega_{\mathcal{R}:\theta}(\theta) \leq \Omega_{REV}(\theta) - 1$, and,

$$l - \pi + \Lambda^{-1}_{REV,l-\pi}(\mathcal{P}[l-\pi]) \leq \Omega_{\mathcal{R}:\theta}(l-\pi) + 1,$$

establishing the claim.
If θ is $<\pi_1^{\mathcal{P}[1..l-1]}, l>$ malignant and $\mathcal{P}[l-\pi] = \mathcal{P}[1]$, then

$$\Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l-\pi]) = \Lambda^{-1}_{REV,l-\pi}(\mathcal{P}[l-\pi]) + 2 = 3,$$

and the claim follows.

4. We first prove that:

$$\Omega_{\mathcal{R}:\theta}(l-1) + \Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l-\pi]) \leq$$
$$\Omega_{REV}(\theta) + \pi - \theta + \Omega_{\mathcal{R}:\theta}(l-\pi) + \lfloor \log_2 \frac{2\pi}{\theta + \hat{\theta}} \rfloor,$$

which is equivalent to showing that:

$$\Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l-\pi]) - 1 \leq \Omega_{REV}(\theta) - \theta + \lfloor \log_2 \frac{2\pi}{\theta+\hat{\theta}} \rfloor.$$

Let $d_g = \max\left\{|\Sigma_h^{\mathcal{P}}| \mid h = 1, \ldots, g\right\}$. Since $\pi \in \Pi^{\mathcal{P}[1..l-1]}$, by Proposition 2.6,

$$\Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l-\pi]) - 1 \leq d_l - 1 \leq d_\pi.$$

If $\Omega_{REV}(\theta) - \theta \geq 4$ or if $\Omega_{REV}(\theta) - \theta = 3$ and $d_\theta = 2$, then by Lemma 2.12 and by Lemma 2.20,

$$d_\pi \leq d_\theta + \lfloor \log_2 \frac{\pi}{\theta+1} \rfloor + 1 \leq \Omega_{REV}(\theta) - \theta + \lfloor \log_2 \frac{2\pi}{\theta+\hat{\theta}} \rfloor.$$

If $\Omega_{REV}(\theta) - \theta = d_\theta = 3$, then $d_\theta = d_{\theta-1} + 1$, and by Lemma 2.12,

$$d_\pi \leq d_\theta + \lfloor \log_2 \frac{\pi}{\theta} \rfloor \leq \Omega_{REV}(\theta) - \theta + \lfloor \log_2 \frac{2\pi}{\theta+\hat{\theta}} \rfloor.$$

Since $\theta > \hat{\theta}$, the only remaining possibility is that $\Omega_{REV}(\theta) - \theta = d_\theta = 2$. Then, by a close inspection of the possible pattern prefixes $\mathcal{P}[1..\theta+\hat{\theta}]$, $d_{\theta+\hat{\theta}-1} = 2$, and by Lemma 2.12,

$$d_\pi \leq \Omega_{REV}(\theta) - \theta + \lfloor \log_2 \frac{2\pi}{\theta+\hat{\theta}} \rfloor.$$

But,

$$\lfloor \log_2 \frac{2\pi}{\theta+\hat{\theta}} \rfloor \leq \frac{\pi-\theta}{\hat{\theta}},$$

establishing that

$$\Omega_{\mathcal{R}:\theta}(l-1) + \Lambda^{-1}_{\mathcal{R}:\theta,l}(\mathcal{P}[l-\pi]) \leq$$
$$\Omega_{REV}(\theta) + (1+\frac{1}{\hat{\theta}})(\pi-\theta) + \Omega_{\mathcal{R}:\theta}(l-\pi). \quad \square$$

The constraints in Lemma 2.14 are redundant also for the algorithms $\mathcal{R}$:θ.

Lemma 2.23 *The characteristic constants of the algorithms* $\mathcal{R}$:θ *are given as:*

$$\mathcal{C}_{\mathcal{R}:\theta} = \max\left\{\mathcal{C}_{REV}(\theta-1), \frac{\Omega_{REV}(\pi^\theta_{\text{first}}(\chi_\theta)) + \Lambda^{-1}_{REV,\theta}(\mathcal{P}[\theta]) - \Lambda^{-1}_{REV,\theta}(\chi_\theta)}{\pi^\theta_{\text{first}}(\chi_\theta)}\right\}.$$

Proof. The proof is almost identical to the proof of Lemma 2.17, using Lemma 2.22 instead of Lemma 2.16. If $\theta = \hat{\theta}$, then $\mathcal{C}_{\mathcal{R}:\theta} = 2$, by Inequality 1 and Lemma 2.15. The constraints arising from Inequalities 1 and 2, imply that:

$$\mathcal{C}_{\mathcal{R}:\theta} \geq \max\left\{\mathcal{C}_{REV}(\theta-1), \frac{\Omega_{REV}(\pi^{\theta}_{\text{first}}(\chi_\theta)) + \Lambda^{-1}_{REV,\theta}(\mathcal{P}[\theta]) - \Lambda^{-1}_{REV,\theta}(\chi_\theta)}{\pi^{\theta}_{\text{first}}(\chi_\theta)}\right\}.$$

The other constraints are dominated by those above, since for $l \geq \theta > \hat{\theta}$,

$$\frac{\Omega_{\mathcal{R}:\theta}(l)}{l} \leq \frac{\Omega_{REV}(\theta) + l - \theta}{l} \leq \frac{\Omega_{REV}(\theta)}{\theta},$$

and since,

$$\frac{\Omega_{REV}(\theta) + (1+\frac{1}{\hat{\theta}})(l-\theta)}{l} \leq \max\left\{\frac{\Omega_{REV}(\theta)}{\theta}, \frac{\Omega_{REV}(\hat{\theta})}{\hat{\theta}}\right\}.$$

But,

$$\frac{\Omega_{REV}(\theta)}{\theta} \leq$$
$$\max\left\{\frac{\Omega_{REV}(\pi^{\theta}_{\text{first}}(\chi_\theta)) + \Lambda^{-1}_{REV,\theta}(\mathcal{P}[\theta]) - \Lambda^{-1}_{REV,\theta}(\chi_\theta)}{\pi^{\theta}_{\text{first}}(\chi_\theta)}, \frac{\Omega_{REV}(\theta - \pi^{\theta}_{\text{first}}(\chi_\theta))}{\theta - \pi^{\theta}_{\text{first}}(\chi_\theta)}\right\}.$$

If θ is $<\pi_1^{\mathcal{P}[1..l-1]}, l>$ malignant and $\mathcal{P}[l-\pi] = \mathcal{P}[1]$, then the remaining constraint araising from case 3 in Lemma 2.22 is also dominated by the constraints above, since by Lemma 2.16,

$$\frac{\Omega_{REV}(\theta) + \pi - \theta + 1}{\pi} \leq$$
$$\max\left\{\frac{\Omega_{REV}(\pi^{\theta}_{\text{first}}(\chi_\theta)) + \Lambda^{-1}_{REV,\theta}(\mathcal{P}[\theta]) - \Lambda^{-1}_{REV,\theta}(\chi_\theta)}{\pi^{\theta}_{\text{first}}(\chi_\theta)}, 1 + \frac{1}{\pi - \pi^{\theta}_{\text{first}}(\chi_\theta)}\right\},$$

establishing the claim. □

Given any static prefix-matching algorithm $\mathcal{A}$, define

$$\rho(\mathcal{A}) = \min\left\{l \mid 1 \leq l \leq m \text{ and } \Lambda^{-1}_{\mathcal{A},l}(\mathcal{P}[l]) < \Lambda^{-1}_{REV,l}(P[l])\right\}.$$

If $\rho(\mathcal{A})$ is defined above, then $\mathcal{P}[\rho(\mathcal{A})] \neq \mathcal{P}[1]$, and algorithm $\mathcal{R}{:}\rho(\mathcal{A})$ is also defined.

Theorem 2.24 *Given any static prefix-matching algorithm $\mathcal{A}$, then either*

$$\mathcal{C}_{\mathcal{R}:\rho(\mathcal{A})} \leq \mathcal{C}_{\mathcal{A}} \quad \text{or} \quad \mathcal{C}_{REV} \leq \mathcal{C}_{\mathcal{A}}.$$

Proof. If $\rho(\mathcal{A})$ is not defined, then $\Omega_{REV}(l) \le \Omega_{\mathcal{A}}(l)$, for $l = 1, \ldots, m$, and by Inequality 2,

$$\mathcal{C}_{\mathcal{A}} \ge \max_{l=1,\ldots,m} \left\{ \frac{\Omega_{\mathcal{A}}(l)}{l} \right\} \ge \max_{l=1,\ldots,m} \left\{ \frac{\Omega_{REV}(l)}{l} \right\} = \mathcal{C}_{REV}.$$

Assume that $\rho(\mathcal{A})$ is defined. Then, $\Omega_{REV}(l) \le \Omega_{\mathcal{A}}(l)$, for $l = 1, \ldots, \rho(\mathcal{A}) - 1$, and by Inequality 2,

$$\mathcal{C}_{\mathcal{A}} \ge \max_{l=1,\ldots,\rho(\mathcal{A})-1} \left\{ \frac{\Omega_{REV}(l)}{l} \right\}.$$

Since $\Lambda^{-1}_{\mathcal{A},\rho(\mathcal{A})}(\mathcal{P}[\rho(\mathcal{A})]) < \Lambda^{-1}_{REV,\rho(\mathcal{A})}(\mathcal{P}[\rho(\mathcal{A})])$, there must be at least one symbol

$$\chi \in \left\{ \Lambda_{REV,\rho(\mathcal{A})}(h) \mid h = 1, \ldots, \Lambda^{-1}_{REV,\rho(\mathcal{A})}(\mathcal{P}[\rho(\mathcal{A})]) - 1 \right\},$$

such that $\Lambda^{-1}_{\mathcal{A},\rho(\mathcal{A})}(\chi) \ge \Lambda^{-1}_{REV,\rho(\mathcal{A})}(\mathcal{P}[\rho(\mathcal{A})])$. But then, by Inequality 1,

$$\begin{aligned}
\frac{\Omega_{REV}(\pi^{\rho(\mathcal{A})}_{\text{first}}(\chi_{\rho(\mathcal{A})})) + \Lambda^{-1}_{REV,\rho(\mathcal{A})}(\mathcal{P}[\rho(\mathcal{A})]) - \Lambda^{-1}_{REV,\rho(\mathcal{A})}(\chi_{\rho(\mathcal{A})})}{\pi^{\rho(\mathcal{A})}_{\text{first}}(\chi_{\rho(\mathcal{A})})} &\le \\
\frac{\Omega_{REV}(\pi^{\rho(\mathcal{A})}_{\text{first}}(\chi)) + \Lambda^{-1}_{REV,\rho(\mathcal{A})}(\mathcal{P}[\rho(\mathcal{A})]) - \Lambda^{-1}_{REV,\rho(\mathcal{A})}(\chi)}{\pi^{\rho(\mathcal{A})}_{\text{first}}(\chi)} &\le \\
\frac{\Omega_{\mathcal{A}}(\rho(\mathcal{A}) - 1) + \Lambda^{-1}_{\mathcal{A},\rho(\mathcal{A})}(\chi) - \Omega_{\mathcal{A}}(\rho(\mathcal{A}) - \pi^{\rho(\mathcal{A})}_{\text{first}}(\chi))}{\pi^{\rho(\mathcal{A})}_{\text{first}}(\chi)} &\le \quad \mathcal{C}_{\mathcal{A}},
\end{aligned}$$

and by Lemma 2.23, $\mathcal{C}_{\mathcal{A}} \ge \mathcal{C}_{\mathcal{R}:\rho(\mathcal{A})}$. $\square$

By the last theorem one can find an optimal static algorithm among the m or fewer algorithms $\mathcal{R}{:}\theta$, $2 \le \theta \le m$, and algorithm REV. Thus, we define the static algorithm $REVERSE\ OPTIMAL$, or $\mathcal{RO}$ for short, to be the algorithm among algorithms $\mathcal{R}{:}\theta$ and REV with the smallest characteristic constant. We define $\mathcal{C}^{\mathcal{P}[1..m]}_{on-line} = \mathcal{C}_{\mathcal{RO}(\mathcal{P}[1..m])}$.

Example. Consider the pattern $\mathcal{P}[1..10] =$ '*abaacabacb*'. Then, $\mathcal{C}_{\mathcal{R}:2} = 2$, $\mathcal{C}_{\mathcal{R}:5} = \frac{5}{3}$, $\mathcal{C}_{\mathcal{R}:7} = \frac{5}{3}$, $\mathcal{C}_{\mathcal{R}:9} = \frac{12}{7}$, $\mathcal{C}_{\mathcal{R}:10} = \frac{16}{9}$, and $\mathcal{C}_{REV} = \frac{17}{10}$. Thus, algorithm $\mathcal{RO}$ can be either algorithm $\mathcal{R}{:}5$ or algorithm $\mathcal{R}{:}7$.

Remark. If the pattern contains only two distinct symbols, then the analysis above is substantially simplified. The simplified proof for this special case, which captures the main ideas, was sketched in the conference extended abstract [8].

2.6 The lower bound

By the observations made before, static algorithms must make at least $\mathcal{C}^{\mathcal{P}[1..m]}_{on-line} \times (n-m)+m$ symbol comparisons. We prove that the same lower bound holds for any on-line algorithm.

Theorem 2.25 *Any on-line prefix-matching algorithm must make at least* $\mathcal{C}^{\mathcal{P}[1..m]}_{on-line} \times (n-m)+m$ *symbol comparisons.*

Proof. The proof is similar to the proof of Theorem 2.24. Recall that $\mathcal{C}^{\mathcal{P}[1..m]}_{on-line} = \mathcal{C}_{\mathcal{RO}}$. We prove by induction that the number of symbol comparisons made by any on-line algorithm, when reaching text position t, just before comparing the text symbol $\mathcal{T}[t]$, is at least

$$\mathcal{C}_{\mathcal{RO}} \times (k_0^t - 1) + \Omega_{REV}(t-k_0^t),$$

and that $\Omega_{REV}(l)/l < \mathcal{C}_{\mathcal{RO}}$, for $l = 1, \ldots, t-k_0^t$. The inductive base clearly holds for $t=1$. We assume that the inductive hypothesis holds at text position t and show how an adversary can force the inductive hypothesis to hold at text position $t+1$.

Let $l = t - k_0^t + 1$. Observe that by answering comparisons made by the on-line algorithm as unequal, an adversary will eventually force the algorithm to compare $\mathcal{T}[t]$ to all symbols in $\Sigma_l^{\mathcal{P}}$. If $(\Omega_{REV}(l-1) + |\Sigma_l^{\mathcal{P}}|)/l \geq C_{\mathcal{RO}}$, then the adversary forces the algorithm to compare $\mathcal{T}[t]$ to all symbols in $\Sigma_l^{\mathcal{P}}$, giving unequal answers, resulting in $k_0^{t+1} = t+1$, and maintaining the inductive hypothesis.

Otherwise, let $\Lambda^{-1}(\sigma)$ be the number of comparisons that the on-line algorithm makes at text position t until comparing $\sigma \in \Sigma_l^{\mathcal{P}}$ to $\mathcal{T}[t]$. Notice that $\Lambda^{-1}(\sigma)$ is not known to the adversary in advance. Then, if $\Lambda^{-1}(\mathcal{P}[l]) \geq \Lambda^{-1}_{REV,l}(\mathcal{P}[l])$, the adversary sets $\mathcal{T}[t] = \mathcal{P}[l]$ and the inductive hypothesis is maintained by the definitions. Observe that by Lemma 2.17, this situation might not continue until $l = m$, since $\mathcal{C}_{\mathcal{RO}} \leq \mathcal{C}_{REV}$.

If $\Lambda^{-1}(\mathcal{P}[l]) < \Lambda^{-1}_{REV,l}(\mathcal{P}[l])$, then the adversary answers that $\mathcal{T}[t] \neq \mathcal{P}[l]$ and the algorithm must continue comparing the symbols in $\Sigma_l^{\mathcal{P}}$. There must be at least one symbol $\chi \in \left\{\Lambda_{REV,l}(h) \mid h = 1, \ldots, \Lambda^{-1}_{REV,l}(\mathcal{P}[l]) - 1\right\}$, such that $\Lambda^{-1}(\chi) \geq \Lambda^{-1}_{REV,l}(\mathcal{P}[l])$. The adversary sets $\mathcal{T}[t] = \chi$. Then, by the definition of the algorithms $\mathcal{RO}$ and $\mathcal{R}{:}\theta$, $\mathcal{C}_{\mathcal{RO}} \leq \mathcal{C}_{\mathcal{R}:l}$, and by the inductive hypothesis and by Lemma 2.23, we get that $k_0^{t+1} = k_0^t + \pi^l_{\text{first}}(\chi)$, and

$$\frac{\Omega_{REV}(l-1) + \Lambda^{-1}(\chi) - \Omega_{REV}(l - \pi^l_{\text{first}}(\chi))}{\pi^l_{\text{first}}(\chi)} \geq \frac{\Omega_{REV}(\pi^l_{\text{first}}(\chi)) + \Lambda^{-1}_{REV,l}(\mathcal{P}[l]) - \Lambda^{-1}_{REV,l}(\chi)}{\pi^l_{\text{first}}(\chi)} \geq \mathcal{C}_{\mathcal{R}:l},$$

establishing the inductive claim.

Once the inductive claim is established, recalling that $\Omega_{REV}(l) \geq l$ and that $\mathcal{C}_{\mathcal{RO}} \geq 1$, we get that the number of comparisons made by the algorithm is at least

$$\mathcal{C}_{\mathcal{RO}} \times (k_0^t - 1) + \Omega_{REV}(t - k_0^t) \geq \mathcal{C}_{\mathcal{RO}} \times (n - m) + m. \quad \Box$$

Remark. The lower bound given above could be expressed in a tighter form. However, our technique will always leave some gap between the lower and upper bounds due to boundary situations. Observe that as the end of the text is reached, there is no need to consider the whole pattern, but only its prefix that can still occur within the text. Thus, it makes sense to use at text position t, the static algorithm $\mathcal{RO}(\mathcal{P}[1..\min(m, n - k_0^t + 1)])$, that depends on t. Notice that this algorithm is either some fixed algorithm $\mathcal{R}{:}\theta$, or algorithm REV, and therefore, it might not be static over the whole text. In addition, since $\mathcal{C}_{\mathcal{RO}(\mathcal{P}[1..l])}$ is non-decreasing in l, it also makes sense, in many cases, to break ties between the characteristic constants $\mathcal{C}_{\mathcal{R}:\theta}$ and $\mathcal{C}_{REV}$, in favour of using the algorithm $\mathcal{R}{:}\theta$ with smaller value of θ.

Nevertheless, in some cases, optimal static algorithms do not give the best on-line solution. For example, consider the pattern $\mathcal{P}[1..4] =$ '*abac*' and a text of length 4. The optimal static algorithm can be forced by an adversary to make 7 comparisons, while an algorithm that will compare the text symbol $\mathcal{T}[2]$, first to '*b*' and then to '*a*', a comparison order that can never be followed by an optimal static algorithm, will make at most 6 comparisons.

2.7 Infinite length patterns

The prefix-matching problem can be analogously defined for infinite length patterns $\mathcal{P}[1..\infty]$ by observing that the lengths of the pattern prefixes that occur in the text can not exceed the text length. We define,

$$C_{on-line}^{\mathcal{P}[1..\infty]} = \lim_{m\to\infty} \lim_{n\to\infty} \frac{c_{on-line}^{\mathcal{P}[1..m]}(n)}{n} = \lim_{m\to\infty} \mathcal{C}_{on-line}^{\mathcal{P}[1..m]}.$$

The constant $C_{on-line}^{\mathcal{P}[1..\infty]}$ is always well defined since $\mathcal{C}_{on-line}^{\mathcal{P}[1..m]}$ is non-decreasing in m and bounded above by Lemma 2.15. We prove next that the characteristic constants of infinite length patterns cover the whole real interval $[1..2]$.

Lemma 2.26 *Given any real constant α, $1 \leq \alpha \leq 2$, there exists an infinite length pattern $\mathcal{P}[1..\infty]$, such that, $C_{on-line}^{\mathcal{P}[1..\infty]} = \alpha$.*

Proof. The proof constructs for any given α, $1 \leq \alpha \leq 2$, an infinite length pattern $\mathcal{P}[1..\infty]$ over the alphabet $\{$'*a*', '*b*'$\}$, whose characteristic constant $C_{on-line}^{\mathcal{P}[1..\infty]}$

is equal to α. The pattern is defined inductively by setting $\mathcal{P}[1] = \text{`}a\text{'}$, and, for $l = 2, 3, \ldots,$

$$\mathcal{P}[l] = \begin{cases} \text{`}b\text{'} & \text{if } \frac{\Omega_{REV(\mathcal{P}[1..l-1])}(l-1)+2}{l} \leq \alpha \\ \text{`}a\text{'} & \text{otherwise.} \end{cases}$$

By Corollary 2.19, $\mathcal{C}_{REV(\mathcal{P}[1..l])} \leq \alpha$. We prove that $\mathcal{C}_{\mathcal{R}:l} \geq \alpha$, for all l, and that,

$$C_{on-line}^{\mathcal{P}[1..\infty]} = \lim_{l\to\infty} \mathcal{C}_{REV(\mathcal{P}[1..l])} = \alpha.$$

If $\mathcal{P}[1..\infty]$ contains finitely many symbols 'a', then $\lim_{l\to\infty} \mathcal{C}_{REV(\mathcal{P}[1..l])} = \alpha = 2$, by Corollary 2.19. If $\mathcal{P}[l] = \text{`}b\text{'}$, then algorithm $\mathcal{R}{:}l$ is defined. Let $\pi = \pi^l_{\text{first}}(\text{`}a\text{'})$. If $\pi + 1 < l$, then $\mathcal{P}[\pi + 1] = \text{`}a\text{'}$, and by Lemma 2.23,

$$\mathcal{C}_{\mathcal{R}:l} \geq \frac{\Omega_{REV}(\pi) + 1}{\pi} \geq \frac{\Omega_{REV}(\pi + 1) + 1}{\pi + 1} \geq \alpha.$$

If $\pi + 1 = l$, then let $h = \max\{g \mid 1 \leq g < l \text{ and } \mathcal{P}[g] = \text{`}a\text{'}\}$. By Lemma 2.23 and by the definition of $\mathcal{P}[h] = \text{`}a\text{'}$,

$$\mathcal{C}_{\mathcal{R}:l} \geq \frac{\Omega_{REV}(\pi) + 1}{\pi} \geq \frac{\Omega_{REV}(h) + 1}{h} \geq \alpha.$$

Assume by contradiction that there exists an $\epsilon > 0$, such that $\alpha - \mathcal{C}_{REV(\mathcal{P}[1..l])} \geq \epsilon$, for all l. Let h be a position of $\mathcal{P}[1..\infty]$, such that $h \geq 1/\epsilon$ and $\mathcal{P}[h] = \text{`}a\text{'}$. Then, by Corollary 2.19,

$$\frac{\Omega_{REV}(h-1) + 2}{h} = \frac{\Omega_{REV}(h)}{h} + \frac{1}{h} \leq \mathcal{C}_{REV(\mathcal{P}[1..h])} + \epsilon \leq \alpha,$$

in contradiction to the definition of $\mathcal{P}[h] = \text{`}a\text{'}$. □

3 Off-line prefix-matching

The analysis of off-line prefix-matching algorithms is complicated by the greater freedom they possess. Breslauer, Colussi and Toniolo [7] showed that for the pattern 'ab^{m-1}' off-line algorithms are not better than on-line algorithms. Nevertheless, off-line algorithms can be superior to on-line algorithms in certain cases as we show next.

Lemma 3.1 *There exists pattern strings for which off-line prefix-matching algorithms require fewer symbol comparisons than on-line algorithms.*

Proof. Consider the pattern $\mathcal{P}[1..7] = \text{`}aaaabbb\text{'}$. This pattern belongs to the family of patterns discussed in the example in Section 2.3. It is not difficult to verify that $\mathcal{C}_{on-line}^{`aaaabbb'} = \frac{7}{5}$ and thus, on-line prefix-matching algorithms require about $\frac{7}{5}n$ symbol comparisons. We show next that there exists an off-line prefix-matching algorithm that requires only about $\frac{4}{3}n$ symbol comparisons. (The lower bound proof of Breslauer, Colussi and Toniolo [7] can be applied almost unchanged to this pattern, showing that this bound is tight for off-line algorithms, up to a small additive constant.)

The off-line algorithm relies on the fact that the optimal static algorithm for the pattern $\mathcal{P}[1..7]$ is $\mathcal{R}$:6, while the optimal static algorithm for the pattern $\mathcal{P}[1..6]$ is REV. The off-line algorithm examines a single text position ahead to help it decide whether to proceed as algorithm $\mathcal{R}$:6 or as algorithm REV.

Observe that the comparison order functions $\Lambda_{\mathcal{R}:6,l}$ and $\Lambda_{REV,l}$ are identical for $l = 1, \ldots, 5$. The off-line algorithm proceeds in the same manner as these on-line algorithms as long as $t - k_0^t + 1 \leq 5$. When $t - k_0^t + 1 = 6$, the off-line algorithm compares the text symbol $\mathcal{T}[k_0^t + 9]$ to 'a'. If $\mathcal{T}[k_0^t + 9] =$ 'a', then the algorithm proceeds as algorithm $\mathcal{R}$:6, and otherwise as algorithm REV. In both cases, the off-line algorithm will avoid comparing $\mathcal{T}[k_0^t + 9]$ to 'a' again. It is straightforward to verify that this off-line algorithm makes at most $\frac{4}{3}n$ symbol comparisons. $\square$

4 The self-prefix problem

In this section we consider the special case of the prefix-matching problem where the pattern and the text are the same string and all symbol comparisons are accounted[5]. This problem, which we call the *self-prefix* problem, is the essence of the pattern preprocessing step of the prefix-matching algorithm. Throughout this section we assume that the input to the self-prefix problem is the pattern string $\mathcal{P}[1..m]$.

Proposition 4.1 *The periods lengths $\pi_1^{\mathcal{P}[1..l]}$, for $l = 1, \ldots, m$, can be computed from the output of the self-prefix problem $\Phi[1..m]$, and vice versa, in $O(m)$ time without making any extra symbol comparisons.*

Observe that the Knuth-Morris-Pratt string-matching algorithm computes in its pattern preprocessing step, the so called *failure function*, $f(l) = l - \pi_1^{\mathcal{P}[1..l]}$, which is essentially the same as the prefix period lengths $\pi_1^{\mathcal{P}[1..l]}$, for $l = 1, \ldots, m$.

Clearly, if the text is identical to the pattern, then there must be an occurrence of the pattern starting at the first position of the text. Since we do not wish to spend any labour to discover this trivial fact, we formally define the text $\mathcal{T}[1..m-1]$ to be $\mathcal{P}[2..m]$. Thus, in the self-prefix problem we always have

[5] In contrast to the unaccounted pattern preprocessing step in the discussion about on-line prefix-matching algorithms.

that:

$$k_h^t = \pi_{h+1}^{\mathcal{P}[1..t]} \quad \text{for} \quad t = 1, \ldots, m-1 \quad \text{and} \quad h = 0, \ldots, |\Pi^{\mathcal{P}[1..t]}| - 2.$$

On-line self-prefix algorithms, which are defined analogously to on-line prefix-matching algorithms, always compare the symbol $\mathcal{T}[t] = \mathcal{P}[t+1]$, $t = 1, \ldots, m-1$, to the symbols $\sigma \in \Sigma^{\mathcal{P}}_{t-\pi_1^{\mathcal{P}[1..t]}+1}$.

The self-prefix problem can be solved by any on-line static prefix-matching algorithm $\mathcal{A}$. Denote by $\omega_{\mathcal{A}}(l)$, the number of symbol comparisons made by the static prefix-matching algorithm $\mathcal{A}$, when used to solve the self-prefix problem on the input prefix $\mathcal{P}[1..l]$. Then,

$$\omega_{\mathcal{A}}(l) = \begin{cases} 0 & l = 1 \\ \omega_{\mathcal{A}}(l-1) + \Lambda_{\mathcal{A}, l-\pi_1^{\mathcal{P}[1..l-1]}}(\mathcal{P}[l]) & l = 2, \ldots, m \text{ and } \mathcal{P}[l] \in \Sigma^{\mathcal{P}}_{l-\pi_1^{\mathcal{P}[1..l-1]}} \\ \omega_{\mathcal{A}}(l-1) + |\Sigma^{\mathcal{P}}_{l-\pi_1^{\mathcal{P}[1..l-1]}}| & l = 2, \ldots, m \text{ and } \mathcal{P}[l] \notin \Sigma^{\mathcal{P}}_{l-\pi_1^{\mathcal{P}[1..l-1]}}. \end{cases}$$

Let $\mu(\mathcal{P}[1..l])$ be the number of non-empty *unbordered prefixes* of the string $\mathcal{P}[1..l]$. We first prove some upper bounds on the number of symbol comparisons made by the self-prefix algorithm REV.

Lemma 4.2 *The number of symbol comparisons made by the self-prefix algorithm REV satisfies:*

$$\omega_{REV}(m) = \Omega_{REV}(m) - \mu(\mathcal{P}[1..m]).$$

Proof. We prove by induction on $l = 1, \ldots, m$, that:

$$\omega_{REV}(l) = \Omega_{REV}(l) - \mu(\mathcal{P}[1..l]).$$

The base of the induction for $l = 1$ clearly holds. Assume that the inductive hypothesis holds for $l-1$. If $\mathcal{P}[l] \in \Sigma^{\mathcal{P}}_{l-\pi_1^{\mathcal{P}[1..l-1]}}$, then $P[1..l]$ is bordered and by the definition of algorithm REV, by Proposition 2.6 and by Lemma 2.8,

$$\begin{aligned} \omega_{REV}(l) = \Omega_{REV}(l-1) + \Lambda_{REV, l-\pi_1^{\mathcal{P}[1..l-1]}}(\mathcal{P}[l]) - \mu(\mathcal{P}[1..l-1]) \\ = \Omega_{REV}(l) - \mu(\mathcal{P}[1..l]). \end{aligned}$$

Otherwise, $P[1..l]$ is unbordered and,

$$\begin{aligned} \omega_{REV}(l) &= \Omega_{REV}(l-1) + |\Sigma^{\mathcal{P}}_{l-\pi_1^{\mathcal{P}[1..l-1]}}| - \mu(\mathcal{P}[1..l-1]) \\ &= \Omega_{REV}(l-1) + |\Sigma^{\mathcal{P}}_l| - \mu(\mathcal{P}[1..l]) \\ &= \Omega_{REV}(l) - \mu(\mathcal{P}[1..l]). \quad \Box \end{aligned}$$

The pattern preprocessing of the Knuth-Morris-Pratt [32] string-matching algorithm makes in some cases as many as $2m - 4$ symbol comparisons. We show next that the self-prefix algorithm REV is more efficient.

Corollary 4.3 *The number of symbol comparisons made by the self-prefix algorithm REV is at most* $2m - \lceil \log_2 m \rceil - 2$.

Proof. Let $\mathcal{P}[1..u_1], \mathcal{P}[1..u_2], \ldots,$ be all the unbordered prefixes of $\mathcal{P}[1..m]$, such that $1 = u_1 < \cdots < u_h$, and $u_h = m$ whether $\mathcal{P}[1..m]$ is bordered or not. Then, $\omega_{REV}(u_1) = 0$, and by Lemma 2.17 and Corollary 2.18, for $g = 2, \ldots, h$,

$$\begin{aligned} \omega_{REV}(u_g) &\leq \omega_{REV}(u_{g-1}) + \lfloor \mathcal{C}_{REV(\mathcal{P}[1..u_{g-1}])} \times (u_g - u_{g-1}) \rfloor \\ &\leq \omega_{REV}(u_{g-1}) + 2(u_g - u_{g-1}) - \lceil \frac{u_g}{u_{g-1}} - 1 \rceil. \end{aligned}$$

Expanding the last inequality,

$$\omega_{REV}(m) \leq 2m - 2 - \sum_{g=2}^{h} \lceil \frac{u_g}{u_{g-1}} - 1 \rceil \leq 2m - \lceil \log_2 m \rceil - 2. \quad \square$$

Notice that the last bound is tight for infinitely many strings, since the self-prefix algorithm REV will make $2m - \log_2 m - 2$ symbol comparisons when presented with the input strings $\mathcal{P}[1..m] = \Delta_h[1..2^h]$ that are defined in Section 2.1.

As long that the input string $\mathcal{P}[1..m]$ contains at most two distinct symbols, algorithm REV achieves better bounds. The next upper bound is tight with respect to the lower bound given by Breslauer, Colussi and Toniolo [7] for on-line self-prefix algorithms. Notice that we do not assume a priori that there are at most two distinct symbols in $\mathcal{P}[1..m]$ since in that case inequality to one alphabet symbol would imply equality to the other and $m-1$ symbol comparisons would suffice.

Corollary 4.4 *The number of symbol comparisons made by the self-prefix algorithm REV as long that the input string* $\mathcal{P}[1..m]$ *contains at most two distinct symbols is at most* $2m - \lceil 2\sqrt{m} \rceil$.

Proof. By Lemma 2.17 and Corollary 2.18,

$$\mathcal{C}_{REV(\mathcal{P}[1..m])} \leq 2 - \frac{1}{\mu(\mathcal{P}[1..m])}.$$

But, by Lemma 4.2,

$$\begin{aligned} \omega_{REV}(m) &\leq \lfloor \mathcal{C}_{REV(\mathcal{P}[1..m])} \times m \rfloor - \mu(\mathcal{P}[1..m]) \\ &\leq 2m - \lceil \frac{m}{\mu(\mathcal{P}[1..m])} \rceil - \mu(\mathcal{P}[1..m]) \\ &\leq 2m - \lceil 2\sqrt{m} \rceil. \quad \square \end{aligned}$$

We are now ready to prove the main upper and lower bounds for on-line self-prefix algorithms.

Theorem 4.5 *There exists an on-line self-prefix algorithm that makes at most $2m - \lceil\sqrt{2m}\rceil$ symbol comparisons. This is roughly tight for on-line self-prefix algorithms.*

Proof. We describe an on-line self-prefix algorithm that makes about $2m - \lceil\sqrt{2m}\rceil$ symbol comparisons and prove that this is roughly the best possible for on-line algorithms (up to an additive constant 2). Let $\omega(h)$ denote the number of symbol comparisons made by the on-line self-prefix algorithm on the pattern prefix $\mathcal{P}[1..h]$. The self-prefix algorithm starts making comparisons in the same order as the static self-prefix algorithm REV until it reaches the smallest position ζ, such that $\mathcal{C}_{REV(\mathcal{P}[1..\zeta])} > \mathcal{C}_{\mathcal{R}:l_\zeta}$, where algorithm $\mathcal{R}:l_\zeta$ is the algorithm with the smallest characteristic constant for $2 \leq l_\zeta \leq \zeta$. From position $\zeta + 1$ until the end of the string $\mathcal{P}[1..m]$ the algorithm follows the comparison order of the static self-prefix algorithm $\mathcal{R}:l_\zeta$. If no such position ζ is ever reached, then by Lemma 4.2,

$$\omega(m) = \omega_{REV}(m) \leq \lfloor \mathcal{C}_{REV} \times m \rfloor - \mu(\mathcal{P}[1..m]).$$

Otherwise, $\mathcal{P}[1..\zeta]$ must be unbordered by Lemma 2.17 and Lemma 2.23. By the minimality of ζ, $\mathcal{C}_{REV(\mathcal{P}[1..\zeta])} = \Omega_{REV}(\zeta)/\zeta$ and $\mathcal{C}_{REV(\mathcal{P}[1..\zeta])} \times \zeta = \lfloor \mathcal{C}_{\mathcal{R}:l_\zeta} \times \zeta \rfloor + 1$. Therefore, by Lemma 4.2,

$$\begin{aligned}
\omega(m) &\leq \omega_{REV}(\zeta) + \lfloor \mathcal{C}_{\mathcal{R}:l_\zeta} \times (m - \zeta) \rfloor \\
&= \Omega_{REV}(\zeta) + \lfloor \mathcal{C}_{\mathcal{R}:l_\zeta} \times (m - \zeta) \rfloor - \mu(\mathcal{P}[1..\zeta]) \\
&= \lfloor \mathcal{C}_{REV(\mathcal{P}[1..\zeta])} \times \zeta \rfloor + \lfloor \mathcal{C}_{\mathcal{R}:l_\zeta} \times (m - \zeta) \rfloor - \mu(\mathcal{P}[1..\zeta]) \\
&\leq \lfloor \mathcal{C}_{\mathcal{R}:l_\zeta} \times m \rfloor - \mu(\mathcal{P}[1..\zeta]) + 1.
\end{aligned}$$

The $\alpha\beta$ decomposition of $\mathcal{P}[1..m]$ is defined as follows: α_1 is the number of symbols at the beginning of $\mathcal{P}[1..m]$ that are equal to $\mathcal{P}[1]$; β_1 is the number of following symbols that are different from $\mathcal{P}[1]$; α_2 is the number of following symbols that is equal to $\mathcal{P}[1]$; β_2 is the number of following symbols that are different from $\mathcal{P}[1]$; and so on. Let $\delta_h = \sum_{g=1}^{h}(\alpha_g + \beta_g)$ and let b be the smallest integer such that $\delta_b = m$. E.g. the $\alpha\beta$ decomposition of $\mathcal{P}[1..9] =$ '*aabbcacca*' is $\alpha_1 = 2$, $\beta_1 = 3$, $\alpha_2 = 1$, $\beta_2 = 2$, $\alpha_3 = 1$ and $\beta_3 = 0$; $\delta_1 = 5$, $\delta_2 = 8$, $\delta_3 = 9$ and $b = 3$.

The proof distinguishes between the following cases.

1. If $\beta_2 = 0$, then the algorithm follows the comparison order of algorithm REV. The number of symbol comparison made is at most $m - 1$. From now on assume that $\beta_2 \geq 1$.

2. If $\alpha_1 \geq 2$, notice that if $\beta_1 \geq 2$, then algorithm $\mathcal{R}:\alpha_1{+}2$ is defined and $\mathcal{C}_{\mathcal{R}:\alpha_1+2} \leq \frac{5}{3}$; otherwise, if $\beta_1 = 1$, then $\mathcal{C}_{\mathcal{R}:\delta_1+\alpha_2+1} \leq \frac{5}{3}$. If the on-line

self-prefix algorithm does not reach the position ζ defined above, then $\mathcal{C}_{REV} \leq \frac{5}{3}$. Since $\mu(\mathcal{P}[1..m]) \geq 2$,

$$\omega(m) = \omega_{REV}(m) \leq$$
$$\lfloor \mathcal{C}_{REV} \times m \rfloor - \mu(\mathcal{P}[1..m]) \leq \lfloor \frac{5}{3} m \rfloor - 2 \leq 2m - \lceil \sqrt{2m} \rceil.$$

If the position ζ defined above is reached, then $\mathcal{C}_{\mathcal{R}:l_\zeta} \leq \frac{5}{3}$ and $\mu(\mathcal{P}[1..\zeta]) \geq 3$. But then,

$$\omega(m) \leq \lfloor \mathcal{C}_{\mathcal{R}:l_\zeta} \times m \rfloor - \mu(\mathcal{P}[1..m]) + 1 \leq \lfloor \frac{5}{3} m \rfloor - 2 \leq 2m - \lceil \sqrt{2m} \rceil.$$

3. The most complicated case is where $\alpha_1 = 1$. If $\alpha_2 = 1$ and $\mathcal{P}[\delta_1 + 2] = \mathcal{P}[2]$, or if $\alpha_2 \geq 2$, then $\mathcal{C}_{\mathcal{R}:\delta_1+\alpha_2+1} \leq 2 - 1/(\delta_1 + 1)$. If the algorithm does not reach the position ζ defined above, then $\mathcal{C}_{REV} \leq 2 - 1/(\delta_1 + 1)$ and $\mu(\mathcal{P}[1..m]) \geq \delta_1$. Thus,

$$\begin{aligned} \omega(m) = \omega_{REV}(m) &\leq \lfloor \mathcal{C}_{REV} \times m \rfloor - \mu(\mathcal{P}[1..m]) \\ &\leq 2m - \lceil \frac{m}{\delta_1 + 1} \rceil - \delta_1 \\ &\leq 2m - \lceil \sqrt{2m} \rceil. \end{aligned}$$

If the algorithm reaches the position ζ defined above, then $\mathcal{C}_{\mathcal{R}:l_\zeta} \leq 2 - 1/(\delta_1 + 1)$ and $\mu(\mathcal{P}[1..\zeta]) \geq \delta_1 + 1$. Hence,

$$\omega(m) \leq \lfloor \mathcal{C}_{\mathcal{R}:l_\zeta} \times m \rfloor - \mu(\mathcal{P}[1..\zeta]) + 1 \leq$$
$$2m - \lceil \frac{m}{\delta_1 + 1} \rceil - \delta_1 \leq 2m - \lceil \sqrt{2m} \rceil.$$

We have isolated the remaining case where $\alpha_2 = 1$ and $\mathcal{P}[\delta_1 + 2] \neq \mathcal{P}[2]$. Let d be the largest integer such that $\alpha_g = 1$ and $\mathcal{P}[\delta_{g-1} + 2] \neq \mathcal{P}[2]$, for all $g = 2, \ldots, d$. Then, $\mathcal{C}_{\mathcal{R}:l} = 2$, for all $1 \leq l \leq \delta_d$ such that algorithm $\mathcal{R}$:l is defined and $\mu(\mathcal{P}[1..\delta_d]) = \delta_d - d + 1 \geq \delta_d/2 + 1$.

We get similarly to the other cases above that $\mathcal{C}_{\mathcal{R}:\delta_d+\alpha_{d+1}+1} \leq 2 - 1/(\delta_d + 1)$. If the algorithm does not reach the position ζ defined above, then $\mathcal{C}_{REV} \leq 2 - 1/(\delta_d + 1)$ and $\mu(\mathcal{P}[1..m]) \geq \delta_d/2 + 1$. Therefore,

$$\begin{aligned} \omega(m) = \omega_{REV}(m) &\leq \lfloor \mathcal{C}_{REV} \times m \rfloor - \mu(\mathcal{P}[1..m]) \\ &\leq 2m - \lceil \frac{m}{\delta_d + 1} \rceil - \frac{\delta_d + 1}{2} \\ &\leq 2m - \lceil \sqrt{2m} \rceil. \end{aligned}$$

If the algorithm reaches the position ζ, then $\mathcal{C}_{\mathcal{R}:l_\zeta} \le 2 - 1/(\delta_d + 1)$ and $\mu(\mathcal{P}[1..m]) \ge \delta_d/2 + 2$. Hence,

$$\begin{aligned}
\omega(m) &\le \lfloor \mathcal{C}_{\mathcal{R}:l_\zeta} \times m \rfloor - \mu(\mathcal{P}[1..m]) + 1 \\
&\le 2m - \lceil \frac{m}{\delta_d + 1} \rceil - \frac{\delta_d + 1}{2} \\
&\le 2m - \lceil \sqrt{2m} \rceil .
\end{aligned}$$

To see that this is roughly the best possible for on-line algorithms consider the pattern $\mathcal{P}[1..m]$ that starts with the prefix $\mathcal{P}[1..\delta] =$ '$abac_1ac_2 \ldots ac_{h-1}ac_h$', such that $\delta = \lceil \sqrt{2m} \rceil$ is even and $c_i \notin \{\text{'}a\text{'}, \text{'}b\text{'}\}$. Then, any on-line self-prefix algorithm makes $\frac{3}{2}\delta - 1$ symbol comparisons on the pattern prefix $\mathcal{P}[1..\delta]$, and by carefully examining the proof of Theorem 2.25, at least $\mathcal{C}_{REV(\mathcal{P}[1..\delta])} \times (m - \delta) - 1 = (2 - 1/\delta)(m - \delta) - 1$ comparisons on the remaining suffix. This sums up to at least $2m - \lceil \sqrt{2m} + 3/2 \rceil$ symbol comparisons. □

Example. Consider the strings $\mathcal{P}[1..m] = \Delta_h[1..2^h]$, $h \ge 3$, that are defined in Section 2.1. Then, $\mathcal{C}_{\mathcal{R}:2} = 2$, $\mathcal{C}_{\mathcal{R}:4} = 2$, $\mathcal{C}_{\mathcal{R}:6} = \frac{9}{5}$ and $\mathcal{C}_{REV(\mathcal{P}[1..8])} = \frac{15}{8}$. Thus, the optimal static prefix-matching algorithm $\mathcal{RO}$ is algorithm $\mathcal{R}$:6 and $\mathcal{C}_{\mathcal{RO}} = \frac{9}{5}$. The self-prefix algorithm that is used in Theorem 4.5 is also the static algorithm $\mathcal{R}$:6, since algorithm REV that is used on the input prefix $\mathcal{P}[1..8]$ before the algorithm decides to use algorithm $\mathcal{R}$:6, is used only based on its definition on the input prefix $\mathcal{P}[1..4]$, which is identical to that of algorithm $\mathcal{R}$:6. On the other hand, consider any string $\mathcal{P}[1..m]$, whose prefix $\mathcal{P}[1..8] =$ '$ababacac$'. Then, $\mathcal{C}_{\mathcal{R}:2} = 2$, $\mathcal{C}_{\mathcal{R}:4} = \frac{5}{3}$, $\mathcal{C}_{\mathcal{R}:6} = \frac{7}{4}$, $\mathcal{C}_{\mathcal{R}:8} = \frac{11}{6}$ and $\mathcal{C}_{REV(\mathcal{P}[1..8])} = \frac{7}{4}$. Thus, the optimal static prefix-matching algorithm $\mathcal{RO}$ is algorithm $\mathcal{R}$:4 and $\mathcal{C}_{\mathcal{RO}} = \frac{5}{3}$. However, the self-prefix algorithm that is used in Theorem 4.5 is not static, since it behaves as algorithm REV on the input prefix $\mathcal{P}[1..8]$ before its decides to behave as algorithm $\mathcal{R}$:4.

Remark. Breslauer, Colussi and Toniolo [7] proved a $2m - \lceil 2\sqrt{m} \rceil$ lower bound for the self-prefix problem. They fail to mention, however, that their proof holds for on-line algorithms, but not for general off-line algorithms, for which the same proof gives only a $2m - \lceil 2\sqrt{2m} \rceil + 1$ lower bound. We have designed an off-line algorithm (not given here) that makes at most $2m - \lceil 2\sqrt{m} \rceil$ symbol comparisons. We suspect that this can be improved.

5 Implementation details

In this section we show that any given static prefix-matching algorithm $\mathcal{A}$, which is described by its comparison order, can be implemented in the standard random access machine computational model [1] in linear time and space. The same techniques are used to implement the on-line self-prefix algorithm from Theorem 4.5 and the optimal static prefix-matching algorithm $\mathcal{RO}$, including the pattern preprocessing that creates it, in linear time and space. The implementation details somewhat resemble those of the string-matching algorithm of Breslauer and Galil [9].

We assume that the description of the static prefix-matching algorithm $\mathcal{A}(\mathcal{P}[1..m])$ includes the delay $|\Sigma_l^{\mathcal{P}}|$ and the period lengths $\pi_1^{\mathcal{P}[1..l]}$, for $l = 1, \ldots, m$, and the comparison order functions of algorithm $\mathcal{A}$, that are given as:

$$\pi_{\text{first}}^l(\Lambda_{\mathcal{A},l}(h)) \quad \text{for } l = 1, \ldots, m \text{ and } h = 1, \ldots, |\Sigma_l^{\mathcal{P}}|.$$

Notice that the functions $\Lambda_{\mathcal{A},l}(h)$ need not be a part of the description since,

$$\Lambda_{\mathcal{A},l}(h) = \mathcal{P}[l - \pi_{\text{first}}^l(\Lambda_{\mathcal{A},l}(h))].$$

The delay $|\Sigma_l^{\mathcal{P}}|$ can also be computed from $\pi_1^{\mathcal{P}[1..l]}$, for $l = 1, \ldots, m$, since by Proposition 2.6,

$$|\Sigma_l^{\mathcal{P}}| = \begin{cases} |\Sigma^{\mathcal{P}}_{l-\pi_1^{\mathcal{P}[1..l-1]}}| & \text{if } \pi_1^{\mathcal{P}[1..l]} < l \\ |\Sigma^{\mathcal{P}}_{l-\pi_1^{\mathcal{P}[1..l-1]}}| + 1 & \text{otherwise.} \end{cases}$$

The reason that we choose to represent the comparison order functions by $\pi_{\text{first}}^l(\Lambda_{\mathcal{A},l}(h))$ is that the function $\pi_{\text{first}}^l(\sigma)$ is indexed by an alphabet symbol $\sigma \in \Sigma_l^{\mathcal{P}}$, which is difficult to implement efficiently without making additional symbol comparisons. The description of an algorithm requires $O(m)$ space, by Lemma 2.11.

Lemma 5.1 *Any given static prefix-matching algorithm $\mathcal{A}(\mathcal{P}[1..m])$ can be implemented in $O(n)$ time with constant auxiliary space.*

Proof. The generic implementation of the static algorithm $\mathcal{A}(\mathcal{P}[1..m])$ is given in Figure 2. The implementation clearly does not make any extraneous symbol comparisons and uses constant auxiliary space, apart from the output array $\Phi[1..n]$ that is used only to store the results of the computation.

Let $l = t - k_0^t + 1$. By Lemma 2.9 and Lemma 2.5, the members $k_i^t \in \mathcal{K}^t$ are grouped into $|\Sigma_l^{\mathcal{P}}|$ classes according to the symbols $\mathcal{P}[t - k_i^t + 1] \in \Sigma_l^{\mathcal{P}}$. If the algorithm reaches the conclusion that $\mathcal{T}[t] \neq \sigma$, for some $\sigma \in \Sigma_l^{\mathcal{P}}$, then it sets $\Phi[k_i^t] = t - k_i^t$, for all members $k_i^t \in \mathcal{K}^t$, such that,

$$k_i^t \in \left\{ k_0^t + \pi + \tau \mid \pi = \pi_{\text{first}}^l(\sigma) \text{ and } \tau \in \Pi^{\mathcal{P}[1..l-\pi]} \text{ and } \tau < l - \pi \right\}.$$

When the algorithm reaches the end of the text, there might still be members $k_i^{n+1} \in \mathcal{K}^{n+1}$, such that the pattern prefixes that start at these positions of the text continues until the end of the text. The algorithm sets $\Phi[k_i^{n+1}] = n - k_i^{n+1} + 1$, for all $k_i^{n+1} \in \mathcal{K}^{n+1}$.

The total time spent by the algorithm is $O(n)$, since the algorithm repeats the main loop n times and assigns values to each of the n entries $\Phi[h]$ of the output array only once. $\square$

– The implementation of the static prefix-matching algorithm $\mathcal{A}(\mathcal{P}[1..m])$.

$k = 1$ – k is k_0^t until it is updated to be k_0^{t+1}.

for $t = 1$ **to** n **do**

    $h = 1$

    $l = t - k + 1$

            – Find a symbol in $\Sigma_l^{\mathcal{P}}$ that matches $\mathcal{T}[t]$.

    while $h \leq |\Sigma_l^{\mathcal{P}}|$ **and** $\mathcal{T}[t] \neq \Lambda_{\mathcal{A},l}(h)$ **do**

        $h = h + 1$

    – Terminate the pattern prefixes that start at positions $k_i^t \in K^t$, such that

    – $\mathcal{P}[t - k_i^t + 1] \neq \mathcal{T}[t]$. (Lemma 2.5, Lemma 2.9 and Proposition 2.4.)

    for $g = 1$ **to** $|\Sigma_l^{\mathcal{P}}|$ **do**

        if $g \neq h$ **then** – $\mathcal{T}[t] \neq \Lambda_{\mathcal{A},l}(g)$.

            $\kappa = k + \pi_{\text{first}}^l(\Lambda_{\mathcal{A},l}(g))$

            while $\kappa \leq t$ **do**

                $\Phi[\kappa] = t - \kappa$

                $\kappa = \kappa + \pi_1^{\mathcal{P}[1..t-\kappa+1]}$

            end

        end

    if $h \leq |\Sigma_l^{\mathcal{P}}|$ **then** – A matching symbol was found.

        if $l = m$ **and** $\pi_{\text{first}}^l(\Lambda_{\mathcal{A},l}(h)) = 0$ **then**

            $\Phi[k] = m$ – A complete occurrence of $\mathcal{P}[1..m]$ was found.

            $k = k + \pi_1^{\mathcal{P}[1..m]}$

        else

            $k = k + \pi_{\text{first}}^l(\Lambda_{\mathcal{A},l}(h))$

    else – No match was found.

        $k = t + 1$

end

– Terminate all the remaining pattern prefixes. (Lemma 2.5 and Proposition 2.4.)

while $k \leq n$ **do**

    $\Phi[k] = n - k + 1$

    $k = k + \pi_1^{\mathcal{P}[1..n-k+1]}$.

end

Figure 2: The generic static prefix-matching algorithm.

Lemma 5.2 *The self-prefix algorithm and the pattern preprocessing step that constructs the optimal static prefix-matching algorithm $\mathcal{RO}(\mathcal{P}[1..m])$ can be implemented in $O(m)$ time and space.*

Proof. By Proposition 4.1, in order to solve the self-prefix problem it suffices to compute the period lengths $\pi_1^{\mathcal{P}[1..l]}$, for $l = 1, \ldots, m$. The pattern preprocessing implements the algorithm in Theorem 4.5, constructs algorithm REV and all the algorithms $\mathcal{R}{:}\theta$, and computes their characteristic constants $\mathcal{C}_{\mathcal{R}:\theta}$ and $\mathcal{C}_{REV}$. The algorithms $REV(\mathcal{P}[1..l])$ and $\mathcal{R}{:}\theta(\mathcal{P}[1..l])$ are constructed as soon as $\pi_1^{\mathcal{P}[1..l]}$ is given, so they can be used immediately, already at the next pattern position. The choice of which algorithm to use at the following pattern positions and which algorithm will be algorithm $\mathcal{RO}$ is made by comparing the characteristic constants of these algorithms.

Recall that the static algorithms $\mathcal{R}{:}\theta$ are constructed from three parts. The first part is identical to algorithm REV, while the second and third parts are small variations on algorithm REV. Let algorithm $\mathcal{M}$ be defined at positions l, such that $\mathcal{P}[l] \neq \mathcal{P}[1]$, as[6]:

$$\Lambda_{\mathcal{M},l}(h) = \begin{cases} \Lambda_{REV,l}(h) & 1 \leq h \leq |\Sigma_l^{\mathcal{P}}| \text{ and } h \notin \left\{\Lambda_{REV,l}^{-1}(P[l]), \Lambda_{REV,l}^{-1}(\chi_l)\right\} \\ \mathcal{P}[l] & h = \Lambda_{REV,l}^{-1}(\chi_l) \\ \chi_l & h = \Lambda_{REV,l}^{-1}(\mathcal{P}[l]), \end{cases}$$

where χ_l is defined in Section 2.5. Algorithm $\mathcal{E}$ is defined such that,

$$\Lambda_{\mathcal{E},l}(h) = \begin{cases} \mathcal{P}[l] & \theta < l \leq m \text{ and } h = 1 \\ \Lambda_{REV,l}(h-1) & \theta < l \leq m \text{ and } 2 \leq h \leq \Lambda_{REV,l}^{-1}(\mathcal{P}[l]) \\ \Lambda_{REV,l}(h) & \theta < l \leq m \text{ and } \Lambda_{REV,l}^{-1}(\mathcal{P}[l]) < h \leq |\Sigma_l^{\mathcal{P}}|. \end{cases}$$

Our construction builds the three algorithms REV, $\mathcal{M}$ and $\mathcal{E}$. Then, algorithm $\mathcal{R}{:}\theta$ is given as:

$$\Lambda_{\mathcal{R}:\theta,l}(h) = \begin{cases} \Lambda_{REV,l}(h) & l = 1, \ldots, \theta - 1 \\ \Lambda_{\mathcal{M},\theta}(h) & l = \theta \\ \Lambda_{\mathcal{E},l}(h) & l = \theta + 1, \ldots, m. \end{cases}$$

With the exception that if θ is $< \pi, l >$ malignant, then $\Lambda_{\mathcal{R}:\theta,l}(h) = \Lambda_{\mathcal{M},l}(h)$, for $h = 1, 2, 3$. (It so happens that algorithm $\mathcal{M}$ has the right comparison order in this case. Notice that it is possible to test if θ is malignant in constant time without making any extra symbol comparisons.)

This definition in three parts allows to construct the algorithms $\mathcal{R}{:}\theta$ in $O(m)$ time and their description to be stored in $O(m)$ space. The construction of these algorithms proceeds as the period lengths $\pi_1^{\mathcal{P}[1..l]}$ are being computed using the

[6]Algorithm $\mathcal{M}$ is not used at positions l, such that $\mathcal{P}[l] = \mathcal{P}[1]$, and it can be defined arbitrarily, or left undefined, at these positions.

previously constructed algorithms. Observe that $\pi_1^{\mathcal{P}[1..l]}$ is computed by a single iteration of the main loop in Figure 2.

At the beginning of this section we have shown how to compute $|\Sigma_l^{\mathcal{P}}|$, given the period lengths $\pi_1^{\mathcal{P}[1..l]}$. By the definition of algorithm REV and by Proposition 2.6,

$$\Lambda_{REV,l}(h) = \begin{cases} \Lambda_{REV,l-\pi_1^{\mathcal{P}[1..l-1]}}(h) & h = 1, \ldots, |\Sigma^{\mathcal{P}}_{l-\pi_1^{\mathcal{P}[1..l-1]}}| \\ \mathcal{P}[l] & \text{if } |\Sigma_l^{\mathcal{P}}| > |\Sigma^{\mathcal{P}}_{l-\pi_1^{\mathcal{P}[1..l-1]}}| \text{ and } h = |\Sigma_l^{\mathcal{P}}|. \end{cases}$$

The static algorithms $\mathcal{M}$ and $\mathcal{E}$ are constructed by appropriately modifying the comparison order functions of algorithm REV. The characteristic constants $\mathcal{C}_{REV(\mathcal{P}[1..l])}$ and $\mathcal{C}_{\mathcal{R}:l}$ can be computed efficiently by Lemma 2.17 and Lemma 2.23. The construction of the static prefix-matching algorithms described above and the computation of their characteristic constants takes $O(|\Sigma_l^{\mathcal{P}}|)$ time at position l, amounting to $O(m)$ time over the whole input string $\mathcal{P}[1..m]$, by Lemma 2.11. $\Box$

The following theorem is an immediate consequence of Lemma 5.1 and Lemma 5.2.

Theorem 5.3 *The optimal static prefix-matching algorithm $\mathcal{RO}(\mathcal{P}[1..m])$, including its preprocessing step, can be implemented in $O(n+m)$ time using $O(m)$ auxiliary space.*

6 Conclusions

We have been able to obtain roughly tight bounds on the number of comparisons required by on-line self-prefix algorithms and almost tight bounds for on-line prefix-matching algorithms with any given pattern. We hope that our techniques and results can prove useful in improving the bounds for the string-matching problem and in resolving some of the following problems.

1. What is the *exact* number of symbol comparisons required in on-line prefix-matching algorithms as a function of the pattern string and the text length? In off-line algorithms? If the pattern preprocessing is also accounted?

2. How many comparisons are required in general off-line self-prefix matching algorithms?

3. For which pattern strings off-line algorithms require fewer symbol comparisons than on-line?

4. Do comparisons of pairs of text symbols help in off-line prefix-matching algorithms? Paterson, Tassa and Zwick [36] have recently shown that comparisons of pairs of text symbols can help in the string-matching problem.

7 Acknowledgments

We thank Raffaele Giancarlo for his comments on an early version of this paper.

References

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

[2] A. Amir, G. Benson, and M. Farach. An alphabet-independent approach to two-dimensional pattern-matching. *SIAM J. Comput.*, 23(2):313–323, 1994.

[3] A. Apostolico and R. Giancarlo. The Boyer-Moore-Galil string searching strategies revisited. *SIAM J. Comput.*, 15(1):98–105, 1986.

[4] R.S. Boyer and J.S. Moore. A fast string searching algorithm. *Comm. of the ACM*, 20:762–772, 1977.

[5] D. Breslauer. Fast Parallel String Prefix-Matching. *Theoret. Comput. Sci.*, 137(2):269–278, 1995.

[6] D. Breslauer. Saving Comparisons in the Crochemore-Perrin String Matching Algorithm. *Theoret. Comput. Sci.*, to appear.

[7] D. Breslauer, L. Colussi, and L. Toniolo. Tight Comparison Bounds for the String Prefix-Matching Problem. *Inform. Process. Lett.*, 47(1):51–57, 1993.

[8] D. Breslauer, L. Colussi, and L. Toniolo. On the Exact Complexity of the String Prefix-Matching Problem. In *Proc. 2nd European Symposium on Algorithms*, number 855 in Lecture Notes in Computer Science, pages 483–494. Springer-Verlag, Berlin, Germany, 1994.

[9] D. Breslauer and Z. Galil. Efficient Comparison Based String Matching. *J. Complexity*, 9(3):339–365, 1993.

[10] R. Cole. Tight Bounds on the Complexity of the Boyer-Moore String Matching Algorithm. *SIAM J. Comput.*, 23(5):1075–1091, 1994.

[11] R. Cole, M. Crochemore, Z. Galil, L. Gąsieniec, R. Hariharan, S. Muthukrishnan, K. Park, and W. Rytter. Optimally fast parallel algorithms for preprocessing and pattern matching in one and two dimensions. In *Proc. 34th IEEE Symp. on Foundations of Computer Science*, pages 248–258, 1993.

[12] R. Cole and R. Hariharan. Tighter Bounds on the Exact Complexity of String Matching. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, pages 600–609, 1992.

[13] R. Cole, R. Hariharan, M. Paterson, and U. Zwick. Tighter Lower Bounds on the Exact Complexity of String Matching. *SIAM J. Comput.*, 24(1):30–45, 1995.

[14] L. Colussi. Correctness and efficiency of string matching algorithms. *Inform. and Control*, 95:225–251, 1991.

[15] L. Colussi. Fastest Pattern Matching in Strings. *J. Algorithms*, 16(2):163–189, 1994.

[16] M. Crochemore. Off-line serial exact string searching. Technical Report 92.1, Institut Gaspard Monge, Université de Marne la Vallée, Marne la Vallée, France, 1992.

[17] M. Crochemore. String-matching on ordered alphabets. *Theoret. Comput. Sci.*, 92:33–47, 1992.

[18] M. Crochemore, L. Gąsieniec, and W. Rytter. Turbo-BM. Technical Report 92.61, Laboratoire Informatique Théorique et Programmation, Université Paris 7, Paris, France, July 1992.

[19] M. Crochemore, T. Lecroq, A. Czumaj, L. Gąsieniec, S. Jarominek, W. Plandowski, and W. Rytter. Speeding Up Two String-Matching Algorithms. In *Proc. 9th Symp. on Theoretical Aspects of Computer Science*, number 577 in Lecture Notes in Computer Science, pages 589–600. Springer-Verlag, Berlin, Germany, 1992.

[20] M. Crochemore and D. Perrin. Two-way string-matching. *J. Assoc. Comput. Mach.*, 38(3):651–675, 1991.

[21] M. Crochemore and W. Rytter. Periodic Prefixes in Texts. In R. Capocelli, A. De Santis, and U. Vaccaro, editors, *Proc. of the Sequences '91 Workshop: "Sequences II: Methods in Communication, Security and Computer Science"*, pages 153–165. Springer-Verlag, Berlin, Germany, 1993.

[22] Z. Galil. Optimal parallel algorithms for string matching. *Inform. and Control*, 67:144–157, 1985.

[23] Z. Galil and R. Giancarlo. On the exact complexity of string matching: lower bounds. *SIAM J. Comput.*, 20(6):1008–1020, 1991.

[24] Z. Galil and R. Giancarlo. The exact complexity of string matching: upper bounds. *SIAM J. Comput.*, 21(3):407–437, 1992.

[25] Z. Galil and K. Park. Truly Alphabet-Independent Two-Dimensional Pattern Matching. In *Proc. 33th IEEE Symp. on Foundations of Computer Science*, pages 247–256, 1992.

[26] Z. Galil and J. Seiferas. Time-space-optimal string matching. *J. Comput. System Sci.*, 26:280–294, 1983.

[27] L. Gąsieniec and K. Park. Work-Time Optimal Parallel Prefix Matching. In *Proc. 2nd European Symposium on Algorithms*, number 855 in Lecture Notes in Computer Science, pages 471–482. Springer-Verlag, Berlin, Germany, 1994.

[28] L. Gąsieniec, W. Plandowski, and W. Rytter. Constant-space string matching with smaller number of comparisons: sequential sampling. In *Proc. 6th Symp. on Combinatorial Pattern Matching*, number 937 in Lecture Notes in Computer Science, pages 78–89. Springer-Verlag, Berlin, Germany, 1995.

[29] L.J. Guibas and A.M. Odlyzko. A New Proof of the Linearity of the Boyer-Moore String Matching Algorithm. *SIAM J. Comput.*, 9(4):672–682, 1980.

[30] C. Hancart. On Simon's string searching algorithm. *Inform. Process. Lett.*, 47(2):95–99, 1993.

[31] R. Hariharan and S. Muthukrishnan. Optimal Parallel Algorithms for Prefix Matching. In *Proc. 21th International Colloquium on Automata, Languages, and Programming*, number 820 in Lecture Notes in Computer Science, pages 203–214. Springer-Verlag, Berlin, Germany, 1994.

[32] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6:322–350, 1977.

[33] T. Lecroq. A variation on the Boyer-Moore algorithm. *Theoret. Comput. Sci.*, 92:119–144, 1992.

[34] M. Lothaire. *Combinatorics on Words*. Addison-Wesley, Reading, MA., U.S.A., 1983.

[35] G.M. Main and R.J. Lorentz. An $O(n \log n)$ algorithm for finding all repetitions in a string. *J. Algorithms*, 5:422–432, 1984.

[36] M. Paterson, S. Tassa, and U. Zwick. Looking for Mum and Dad. In preparation, 1994.

[37] R.V. Rivest. On the Worst Case Behavior of String-Searching Algorithms. *SIAM J. Comput.*, 6:669–674, 1977.

[38] U. Zwick and M.S. Paterson. Lower bounds for string matching in the sequential comparison model. Manuscript, 1991.

Recent Publications in the BRICS Report Series

RS-95-46 **Dany Breslauer, Livio Colussi, and Laura Toniolo.** ***On the Comparison Complexity of the String Prefix-Matching Problem*****. August 1995. 39 pp. Appears in Leeuwen, editor,** ***Algorithms - ESA '94: Second Annual European Symposium proceedings*****, LNCS 855, 1994, pages 483–494.**

RS-95-45 **Gudmund Skovbjerg Frandsen and Sven Skyum.** ***Dynamic Maintenance of Majority Information in Constant Time per Update*****. August 1995. 9 pp.**

RS-95-44 **Bruno Courcelle and Igor Walukiewicz.** ***Monadic Second-Order Logic, Graphs and Unfoldings of Transition Systems*****. August 1995. 39 pp. To be presented at CSL '95.**

RS-95-43 **Noam Nisan and Avi Wigderson.** ***Lower Bounds on Arithmetic Circuits via Partial Derivatives (Preliminary Version)*****. August 1995. 17 pp. To appear in** ***36th Annual Conference on Foundations of Computer Science*****, FOCS '95, IEEE, 1995.**

RS-95-42 **Mayer Goldberg.** ***An Adequate Left-Associated Binary Numeral System in the λ-Calculus*****. August 1995. 16 pp.**

RS-95-41 **Olivier Danvy, Karoline Malmkjær, and Jens Palsberg.** ***Eta-Expansion Does The Trick*****. August 1995. 23 pp.**

RS-95-40 **Anna Ingólfsdóttir and Andrea Schalk.** ***A Fully Abstract Denotational Model for Observational Congruence*****. August 1995. 29 pp.**

RS-95-39 **Allan Cheng.** ***Petri Nets, Traces, and Local Model Checking*****. July 1995. 32 pp. Full version of paper appearing in Proceedings of AMAST '95, LNCS 936, 1995.**

RS-95-38 **Mayer Goldberg.** ***Gödelisation in the λ-Calculus*****. July 1995. 7 pp.**

RS-95-37 **Sten Agerholm and Mike Gordon.** ***Experiments with ZF Set Theory in HOL and Isabelle*****. July 1995. 14 pp. To appear in** ***Proceedings of the 8th International Workshop on Higher Order Logic Theorem Proving and its Applications*****, LNCS, 1995.**