Saving comparisons in the Crochemore-Perrin string matching algorithm

D. Breslauer

Computer Science/Department of Algorithmics and Architecture

Report CS-R9322 March 1993

CWI is the National Research Institute for Mathematics and Computer Science. CWI is part of the Stichting Mathematisch Centrum (SMC), the Dutch foundation for promotion of mathematics and computer science and their applications. SMC is sponsored by the Netherlands Organization for Scientific Research (NWO). CWI is a member of ERCIM, the European Research Consortium for Informatics and Mathematics.

P.O. Box 4079, 1009 AB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

Saving Comparisons in the Crochemore-Perrin String Matching Algorithm

Dany Breslauer
CWI
P.O. Box 4079, 1009 AB
Amsterdam, The Netherlands

Abstract

Crochemore and Perrin discovered an elegant linear-time constant-space string matching algorithm that makes at most $2n - m$ symbol comparison. This paper shows how to modify their algorithm to use fewer comparisons.

Given any fixed $\epsilon > 0$, the new algorithm takes linear time, uses constant space and makes at most $n + \lfloor \frac{1+\epsilon}{2}(n-m) \rfloor$ symbol comparisons. If $O(\log m)$ space is available, then the algorithm makes at most $n + \lfloor \frac{1}{2}(n-m) \rfloor$ symbol comparisons. The pattern preprocessing step also takes linear time and uses constant space.

These are the first string matching algorithms that make fewer than $2n - m$ symbol comparisons and use sub-linear space.

1991 Mathematics Subject Classification: 68Q20, 68Q25, 68R15
1991 CR Categories: F.2.2
Keywords and Phrases: Pattern Matching, String Matching, Comparison Model, Time-Space Trade-off, Exact Complexity, Failure Function.
Note: The author was partially supported by the IBM Graduate Fellowship while studying at Columbia University and by the European Research Consortium for Informatics and Mathematics postdoctoral fellowship. Part of this work was done while the author was visiting at Università de L'Aquila, L'Aquila, Italy in summer 1991.

1 Introduction

String matching is the problem of finding all occurrences of a short string $\mathcal{P}[1..m]$ that is called *a pattern* in a longer string $\mathcal{T}[1..n]$ that is called *a text*. In this paper we study the exact comparison complexity of the string matching problem. We assume that the only access the algorithms have to the input strings is by pairwise symbol comparisons that result in equal or unequal answers.

Several algorithms solve the string matching problem in linear time. For a survey on string matching algorithm see Aho's paper [1]. Most known perhaps is the algorithm of Knuth, Morris and Pratt [22] that makes $2n-m$ comparisons in the worst case. A variant of the Boyer-Moore [4] algorithm that was designed by Apostolico and Giancarlo [2] also makes $2n - m$ comparisons. The original Boyer-Moore algorithm makes about $3n$ comparisons as shown recently by Cole [7]. All these algorithms work in two steps: in the first step the pattern is preprocessed and some information is stored and used later in a text processing step. Our bounds do not account for comparisons that are made in the pattern preprocessing step that can compare even all pairs of pattern symbols.

Research on the exact number of comparisons required to solve the string matching problem has been stimulated by Colussi's [10] discovery of an algorithm that makes at most $n + \frac{1}{2}(n-m)$ comparisons. This bound was improved by Galil and Giancarlo [15], Breslauer and Galil [5] and most recently

Report CS-R9322
ISSN 0169-118X
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

by Cole and Hariharan [8] who show that the string matching problem can be solved using at most $n + \frac{8}{3m}(n-m)$ comparisons[1]. Lower bounds given by Galil and Giancarlo [14], Zwick and Paterson [26], Cole and Hariharan [8] and Cole et al. [9] still leave a small gap between the lower and upper bounds.

The computation model considered in this paper consists of random-access read-only input registers, random-access write-only output registers and a limited number of auxiliary random-access read-write data registers. The number of bits per data register is bounded by some constant times the logarithm of $n+m$. The term *space* in this model refers to the number of auxiliary data registers used. Namely, a constant-space algorithm can use only a constant number of auxiliary registers.

The algorithms mentioned above use $O(m)$ auxiliary memory registers. However, the naive approach to string matching can find all occurrences of the pattern in the text in $O(nm)$ time using only constant auxiliary space. Galil and Seiferas [18] were the first to discover a linear-time constant-space string matching algorithm, disproving conjectures about a time-space tradeoff [3, 17]. They also showed that their algorithm can be implemented even on a six-head two-way finite automaton in linear time and conjectured that a multi-head one-way finite automaton can not solve the string matching problem [19, 23, 24]. This conjectured was very recently settled by Jiang and Li [21].

Crochemore and Perrin [12] discovered a simple linear-time constant-space string matching algorithm that makes at most $2n-m$ comparisons. Crochemore and Rytter [13] show how to reduce the number of comparisons made by the Galil-Seiferas [18] algorithm by a better choice of parameters. Crochemore [11] gives another constant-space string matching algorithm. The comparison bounds achieved by Galil and Seiferas [18], Crochemore and Rytter [13] and by Crochemore [11] are larger than $2n-m$.

This paper focuses on the number of comparisons required by constant-space string matching algorithms. It is shown that for any fixed $\epsilon > 0$, there exists a linear-time constant-space string matching algorithm that makes at most $n + \lfloor \frac{1+\epsilon}{2}(n-m) \rfloor$ comparisons. Our results are developed in three steps:

1. The Crochemore-Perrin string matching algorithm is modified to use the periodicity structure of the pattern in order to record some pattern suffixes that occur in the text. The modified algorithm takes linear time and uses $O(m)$ auxiliary space. It makes at most $n + \lfloor \frac{\min(\pi_1, m-\pi_1)}{m}(n-m) \rfloor \leq n + \lfloor \frac{1}{2}(n-m) \rfloor$ comparisons, where π_1 denotes the period length of the pattern.

   The same bound was obtained by Galil and Giancarlo [15] for Colussi's [10] algorithm. Our algorithm and its analysis are much simpler.

2. The periodicity structure of the pattern that is used in the modified algorithm can be stored in $\lceil \log_{\frac{3}{2}} m \rceil$ memory registers. Thus, the algorithm can be implemented using $O(\log m)$ auxiliary memory registers.

3. If only $c \geq 1$ registers are available to store the periodicity structure of the pattern, then we present an algorithm, which is a hybrid between the original Crochemore-Perrin algorithm and the modified algorithm, that makes at most $n + \lfloor \frac{1}{2-(\frac{2}{3})^{c-1}}(n-m) \rfloor$ comparisons.

   This establishes that there exists a linear-time constant-space string matching algorithm that makes fewer than $2n-m$ comparisons.

The pattern preprocessing step of the new algorithms can be implemented in linear time using a constant number of auxiliary memory registers except the registers that store a portion of the periodicity structure of the pattern which is used in the text processing step.

We proceed with the definitions of periods and their basic properties in Section 2. Section 3 overviews the original Crochemore-Perrin algorithm and Section 4 presents the modified algorithm. Section 5 gives more properties of periods which are used in Section 6 to save space. The pattern preprocessing step is discussed in Section 7. We conclude with a list of open problems in Section 8.

[1] All the string matching algorithm that are mentioned take linear time. The pattern preprocessing steps which are not accounted in the bounds take $O(m^2)$ time in Cole and Hariharan's algorithm and linear time in the other algorithms.

2 Properties of Strings

This sections gives some basic definitions and properties of strings.

Definition 2.1 *A string $\mathcal{S}[1..k]$ has a* period *of length π if $\mathcal{S}[i] = \mathcal{S}[i+\pi]$, for $i = 1, \cdots, k-\pi$.*

We define the set $\Pi^{\mathcal{S}[1..k]} = \{\pi_i^{\mathcal{S}} | 0 = \pi_0^{\mathcal{S}} < \pi_1^{\mathcal{S}} < \cdots < \pi_p^{\mathcal{S}} = k\}$ to be the set of all periods of a string $\mathcal{S}[1..k]$. $\pi_1^{\mathcal{S}}$, the smallest non-zero period of $\mathcal{S}[1..k]$ is called *the period* of $\mathcal{S}$. We use the terms *period* and *period length* synonymously.

A *substring* or a *factor* of a string $\mathcal{S}[1..k]$ is a contiguous block of symbols $\mathcal{S}[i..j]$. A *factorization* of $\mathcal{S}[1..k]$ is a way to break $\mathcal{S}$ into few factors. We only consider factorizations of a string into two factors: a *prefix* $\mathcal{S}[1..l]$ and a *suffix* $\mathcal{S}[l+1..k]$. Such a factorization is said to be *non-trivial* if neither of the two factors is equal to the empty string. Note that a factorization can be represented by a single integer which is the position at which the string is partitioned.

Definition 2.2 *Given a factorization $(\mathcal{S}[1..l], \mathcal{S}[l+1..k])$, a* local period *of the factorization is defined as a non-empty string that is consistent with both sides of the factorization. Namely, a string that matches the prefix $\mathcal{S}[1..l]$ aligned at its end and also matches suffix $\mathcal{S}[l+1..k]$ aligned at its start. The shortest local period of a factorization is called* the local period. *See Figure 1 for an example.*

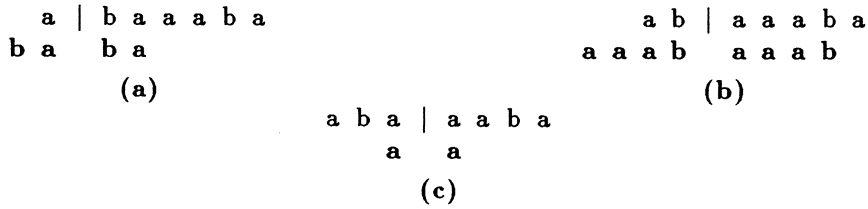

Figure 1. The local periods of the first three non-trivial factorizations of '*abaaaba*'. Note that in some cases the local period can overflow to either side; this happens when the local period is longer than either of the two factors. The factorization (b) is a critical factorization.

Definition 2.3 *A non-trivial factorization of a string $\mathcal{S}[1..k]$ is called a* critical factorization *if the local period of the factorization is of the same length as the period of $\mathcal{S}[1..k]$.*

The following theorem states that critical factorizations always exist. It is the basis for the Crochemore-Perrin string matching algorithm.

Theorem 2.4 *(The Critical Factorization Theorem, Cesari and Vincent [6, 25]) Let $\pi_1^{\mathcal{S}}$ be the period length of a string $\mathcal{S}[1..k]$. Then, if we consider any $\pi_1^{\mathcal{S}} - 1$ consecutive non-trivial factorizations, at least one is a critical factorization.*

3 The Crochemore-Perrin Algorithm

Crochemore and Perrin [12] used the Critical Factorization Theorem to obtain a simple and elegant linear-time constant-space string matching algorithm. The pattern preprocessing step of their algorithm, which is discusses in Section 7, also takes linear time and uses constant space. In the rest of this section we assume that the period length of the pattern and a critical factorization ($\mathcal{P}[1..\chi]$, $\mathcal{P}[\chi+1..m]$) of the pattern, such that $\chi < \pi_1^{\mathcal{P}}$, are given. We describe a somewhat simplified version of the Crochemore-Perrin algorithm.

The Crochemore-Perrin string matching algorithm tries to match the pattern aligned starting at a certain text position. It compares symbols starting from the middle of the pattern and tries first

to match the pattern suffix $\mathcal{P}[\chi + 1..m]$. Only then, after this suffix was discovered in the text, the algorithm tries to match the pattern prefix $\mathcal{P}[1..\chi]$ that was skipped.

LEMMA 3.1 *(Crochemore and Perrin [12]) Let* $(\mathcal{P}[1..\chi], \mathcal{P}[\chi + 1..m])$ *be a critical factorization of the pattern and let* $\rho \leq \max(\chi, m - \chi)$ *be the length of a local period of this factorization. Then* ρ *is a multiple of* $\pi_1^{\mathcal{P}}$*, the period length of the pattern.*

```
- π_1^P is the period length of the pattern P[1..m].
- (P[1..χ], P[χ + 1..m]) is a given critical factorization, such that χ < π_1^P.
- σ is the current text position that the pattern is aligned with.
- θ is the current text position we have to compare.
    σ = 1
    θ = 1 + χ
    while σ ≤ n − m + 1 do
            - Try to match the pattern suffix.
            - '&&' is the conditional and operator.
        while θ < σ + m && T[θ] = P[θ − σ + 1] do
            θ = θ + 1
        if θ < σ + m then  - If there was a mismatch.
            θ = θ + 1
            σ = θ − χ
        else      - The pattern suffix P[χ + 1..m] was matched.
                  - It remains to match the prefix P[1..χ].
                  - The original algorithm compares the symbols in the next statement
                  - from right to left. However, any order can be used.
            if T[σ..σ + χ − 1] = P[1..χ] then
                Report an occurrence of the pattern starting at text position σ.
            σ = σ + π_1^P
            if σ + χ > θ then
                θ = σ + χ
        end
    end
```

FIGURE 2. The Crochemore-Perrin algorithm.

THEOREM 3.2 *(Crochemore and Perrin [12]) There exist a constant-space linear-time string matching algorithm that makes at most* $2n - m$ *comparisons.*

Proof: The Crochemore-Perrin algorithm is given is Figure 2. We prove it correctness and show that is makes at most $2n - m$ symbol comparisons.

The algorithm aligns the pattern starting at some text position σ and tries to match the pattern suffix $\mathcal{P}[\chi + 1..m]$ with the text symbols that are aligned with it. Initially $\sigma = 1$, and later σ is incremented if there are mismatches or if occurrences of the pattern are discovered. The algorithm maintains the invariant that $\mathcal{T}[\sigma + \chi..\theta - 1] = \mathcal{P}[\chi + 1..\theta - \sigma]$, where θ is the text position that is compared next. There are two conditions in which the while loop that tries to match the pattern suffix terminates.

1. **Mismatch:** If the loop that matches the pattern suffix terminated with $\theta < \sigma + m$, then there was a mismatch $\mathcal{T}[\theta] \neq \mathcal{P}[\theta - \sigma + 1]$. Clearly, there can be no occurrence of the pattern starting at text position σ.

   Assume that an occurrence of the pattern starts at text position $\overline{\sigma}$, $\sigma < \overline{\sigma} \leq \theta - \chi$. Then, the critical factorization $(\mathcal{P}[1..\chi], \mathcal{P}[\chi + 1..m])$ must have a local period of length $\overline{\sigma} - \sigma$. See Figure 3.

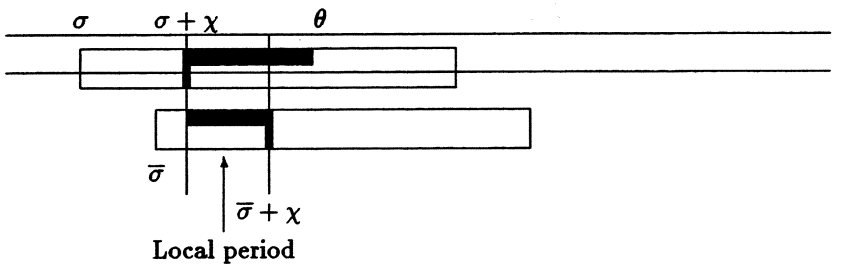

FIGURE 3. Applying critical factorizations. If $\mathcal{T}[\sigma+\chi..\theta-1] = \mathcal{P}[\chi+1..\theta-\sigma]$ and there is an occurrence of the pattern at text position $\overline{\sigma}$, $\sigma < \overline{\sigma} \leq \theta - \chi$, then the factorization $(\mathcal{P}[1..\chi], \mathcal{P}[\chi+1..m])$ has a local period of length $\overline{\sigma} - \sigma$.

Since $\overline{\sigma} - \sigma \leq m - \chi$, by Lemma 3.1, $\overline{\sigma} - \sigma$ is a multiple of $\pi_1^{\mathcal{P}}$. But then, by the definition of a period, $\mathcal{P}[\theta - \sigma + 1] = \mathcal{P}[\theta - \overline{\sigma} + 1]$ and $\mathcal{T}[\theta] \neq \mathcal{P}[\theta - \overline{\sigma} + 1]$. Therefore, there can be no occurrence of the pattern starting at text position $\overline{\sigma}$ and thus, the smallest text position at which an occurrence of the pattern may start is $\theta - \chi + 1$.

The algorithm proceeds by setting $\sigma = \theta - \chi + 1$.

2. **Match:** If the loop terminated with $\theta = \sigma + m$, then an occurrence of the pattern suffix $\mathcal{P}[\chi+1..m]$ was discovered at text position $\sigma + \chi$. The algorithm proceeds to match the pattern prefix $\mathcal{P}[1..\chi]$ that was skipped. If an occurrence of this pattern prefix is discovered, the algorithm can report an occurrence of the complete pattern starting at text position σ.

   In any case, the pattern is shifted ahead with respect to the text by $\pi_1^{\mathcal{P}}$ positions since an occurrence of the pattern at any text position $\overline{\sigma}$, such that $\sigma < \overline{\sigma} < \sigma + \pi_1^{\mathcal{P}}$, would imply that the critical factorization $(\mathcal{P}[1..\chi], \mathcal{P}[\chi+1..m])$ has a local period whose length is smaller than $\pi_1^{\mathcal{P}}$.

   Note that if after incrementing σ by $\pi_1^{\mathcal{P}}$, $\sigma + \chi < \theta$, then $\mathcal{T}[\sigma..\theta-1] = \mathcal{P}[1..\theta-\sigma]$ and in particular $\mathcal{T}[\sigma+\chi..\theta-1] = \mathcal{P}[\chi+1..\theta-\sigma]$. Therefore, the invariant is already maintained and there is no need to go back and compare parts of the pattern and the text that were compared earlier.

It remains to count the number of comparisons made by the algorithm. There are at most $n - \chi$ comparisons made in the loop that matches the pattern suffix since θ is incremented after each comparison is made and initially $\theta = \chi + 1$. The second comparison statement that matches the pattern prefix makes at most χ comparisons each time it is reached. But then, σ is incremented by $\pi_1^{\mathcal{P}}$ and $\chi < \pi_1^{\mathcal{P}}$. Thus, there are at most $n - m + \chi$ comparisons made by this statement throughout the execution of the algorithm and the total number of comparisons is at most $2n - m$. □

4 Saving Comparisons

The Crochemore-Perrin algorithm is oblivious in the sense that it sometimes "forgets" comparisons that it made and repeats them later. In this section we show how to avoid some of the repeated comparisons. The obvious implementation of the suggested algorithm uses $O(m)$ memory registers to store the periods of the pattern. Section 6 shows how to reduce the space requirements.

THEOREM 4.1 *The Crochemore-Perrin string matching algorithm can be modified in such a way that it takes linear-time and makes at most* $n + \lfloor \frac{\max(\pi_1^{\mathcal{P}}, m-\pi_1^{\mathcal{P}})}{m}(n-m) \rfloor$ *comparisons.*

```
– π_1^P is the period length of the pattern P[1..m].
– (P[1..χ], P[χ + 1..m]) is a given critical factorization, such that χ < π_1^P.
– σ is the current text position that the pattern is aligned with.
– θ is the current text position we have to compare.
– τ is the text position immediately after the last discovered pattern suffix P[χ + 1..m].
– The algorithm does not compare text symbols at positions that are smaller than τ.
    σ = 1
    θ = 1 + χ
    τ = 0
    while σ ≤ n − m + 1 do
          – Try to match the pattern suffix.
          – '&&' is the conditional and operator.
        while θ < σ + m && T[θ] = P[θ − σ + 1] do
            θ = θ + 1
        if θ < σ + m then  – If there was a mismatch.
            θ = θ + 1
            σ = θ − χ
            if σ < τ then  – Maintain the invariant T[σ..τ − 1] = P[1..τ − σ].
                σ = min{τ − m + π | π ∈ Π^P and τ − m + π ≥ σ}
                if σ + χ > θ then
                    θ = σ + χ
            end
        else      – The pattern suffix P[χ + 1..m] was matched.
                  – It remains to match the prefix P[1..χ].
            α = max(σ, τ)
            if T[α..σ + χ − 1] = P[α − σ + 1..χ] then
                Report an occurrence of the pattern starting at text position σ.
            σ = σ + π_1^P
            τ = θ
            if σ + χ > θ then
                θ = σ + χ
        end
    end
```

FIGURE 4. The modified Crochemore-Perrin algorithm.

Proof: The modified Crochemore-Perrin algorithm is given is Figure 4. The main observation in the modified algorithm is that when the original Crochemore-Perrin algorithm tries to match the pattern prefix $\mathcal{P}[1..\chi]$, this prefix might overlap the pattern suffix $\mathcal{P}[\chi+1..m]$ that was previously discovered in the text. It is possible to avoid repeating some comparisons by keeping track of suffix-prefix overlaps. For this purpose, the modified algorithm keeps an additional index τ which holds the text position immediately after the last discovered pattern suffix $\mathcal{P}[\chi+1..m]$.

In addition to the invariant $\mathcal{T}[\sigma+\chi..\theta-1] = \mathcal{P}[\chi+1..\theta-\sigma]$ that was maintained in the algorithm that was given in the previous section, the modified algorithm maintains a second invariant: If $\sigma < \tau$, then $\mathcal{T}[\sigma..\tau-1] = \mathcal{P}[1..\tau-\sigma]$. Namely, if there would be an occurrence of the pattern starting at text position σ and this occurrence overlapped the last discovered pattern suffix $\mathcal{P}[\chi+1..m]$, then the overlapping parts must be are identical. Therefore, if $\sigma < \tau$, then it suffices to compare $\mathcal{T}[\tau..\sigma+\chi-1]$ to $\mathcal{P}[\tau-\sigma..\chi]$ to check if $\mathcal{T}[\sigma..\sigma+\chi-1] = \mathcal{P}[1..\chi]$.

Note that suffix-prefix overlaps correspond to periods since $\mathcal{P}[\pi+1..m] = \mathcal{P}[1..m-\pi]$ if and only if $\pi \in \Pi^{\mathcal{P}}$. The second invariant is clearly maintained after the pattern suffix $\mathcal{P}[\chi+1..m]$ is discovered in the text and the pattern is shifted ahead by $\pi_1^{\mathcal{P}}$ positions. The algorithm makes sure that this invariant is maintained each time that a mismatch is encountered by shifting the pattern further ahead until it is maintained, if necessary.

The correctness of the algorithm follows similarly to Theorem 3.2. We show that the algorithm makes at most $n + \lfloor \frac{\max(\pi_1^{\mathcal{P}}, m-\pi_1^{\mathcal{P}})}{m}(n-m) \rfloor$ comparisons and takes linear time.

Partition the execution of the algorithm into phases. A phase ends after the algorithm has found an occurrence of the pattern suffix $\mathcal{P}[\chi+1..m]$ in the text and it tried to match the pattern prefix $\mathcal{P}[1..\chi]$, or when the end of the text is reached. The following phase starts immediately after the algorithm has shifted the pattern ahead with respect to the text by $\pi_1^{\mathcal{P}}$ positions. Let σ^ϕ denote the value of σ, the text position that the pattern is aligned with, at the end of the phase number ϕ. Then, in the first phase $\sigma^1 \geq 1$ and in the last phase $\sigma^l \leq n-m+1$. In phase number ϕ, $2 \leq \phi \leq l$, $\sigma^{\phi-1} + \pi_1^{\mathcal{P}} \leq \sigma^\phi$. Note that during phase number ϕ, $\tau = \sigma^{\phi-1} + m$ is the text position position immediately after the last discovered pattern suffix $\mathcal{P}[\chi+1..m]$.

We use a simple policy of charging comparisons to text symbols: each comparison is charged to the text symbol that is compared and the charge might be later transferred to a smaller text position. Using this charging policy it is clear that at the beginning of phase number ϕ all text positions that are larger than or equal to τ are not charged with any comparison.

The charges are transferred as follows. The comparisons that were charged during phase number ϕ to text positions between τ and $\sigma^\phi + \chi$ are transferred χ positions back. Note that the number of these comparisons is bounded by $\sigma^\phi - \sigma^{\phi-1} - \pi_1^{\mathcal{P}}$ and only the $m - \pi_1^{\mathcal{P}}$ text positions that are larger than or equal to $\sigma^{\phi-1} + \pi_1^{\mathcal{P}}$ might be charged with a second comparison. This charge transfer has the advantage that all text symbols at positions between $\max(\sigma^\phi, \tau)$ and $\sigma^\phi + \chi$ do not have a comparison charged to them. Each of these text positions are charged with at most one comparison when the algorithm tries to match the pattern prefix $\mathcal{P}[1..\chi]$.

Clearly, a second comparison might be charged to a text position only when the charges are transferred. We obtain an upper bound on the number of text symbols that are charged with a second comparison in phase number ϕ by bounding the ratio between the number of these symbols to $\sigma^\phi - \sigma^{\phi-1}$, the number of positions by which the pattern was shifted in phase number ϕ. If this ratio can be bounded by a constant c in all phases, then there are at most $\lfloor c(\sigma^\phi - \sigma^{\phi-1}) \rfloor$ text symbols charged with a second comparison in phase number ϕ and the total number of text symbols charged with two comparisons is bounded by $\lfloor c \sum_{i=2}^{l} (\sigma^\phi - \sigma^{\phi-1}) \rfloor \leq \lfloor c(n-m) \rfloor$.

There are two cases:

1. There are at most $\sigma^\phi - \sigma^{\phi-1} - \pi_1^{\mathcal{P}}$ text positions charged with a second comparison in phase number ϕ, but in any case no more than $m - \pi_1^{\mathcal{P}}$. The ratio $\frac{\min(\sigma^\phi - \sigma^{\phi-1} - \pi_1^{\mathcal{P}}, m-\pi_1^{\mathcal{P}})}{\sigma^\phi - \sigma^{\phi-1}}$ is maximized for $\sigma^\phi - \sigma^{\phi-1} = m$ and is bounded by $\frac{m-\pi}{m}$.

2. If $m - \pi_1^{\mathcal{P}} > \pi_1^{\mathcal{P}}$, then it is possible to achieve better bounds. If $\phi^{\phi} = \phi^{\phi-1} + \pi_1^{\mathcal{P}}$, then there are clearly no text symbols charged with a second comparison in phase number ϕ since there were no charges transferred.

   Otherwise, there was at least one mismatch while the algorithm was trying to match the pattern suffix $\mathcal{P}[\chi+1..m]$. Since $\chi < \pi_1^{\mathcal{P}}$, we have that $\sigma^{\phi} - \sigma^{\phi-1} > m - \pi_1^{\mathcal{P}}$. The number of text symbols charged with a second comparison in phase number ϕ is bounded by $\sigma^{\phi} - \sigma^{\phi-1} + \pi_1^{\mathcal{P}} - m$ and only text symbols that are larger than or equal to $\tau - \pi_1^{\mathcal{P}}$ might be charged with a second comparison. Thus, in any case there are not more than $\pi_1^{\mathcal{P}}$ such symbols. The ratio $\frac{\min(\sigma^{\phi} - \sigma^{\phi-1} + \pi_1^{\mathcal{P}} - m, \pi_1^{\mathcal{P}})}{\sigma^{\phi} - \sigma^{\phi-1}}$ is maximized for $\sigma^{\phi} - \sigma^{\phi-1} = m$ and is bounded by $\frac{\pi_1^{\mathcal{P}}}{m}$.

Therefore, the total number of comparisons made by the modified algorithm is bounded by $n + \lfloor \frac{\min(\pi_1^{\mathcal{P}}, m - \pi_1^{\mathcal{P}})}{m}(n-m) \rfloor$.

It remains to show that the algorithm takes linear time. The only part which might take longer is the search for the smallest period length of the pattern which is larger than or equal to $\sigma - \tau + m$ when $\sigma < \tau$. It is possible to precompute a table in the preprocessing step that would provide this information in a single step. In Theorem 6.1 we show how this step can be implemented without precomputing such a table. □

5 The Periodicity Structure

The following are well known facts about periods.

Fact 5.1 *If a string $\mathcal{S}[1..k]$ has period length π_a, then it has period length π_b, such that $\pi_a \leq \pi_b$, if and only if the suffix $\mathcal{S}[\pi_a + 1..k]$ has period length $\pi_b - \pi_a$.*

Proof: Assume that $\mathcal{S}[1..k - \pi_a] = \mathcal{S}[\pi_a + 1..k]$ and $\pi_a \leq \pi_b$. Clearly $\mathcal{S}[1..k - \pi_b] = \mathcal{S}[\pi_b + 1..k]$ if and only if $\mathcal{S}[\pi_a + 1..k - \pi_b + \pi_a] = \mathcal{S}[\pi_b + 1..k]$. □

Fact 5.2 *If a string $\mathcal{S}[1..k]$ has period lengths π_a and π_b, such that $\pi_a \leq \pi_b$ and $\pi_a + \pi_b \leq k$, then it also has period length $\pi_b - \pi_a$.*

Proof: Assume that $\mathcal{S}[1..k - \pi_a] = \mathcal{S}[\pi_a + 1..k]$ and $\mathcal{S}[1..k - \pi_b] = \mathcal{S}[\pi_b + 1..k]$. If $\pi_a \leq \pi_b$ and $\pi_a + \pi_b \leq k$, then $\mathcal{S}[\pi_b - \pi_a + 1..\pi_b] = \mathcal{S}[\pi_b + 1..\pi_b + \pi_a] = \mathcal{S}[1..\pi_a]$ and $\mathcal{S}[\pi_b + 1..k] = \mathcal{S}[1..k - \pi_b] = \mathcal{S}[\pi_a + 1..k - \pi_b + \pi_a]$. Thus, $\mathcal{S}[1..k - \pi_b + \pi_a] = \mathcal{S}[\pi_b - \pi_a + 1..k]$. □

Lemma 5.3 *Let $\pi_{\alpha}^{\mathcal{S}}, \pi_{\alpha+1}^{\mathcal{S}} \in \Pi^{\mathcal{S}}$ be period lengths of a string $\mathcal{S}[1..k]$ and let $\overline{\pi} = \pi_{\alpha+1}^{\mathcal{S}} - \pi_{\alpha}^{\mathcal{S}}$. Then,*

1. *$\pi_{\alpha}^{\mathcal{S}} + \delta\overline{\pi} \in \Pi^{\mathcal{S}}$ for non-negative integral values of δ, such that $\pi_{\alpha}^{\mathcal{S}} + \delta\overline{\pi} \leq k$.*
2. *All other period lengths in $\Pi^{\mathcal{S}}$ which are larger than $\pi_{\alpha}^{\mathcal{S}}$ are also larger than $k - \overline{\pi}$.*

Proof: The proof follows from simple properties of periods:

1. By Fact 5.1, the suffix $\mathcal{S}[\pi_{\alpha}^{\mathcal{S}} + 1..k]$ has a period length $\overline{\pi}$. Any integral multiple $\delta\overline{\pi} \leq k - \pi_{\alpha}^{\mathcal{S}}$ is also a period length of this suffix. By Fact 5.1, $\pi_{\alpha}^{\mathcal{S}} + \delta\overline{\pi}$ is a period length of $\mathcal{S}[1..k]$.
2. Let $\pi_{\gamma}^{\mathcal{S}}$ be the smallest period length in $\Pi^{\mathcal{S}}$ which is larger than $\pi_{\alpha}^{\mathcal{S}}$ and is not of the form $\pi_{\alpha}^{\mathcal{S}} + \delta\overline{\pi}$. Then $\pi_{\gamma}^{\mathcal{S}} \geq \pi_{\alpha+1}^{\mathcal{S}}$ and by Fact 5.1, the suffix $\mathcal{S}[\pi_{\alpha}^{\mathcal{S}} + 1..k]$ has period lengths $\overline{\pi}$ and $\pi_{\gamma}^{\mathcal{S}} - \pi_{\alpha}^{\mathcal{S}}$.

   If $\pi_{\gamma}^{\mathcal{S}} \leq k - \overline{\pi}$, then $\pi_{\gamma}^{\mathcal{S}} - \pi_{\alpha}^{\mathcal{S}} + \overline{\pi} \leq k - \pi_{\alpha}^{\mathcal{S}}$ and by Fact 5.2, the suffix $\mathcal{S}[\pi_{\alpha}^{\mathcal{S}} + 1..k]$ has also a period of length $\pi_{\gamma}^{\mathcal{S}} - \pi_{\alpha}^{\mathcal{S}} - \overline{\pi}$. By Fact 5.1, $\mathcal{S}[1..k]$ has a period length $\pi_{\gamma}^{\mathcal{S}} - \overline{\pi}$ in contradiction to the minimality of $\pi_{\gamma}^{\mathcal{S}}$. □

The following lemma shows that the periodicity structure of a string can be represented economically. Note that 0 and k are always period lengths of a string $\mathcal{S}[1..k]$ and do not have to be specified by the representation.

LEMMA 5.4 *Given a string $\mathcal{S}[1..k]$, it is possible to represent all period lengths $\pi_i^{\mathcal{S}} \in \Pi^{\mathcal{S}}$, such that $\pi_i^{\mathcal{S}} \leq k - \lfloor \frac{1}{2}(\frac{2}{3})^{c-1}k \rfloor$, by specifying only $c \geq 1$ period lengths. Furthermore, it is possible to compute this representation from the periods of $\mathcal{S}[1..k]$ in linear time and using constant space while the periods are given in an increasing order, and it is also possible to generate the periods from this representation in an increasing order in time that is linear in the number of generated periods and using constant space.*

Proof: We first show how to construct the economic representation of the periods. The construction takes linear time and uses constant space in addition to the c memory registers that store the representation. The main idea is to use Lemma 5.3 to generate larger period lengths from small ones. Periods which can be generated from smaller periods do not need to be stored.

Initially let $\hat{\pi}_1^{\mathcal{S}} = \pi_1^{\mathcal{S}}$ and $c = 1$. Assume that the remaining periods of $\mathcal{S}[1..k]$ are given in an increasing order starting with $\pi_2^{\mathcal{S}}$ and let $\pi_\alpha^{\mathcal{S}}$ be the next period length.

If $\pi_\alpha^{\mathcal{S}} - \pi_{\alpha-1}^{\mathcal{S}} = \pi_{\alpha-1}^{\mathcal{S}} - \pi_{\alpha-2}^{\mathcal{S}}$, then $\pi_\alpha^{\mathcal{S}}$ is given by the period lengths $\pi_{\alpha-2}^{\mathcal{S}}$ and $\pi_{\alpha-1}^{\mathcal{S}}$, and it does not have to be stored. Otherwise, $\pi_\alpha^{\mathcal{S}}$ is not given by the period lengths $\pi_{\alpha-2}^{\mathcal{S}}$ and $\pi_{\alpha-1}^{\mathcal{S}}$ and if there is space to store more periods, let $\hat{\pi}_{c+1}^{\mathcal{S}} = \pi_\alpha^{\mathcal{S}}$, and increment c by one. (Note, that the period length $\pi_\alpha^{\mathcal{S}}$ might be given by previous periods, but it is more convenient not to check for this condition; e.g. the string '*aabaaabaa*' has a period of length 8 which is given by the period lengths 0 and 4, but 7 is also a period length of this string, $7 - 4 \neq 4 - 0$ and the representation of all periods will consists of the period lengths 4, 7 and 8.) In both cases, all period lengths of $\mathcal{S}[1..k]$ which are smaller than or equal to $\pi_\alpha^{\mathcal{S}}$ are represented by the c periods $\hat{\pi}_1^{\mathcal{S}}, \cdots, \hat{\pi}_c^{\mathcal{S}}$.

It remains to shows that all period lengths $\pi_i^{\mathcal{S}} \in \Pi^{\mathcal{S}}$, such that $\pi_i^{\mathcal{S}} \leq k - \lfloor \frac{1}{2}(\frac{2}{3})^{c-1}k \rfloor$, are specified by the c periods $\hat{\pi}_1^{\mathcal{S}}, \cdots, \hat{\pi}_c^{\mathcal{S}}$. We maintain inductively that $k - \pi_{\alpha-2}^{\mathcal{S}} \leq (\frac{2}{3})^{c-1}k$. The induction hypothesis clearly holds initially since $\pi_{\alpha-2}^{\mathcal{S}} = \pi_0^{\mathcal{S}} = 0$ and $c = 1$.

If $\pi_\alpha^{\mathcal{S}} - \pi_{\alpha-1}^{\mathcal{S}} = \pi_{\alpha-1}^{\mathcal{S}} - \pi_{\alpha-2}^{\mathcal{S}}$, then $\pi_\alpha^{\mathcal{S}}$ was specified by the previous periods. Clearly, $k - \pi_{\alpha-1}^{\mathcal{S}} < k - \pi_{\alpha-2}^{\mathcal{S}} \leq (\frac{2}{3})^{c-1}k$ and the induction hypothesis holds.

Otherwise, $\pi_\alpha^{\mathcal{S}}$ is the smallest period that is not specified by the c periods $\hat{\pi}_1^{\mathcal{S}}, \cdots, \hat{\pi}_c^{\mathcal{S}}$. By Lemma 5.3, $k - \pi_\alpha^{\mathcal{S}} < \pi_{\alpha-1}^{\mathcal{S}} - \pi_{\alpha-2}^{\mathcal{S}}$. But $k - \pi_\alpha^{\mathcal{S}} \leq k - \pi_{\alpha-1}^{\mathcal{S}} - 1$ and,

$$2(k - \pi_\alpha^{\mathcal{S}}) < \pi_{\alpha-1}^{\mathcal{S}} - \pi_{\alpha-2}^{\mathcal{S}} + k - \pi_{\alpha-1}^{\mathcal{S}} - 1 < k - \pi_{\alpha-2}^{\mathcal{S}} - 1.$$

Therefore, $k - \pi_\alpha^{\mathcal{S}} < \lfloor \frac{1}{2}(k - \pi_{\alpha-2}^{\mathcal{S}}) \rfloor$. By the induction hypothesis $k - \pi_{\alpha-2}^{\mathcal{S}} \leq (\frac{2}{3})^{c-1}k$ and,

$$k - \pi_\alpha^{\mathcal{S}} < \lfloor \frac{1}{2}(\frac{2}{3})^{c-1}k \rfloor.$$

Thus, all period lengths of $\mathcal{S}[1..k]$ that are smaller than or equal to $k - \lfloor \frac{1}{2}(\frac{2}{3})^{c-1}k \rfloor$ are given by the c periods $\hat{\pi}_1^{\mathcal{S}}, \cdots, \hat{\pi}_c^{\mathcal{S}}$.

We show that $k - \pi_{\alpha-1}^{\mathcal{S}} \leq (\frac{2}{3})^c k$ and the induction hypothesis still holds. Recall that by Lemma 5.3, $\pi_\alpha^{\mathcal{S}} > k - (\pi_{\alpha-1}^{\mathcal{S}} - \pi_{\alpha-2}^{\mathcal{S}})$. Assume that $\pi_{\alpha-1}^{\mathcal{S}} - \pi_{\alpha-2}^{\mathcal{S}} \leq \frac{1}{3}(k - \pi_{\alpha-2}^{\mathcal{S}})$. Then by Lemma 5.3, $\pi_{\alpha-2}^{\mathcal{S}} + 2(\pi_{\alpha-1}^{\mathcal{S}} - \pi_{\alpha-2}^{\mathcal{S}})$ is also a period length of $\mathcal{S}[1..k]$. But then,

$$\begin{aligned} \pi_{\alpha-2}^{\mathcal{S}} + 2(\pi_{\alpha-1}^{\mathcal{S}} - \pi_{\alpha-2}^{\mathcal{S}}) &\leq \pi_{\alpha-2}^{\mathcal{S}} + \frac{2}{3}(k - \pi_{\alpha-2}^{\mathcal{S}}) \\ &= k - \frac{1}{3}(k - \pi_{\alpha-2}^{\mathcal{S}}) \\ &\leq k - (\pi_{\alpha-1}^{\mathcal{S}} - \pi_{\alpha-2}^{\mathcal{S}}) < \pi_\alpha^{\mathcal{S}} \end{aligned}$$

But since the periods are given in an increasing order this is not possible. Therefore, $\pi_{\alpha-1}^{\mathcal{S}} - \pi_{\alpha-2}^{\mathcal{S}} > \frac{1}{3}(k - \pi_{\alpha-2}^{\mathcal{S}})$, establishing that,

$$k - \pi_{\alpha-1}^{\mathcal{S}} < k - \pi_{\alpha-2}^{\mathcal{S}} - \frac{1}{3}(k - \pi_{\alpha-2}^{\mathcal{S}}) \leq \frac{2}{3}(k - \pi_{\alpha-2}^{\mathcal{S}}) \leq (\frac{2}{3})^c k.$$

Given the representation $\hat{\pi}_1^{\mathcal{S}}, \cdots, \hat{\pi}_c^{\mathcal{S}}$, one can clearly generate all periods $\pi_i^{\mathcal{S}} \in \Pi^{\mathcal{S}}$, such that $\pi_i^{\mathcal{S}} \leq \max(k - \lfloor \frac{1}{2}(\frac{2}{3})^{c-1}k \rfloor, \hat{\pi}_c^{\mathcal{S}})$, in an increasing order, in time that is linear in the number of periods generated and using constant space. Sometimes, it is possible to continue and generate larger periods, but periods which are not specified by the representation might be skipped. □

COROLLARY 5.5 *The set* $\Pi^{\mathcal{S}[1..k]}$ *of all periods of a string* $\mathcal{S}[1..k]$ *can be represented by specifying only* $\lceil \log_{\frac{3}{2}} k \rceil$ *periods.*

Remark. The compact representation of periods of a string is not new. Galil and Seiferas [16] used similar arguments in a variant of the Knuth-Morris-Pratt string matching algorithm that uses only $O(\log m)$ space. Guibas and Odlyzko [20] characterized all possible periodicity structures of a string of length k and showed that there are $k^{\Theta(\log k)}$ such structures, independent of the alphabet size. Thus, any encoding of the periodicity structure requires $\Omega(\log^2 k)$ bits and our representation can not be uniformly improved by more than a constant multiplicative factor.

6 SAVING SPACE

This section shows how to use the economic representation of the periodicity structure of the pattern in the modified Chrochemore-Perrin algorithm that was given in Section 4.

THEOREM 6.1 *The modified Crochemore-Perrin algorithm from Section* 4 *can be implemented in linear-time using only* $O(\log m)$ *auxiliary memory registers.*

Proof: The algorithm uses constant space except for storing of the periods of the pattern. By Corollary 5.5, the periods can be represented in $O(\log m)$ memory registers. By Lemma 5.4, the periods can be generated from this representation in an increasing order, in time that is linear in the number of periods generated and using constant space.

The periods are used only in one place in the algorithm where the smallest period of the pattern that is larger than or equal to $\sigma - \tau + m$ is needed. But σ only increases during the execution of the algorithm, so as long that τ is fixed, the periods that are needed also increase and can be found by scanning the periods in an increasing order. The time is clearly bounded by the amount of increase of σ, and therefore is linear.

However, τ increases each time an occurrence of the pattern suffix $\mathcal{P}[\chi + 1..m]$ is discovered in the text. In this case the algorithm returns to generate the periods in an increasing order starting from the smallest period. Note, that in this case $\tau = \sigma + m - \pi_1^{\mathcal{P}}$, the algorithm will need only periods that are larger than $\pi_1^{\mathcal{P}}$, and the time to generate the periods will be bounded by the amount of increase of σ. Thus, the algorithm still takes linear time. □

If only constant space is available, then a part of the periodicity structure of the pattern can still be stored. The resulting algorithm is a hybrid between the Crochemore-Perrin algorithm given in Section 3 and the modified algorithm from Section 4.

THEOREM 6.2 *If* $c \geq 1$ *registers are available to store the periodicity structure of the pattern, then the modified Crochemore-Perrin algorithm can be implemented in linear time and constant space. It makes at most* $n + \lfloor \frac{1}{2-(\frac{2}{3})^{c-1}}(n-m) \rfloor$ *comparisons.*

Proof: Since the period $\pi_1^{\mathcal{P}}$ is used in the original algorithm, the number of registers used to store the periodicity structure is larger than one. By Lemma 5.4 all period lengths $\pi_\alpha^{\mathcal{P}} \leq m - \lfloor \frac{1}{2}(\frac{2}{3})^{c-1}m \rfloor$ can be represented by $c \geq 1$ registers.

Recall the proof of Theorem 4.1. In phase number ϕ, if $\sigma^\phi - \sigma^{\phi-1} \leq m - \lfloor \frac{1}{2}(\frac{2}{3})^{c-1}m \rfloor$, then the algorithm can proceed as in Theorem 6.1. The problem arises if $\sigma < \tau$ and $\sigma^\phi - \sigma^{\phi-1} > m - \lfloor \frac{1}{2}(\frac{2}{3})^{c-1}m \rfloor$. Since the algorithm cannot maintain the invariant that $\mathcal{T}[\sigma..\tau - 1] = \mathcal{P}[1..\tau - \sigma]$ it will behave as the original Crochemore-Perrin algorithm of Section 3 and compare the complete prefix $\mathcal{P}[1..\chi]$ of the pattern if necessary.

This may cause second charges to $\min(\pi_1^{\mathcal{P}}, m - \pi_1^{\mathcal{P}})$ text symbols while $m \geq \sigma^{\phi} - \sigma^{\phi-1} > m - \lfloor \frac{1}{2}(\frac{2}{3})^{c-1} m \rfloor$. Thus in phase number ϕ, the ratio between the number of text symbols that are charged with a second comparison to $\sigma^{\phi} - \sigma^{\phi-1}$ is bounded by,

$$\frac{\min(\pi_1^{\mathcal{P}}, m - \pi_1^{\mathcal{P}})}{m - \lfloor \frac{1}{2}(\frac{2}{3})^{c-1} m \rfloor} \leq \frac{\frac{1}{2}}{1 - \frac{1}{2}(\frac{2}{3})^{c-1}} = \frac{1}{2 - (\frac{2}{3})^{c-1}}$$

establishing the claimed bound. □

7 The Pattern Preprocessing

The pattern preprocessing step of the Crochemore and Perrin algorithm takes linear time, uses constant space and make at most $5m$ symbol comparisons. However, it uses order comparisons that may result in less-than, equal-to, or greater-than answers. This preprocessing is not sufficient for our purpose since it does not find all the periods of the pattern. In fact, if the period of the pattern is longer than half of the pattern length, then the Crochemore-Perrin pattern preprocessing step does not compute it at all.

THEOREM 7.1 *The pattern preprocessing step of the algorithms presented in this paper takes linear time and uses constant space. It uses order comparisons to find a critical factorization of the pattern.*

Proof: The preprocessing consists of two parts:

1. A critical factorization of the pattern is computed by Crochemore and Perrin's pattern preprocessing algorithm. This computation requires the use of order comparisons.

2. Galil and Seiferas [18] and Crochemore and Rytter [13] show that their linear-time constant-space string matching algorithms can find all overhanging occurrences of the pattern in the text and therefore find all period lengths of the pattern.

   These algorithms find the periods in an increasing order of their length as required in Lemma 5.4. The construction of the economic representation of the periods proceeds as the periods are found and does not require any additional symbol comparisons. It takes linear time and uses constant space, except for the registers which are used to store the representation.

The number of comparisons made is obviously linear with a constant that is not very large. □

8 Open Problems

There are several remaining open problems about the exact comparison complexity of string matching and of related string problems. Many of the problems listed in Breslauer and Galil's paper [5] can also be asked in the context of constant space algorithms. Two problems which are directly related to this work are:

1. What is the exact number of comparisons required by constant-space string matching algorithms?

2. Is it necessary to use order comparisons in order to find a critical factorization of a string in linear time?

9 Acknowledgments

I am in debt to Alberto Apostolico for discussions that led to the results given in this paper and for comments on early versions of the paper. Uri Zwick provided several suggestions that helped to improve the presentation.

REFERENCES

[1] A. V. Aho. Algorithms for Finding Patterns in Strings. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 257–300. Elsevier Science Publishers B. V., Amsterdam, the Netherlands, 1990.

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

[3] A. B. Borodin, M. J. Fischer, D. G. Kirkpatrick, N. A. Lynch, and M. Tompa. A time-space tradeoff for sorting on non-oblivious machines. In *Proc. 20th IEEE Symp. on Foundations of Computer Science*, pages 294–301, 1979.

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

[5] D. Breslauer and Z. Galil. Efficient Comparison Based String Matching. *J. Complexity*, 1993. To appear.

[6] Y. Cesari and M. Vincent. Une caractérisation des mots periodiques. *C.R. Acad. Sci. Paris*, 286(A):1175–1177, 1978.

[7] R. Cole. Tight bounds on the complexity of the Boyer-Moore pattern matching algorithm. In *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, pages 224–233, 1991.

[8] 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.

[9] R. Cole, R. Hariharan, M.S. Paterson, and U. Zwick. Which patterns are hard to find. In *Proc. 2nd Israeli Symp. on Theoretical Computer Science*, 1993. To appear.

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

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

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

[13] 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, 1993.

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

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

[16] Z. Galil and J. Seiferas. Saving space in fast string-matching. *SIAM J. Comput.*, 2:417–438, 1980.

[17] Z. Galil and J. Seiferas. Linear-time string-matching using only a fixed number of local storage locations. *Theoret. Comput. Sci.*, 13:331–336, 1981.

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

[19] M. Geréb-Graus and M. Li. Three one-way heads cannot do string matching. Manuscript, 1990.

[20] L. Guibas and A. M. Odlyzko. Periods in strings. *Journal of Combinatorial Theory, Series A*, 30:19–42, 1981.

[21] T. Jiang and M. Li. k One-way Heads Cannot Do String Matching. In *Proc. 25th ACM Symp. on Theory of Computing*, 1993. To appear.

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

[23] M. Li. Lower bounds on string-matching. Technical Report TR 84–63, Cornell University, Department of Computer Science, 1984.

[24] M. Li and Y. Yesha. String-matching cannot be done by a two-head one-way deterministic finite automaton. *Inform. Process. Lett.*, 22:231–235, 1986.

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

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