Saving Comparisons in the Crochemore-Perrin String Matching Algorithm

(preliminary version)

Dany Breslauer*
CWI

CUCS-052-92

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

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

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*. To solve the string matching problem one has only to be able to compare symbols of the input strings. In this paper we study the exact comparison complexity of the string matching problem. We assume that the only access an algorithm has 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 [20] 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 $2n-$

*Partially supported by the IBM Graduate Fellowship while studying at Columbia University and by the European Research Consorium for Informatics and Mathematics postdoctoral fellowship. Part of the work was done while this author was visiting at Università de L'Aquila, L'Aquila, Italy in summer 1991.

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 performed in the pattern preprocessing step that can compare even all pairs of pattern symbols for free.

Research on the exact number of comparisons required to solve the string matching problem has been stimulated by Colussi's [9] discovery of an algorithm that makes at most $n + \frac{1}{2}(n - m)$ comparisons. This bound was improved by Galil and Giancarlo [14], Breslauer and Galil [5] and most recently by Cole and Hariharan [8] who show that the string matching problem can be solved using at most $n + \frac{8}{3}(n - m)$ comparisons[1]. Lower bounds given by Galil and Giancarlo [13], Zwick and Paterson [24], and Cole and Hariharan [8] still leave a small gap between the lower and upper bounds.

All the algorithm mentioned above use $O(m)$ auxiliary space. At a certain time the string matching problem was conjectured to have a time-space tradeoff [3, 16]. This conjecture was later disproved when a linear-time constant-space algorithm was discovered by Galil and Seiferas [17]. Their algorithm can even be implemented on a six-head two-way finite automaton in linear time. It is still an open problem whether a k-head one-way finite automaton can do string matching (for $k \leq 3$ the answer is negative [18, 21, 22]).

The computation model used 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. This notion of space is different from the one used in [3] since a constant-space algorithm in our model has a logarithmic capacity.

Crochemore and Perrin [11] discovered a simple linear-time constant-space string matching algorithm that makes at most $2n - m$ comparisons. The Galil-Seiferas algorithm uses more comparisons. Crochemore and Rytter [12] show how to reduce the number of comparisons made by the Galil-Seiferas algorithm by a better choice of parameters. Crochemore [10] gives another constant-space string matching algorithm. The comparison bounds achieved by Crochemore and Rytter [12] and by Crochemore [10] 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 each 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 a pattern suffix that occurs in the text. The modified algorithm takes linear time and uses $O(m)$ auxiliary space. It makes at most

[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.

$n + \lfloor \frac{\min(\pi_1, m-\pi_1)}{m}(n-m) \rfloor \leq n + \lfloor \frac{1}{2}(n-m) \rfloor$ comparisons, where π_1 denotes the period length of the pattern.

This bound is the same as the bound given by Galil and Giancarlo [14] for Colussi's [9] algorithm. Our analysis is much simpler.

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

3. If only c registers are available to store the periodicity structure of the pattern, then the algorithm makes at most $n + \lfloor \frac{\sqrt{2}^{c-1}}{\sqrt{2}^{c+1}-1}(n-m) \rfloor$ comparisons.

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

The pattern preprocessing step of the modified algorithms can be implemented in linear time. The amount of space used is proportional to the portion of the periodicity structure of the pattern that has to be stored.

We proceed with 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.

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 $i = 1..k-\pi$.*

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 $\mathcal{S}[1..k]$. $\pi_1^{\mathcal{S}}$, the smallest non-zero period of $\mathcal{S}$ is called *the period* of $\mathcal{S}$. Note that by the definition of a period, a string $\mathcal{S}[1..k]$ has a period of length π if and only if $\mathcal{S}[1..k-\pi] = \mathcal{S}[\pi+1..k]$. We use the terms *period* and *period length* synonymously.

The following facts about periods are well known and trivial to prove.

Fact 2.2 *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 $\pi_b - \pi_a$.*

Fact 2.3 *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 $\pi_b - \pi_a$.*

Lemma 2.4 *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.* $\pi^{\mathcal{S}}_{\alpha} + \delta\overline{\pi} \in \Pi^{\mathcal{S}}$ *for non-negative integral values of* δ*, such that* $\pi^{\mathcal{S}}_{\alpha} + \delta\overline{\pi} \leq k$*.*

2. All other period lengths in $\Pi^{\mathcal{S}}$ *which are larger than* $\pi^{\mathcal{S}}_{\alpha}$ *are also larger than* $k - \overline{\pi}$*.*

Proof: The proof follows from simple properties of periods:

1. By Fact 2.2, the suffix $\mathcal{S}[\pi^{\mathcal{S}}_{\alpha} + 1..k]$ has a period length $\overline{\pi}$. Any integral multiple $\delta\overline{\pi} \leq k - \pi^{\mathcal{S}}_{\alpha}$ is also a period length of this suffix. By Fact 2.2, $\pi^{\mathcal{S}}_{\alpha} + \delta\overline{\pi}$ is a period length of $\mathcal{S}[1..k]$.

2. Let $\pi^{\mathcal{S}}_{\gamma}$ be the smallest period length in $\Pi^{\mathcal{S}}$ which is larger than $\pi^{\mathcal{S}}_{\alpha}$ and is not of the form $\pi^{\mathcal{S}}_{\alpha} + \delta\overline{\pi}$. Then $\pi^{\mathcal{S}}_{\gamma} \geq \pi^{\mathcal{S}}_{\alpha+1}$ and by Fact 2.2, the suffix $\mathcal{S}[\pi^{\mathcal{S}}_{\alpha} + 1..k]$ has period lengths $\overline{\pi}$ and $\pi^{\mathcal{S}}_{\gamma} - \pi^{\mathcal{S}}_{\alpha}$.

   If $\pi^{\mathcal{S}}_{\gamma} \leq k - \overline{\pi}$, then $\pi^{\mathcal{S}}_{\gamma} - \pi^{\mathcal{S}}_{\alpha} + \overline{\pi} \leq k - \pi^{\mathcal{S}}_{\alpha}$ and by Fact 2.3, the suffix $\mathcal{S}[\pi^{\mathcal{S}}_{\alpha} + 1..k]$ has also a period of length $\pi^{\mathcal{S}}_{\gamma} - \pi^{\mathcal{S}}_{\alpha} - \overline{\pi}$. By Fact 2.2, $\mathcal{S}[1..k]$ has a period length $\pi^{\mathcal{S}}_{\gamma} - \overline{\pi}$ in contradiction to the minimality of $\pi^{\mathcal{S}}_{\gamma}$. $\square$

A *substring* or a *factor* of a string $\mathcal{S}[1..k]$ is 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. In this paper 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 2.5 *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 appears on both sides of the factorization. That is, 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* of the factorization. See Figure 1 for an example.

Definition 2.6 *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 2.7 *(The Critical Factorization Theorem, Cesari and Vincent [6, 23]) Let* $\pi^{\mathcal{S}}_{1}$ *be the period length of a string* $\mathcal{S}[1..k]$*. Then, if we consider any* $\pi^{\mathcal{S}}_{1} - 1$ *consecutive non-trivial factorizations, at least one is a critical factorization.*

a | b a a a b a
b a   b a

(a)

a b | a a a b a
a a a b   a a a b

(b)

a b a | a a b a
a   a

(c)

Figure 1: The local periods at 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.

3 The Crochemore-Perrin Algorithm

Crochemore and Perrin [11] 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 also takes linear time and uses constant space. It is discussed is Section 7. 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.

The Crochemore-Perrin string matching algorithm tries to match the pattern aligned 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 [11]) Let* $(\mathcal{P}[1..\chi], \mathcal{P}[\chi+1..m])$ *be a critical factorization of the pattern and let* $\rho \leq \min(\chi, m-\chi)$ *be the length of a local period of this factorization. Then* ρ *is a multiple of* $\pi_1^{\mathcal{P}}$.

Theorem 3.2 *(Crochemore and Perrin [11]) 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 $\mathcal{P}[\chi+1..m]$. Initially $\sigma = 1$. The algorithm maintains the invariant that $\mathcal{T}[\sigma+\chi..\theta-1] = \mathcal{P}[\chi+1..\theta-\sigma]$. There are two conditions in which the while loop terminates:

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

– $\pi_1^{\mathcal{P}}$ is the period length of the pattern $\mathcal{P}[1..m]$.
– $(\mathcal{P}[1..\chi], \mathcal{P}[\chi+1..m])$ is a given critical factorization, such that $\chi < \pi_1^{\mathcal{P}}$.
– σ is the current text position that the pattern is aligned with.
– θ is the current text position we have to compare.

$\sigma = 1$
$\theta = 1 + \chi$
while $\sigma \leq n - m + 1$ **do**
  while $\theta < \sigma + m$ **and** $\mathcal{T}[\theta] = \mathcal{P}[\theta - \sigma + 1]$ **do**
    $\theta = \theta + 1$
  if $\theta < \sigma + m$ **then** – If there was a mismatch.
    $\theta = \theta + 1$
    $\sigma = \theta - \chi$
  else – The pattern suffix $\mathcal{P}[\chi + 1..m]$ was matched.
    – It remains to match the prefix $\mathcal{P}[1..\chi]$.
    – The original algorithm compares the symbols in the next statement
    – from right to left. However, any order can be used.
    if $\mathcal{T}[\sigma..\sigma + \chi - 1] = \mathcal{P}[1..\chi]$ **then**
      Report an occurrence of the pattern starting at text position σ.
    $\sigma = \sigma + \pi_1^{\mathcal{P}}$
    if $\sigma + \chi > \theta$ **then**
      $\theta = \sigma + \chi$
  end
end

Figure 2: The Crochemore-Perrin algorithm.

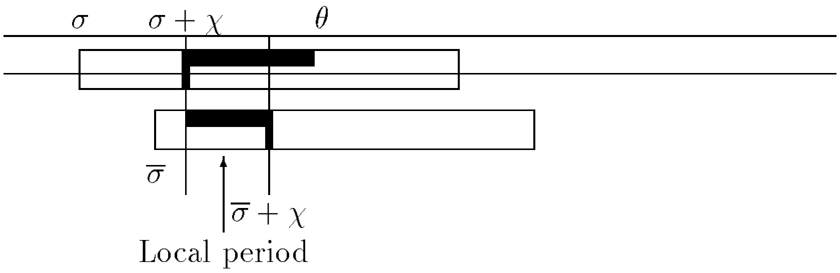

Figure 3: Applying critical factorizations. If $\mathcal{T}[\sigma + \chi..\theta - 1] = \mathcal{P}[\chi + 1..\theta - \sigma]$ and there is an occurrence of the pattern at text position $\overline{\sigma}$, $\sigma < \overline{\sigma} \leq \theta - \chi$, then the factorization $(\mathcal{P}[1..\chi],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

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.

Since $\overline{\sigma} - \sigma \leq m - \chi$, by Lemma 3.1, $\overline{\sigma} - \sigma$ is a multiple of $\pi_1^{\mathcal{P}}$. But by the definition of a period, $\mathcal{P}[\theta - \sigma + 1] = \mathcal{P}[\theta - \overline{\sigma} + 1]$ and $\mathcal{T}[\theta] \neq \mathcal{P}[\theta - \overline{\sigma} + 1]$, and there can be no occurrence of the pattern starting at text position $\overline{\sigma}$. 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. If the while 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 the pattern prefix is discovered the algorithm can report an occurrence of the pattern 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 that is shorter than $\pi_1^{\mathcal{P}}$.

   Note that if $\sigma + \chi < \theta$ after incrementing σ by $\pi_1^{\mathcal{P}}$, 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 maintained and there is no need to go back and compare parts of the pattern that were already compared.

It remains to count the number of comparisons made by the algorithm. There are at most $n - \chi$ comparisons made in the while loop since θ is incremented after each comparison and initially $\theta = \chi + 1$. The second comparison statement makes each time at most χ comparisons. 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. Therefore, 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 these comparisons. The obvious implementation of the modified 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 modified Crochemore-Perrin string matching algorithm 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.*

– $\pi_1^{\mathcal{P}}$ is the period length of the pattern $\mathcal{P}[1..m]$.
– $(\mathcal{P}[1..\chi], \mathcal{P}[\chi+1..m])$ is a given critical factorization, such that $\chi < \pi_1^{\mathcal{P}}$.
– 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 of the end of last discovered pattern suffix $\mathcal{P}[\chi+1..m]$.
– The algorithm does not compare text symbols at positions smaller than τ.

```
σ = 1
θ = 1 + χ
τ = 0
while σ ≤ n − m + 1 do
    while θ < σ + m and 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. In addition to the invariant that $\mathcal{T}[\sigma+\chi..\theta-1] = \mathcal{P}[\chi+1..\theta-\sigma]$ the modified algorithm maintains that if $\sigma < \tau$, then $\mathcal{T}[\sigma..\tau-1] = \mathcal{P}[1..\tau-\sigma]$. 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 incremented σ by $\pi_1^{\mathcal{P}}$. Let $\sigma_{\mathcal{E}}^{\phi}$ be the value of σ at the end of the phase number ϕ. Then, in the first phase $\sigma_{\mathcal{E}}^{1} \geq 1$ and in the last phase $\sigma_{\mathcal{E}}^{l} \leq n-m+1$. In any phase $\phi \geq 2$, $\sigma_{\mathcal{E}}^{\phi-1} + \pi_1^{\mathcal{P}} \leq \sigma_{\mathcal{E}}^{\phi}$. Define $\Gamma^{\phi} = \sigma_{\mathcal{E}}^{\phi} - \sigma_{\mathcal{E}}^{\phi-1}$.

Recall that after the pattern suffix $\mathcal{P}[\chi+1..m]$ was discovered in the text, the algorithm tries match the pattern prefix $\mathcal{P}[1..\chi]$. The main observation in the modified algorithm is that if this pattern prefix overlaps with a previously discovered pattern suffix, then it is not necessary to compare the overlapping parts. Namely, if we define $\tau = \sigma_{\mathcal{E}}^{\phi-1} + m$, then if $\sigma < \tau$, then $\mathcal{T}[\sigma_{\mathcal{E}}^{\phi}..\tau-1] = \mathcal{P}[1..\tau-\sigma_{\mathcal{E}}^{\phi}]$ and it suffices to compare $\mathcal{T}[\tau..\sigma_{\mathcal{E}}^{\phi}+\chi-1]$ to $\mathcal{P}[\tau-\sigma_{\mathcal{E}}^{\phi}..\chi]$.

We use a simple policy of charging comparisons to text symbols: each comparison is charged to the text symbol that is compared. However, 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 credits are transferred as follows. The comparisons that were charged during phase number ϕ to text positions between τ and $\sigma_{\mathcal{E}}^{\phi}+\chi$ are transferred χ positions back. Note that the number of these comparisons is bounded by $\Gamma^{\phi} - \pi_1^{\mathcal{P}}$ and only the $m - \pi_1^{\mathcal{P}}$ text positions that are larger than or equal to $\sigma_{\mathcal{E}}^{\phi-1} + \pi_1^{\mathcal{P}}$ might be charged with a second comparison. This charge transfer has the advantage that all text symbols at positions between $\max(\sigma_{\mathcal{E}}^{\phi}, \tau)$ and $\sigma_{\mathcal{E}}^{\phi}+\chi$ 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 ϕ by bounding the ratio between the number of these symbols to Γ^{ϕ}. If this ratio can be bounded by a constant c in all phases, then there are at most $\lfloor c\Gamma^{\phi} \rfloor$ text symbols charged with a second comparison in phase ϕ and the total number of text symbols charged with two comparisons is bounded by $\lfloor c\sum_{i=2}^{l}\Gamma^{\phi} \rfloor \leq \lfloor c(n-m) \rfloor$.

There are two cases:

1. There are at most $\Gamma^{\phi} - \pi_1^{\mathcal{P}}$ text positions charged with a second comparison in phase ϕ, but in any case no more than $m - \pi_1^{\mathcal{P}}$. The ratio $\frac{\min(\Gamma^{\phi}-\pi_1^{\mathcal{P}}, m-\pi_1^{\mathcal{P}})}{\Gamma^{\phi}}$ is maximized for $\Gamma^{\phi} = m$ 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 $\phi_{\mathcal{E}}^{\phi} = \phi_{\mathcal{E}}^{\phi-1} + \pi_1^{\mathcal{P}}$, then there are clearly no text symbols charged with a second comparison in rounds ϕ since

there were no charges transferred.

Otherwise, there was at least one mismatch while the algorithm was trying to match the pattern suffix $\mathcal{P}[\chi+1..m]$. By Lemma 2.4, $\Gamma^{\phi} > m - \pi_1^{\mathcal{P}}$. The number of text symbols charged with a second comparison in round ϕ is bounded by $\Gamma^{\phi} + \pi_1^{\mathcal{P}} - m$ 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 then $\pi_1^{\mathcal{P}}$ such symbols. The ratio $\frac{\min(\Gamma^{\phi}+\pi_1^{\mathcal{P}}-m,\pi_1^{\mathcal{P}})}{\Gamma^{\phi}}$ is maximized for $\Gamma^{\phi} = m$ 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 theorem shows that the periodicity structure of a string can be represented economically.

Theorem 5.1 *Given a string $\mathcal{S}[1..k]$, it is possible to represent all period lengths $\pi_\alpha^{\mathcal{S}} \in \Pi^{\mathcal{S}}$, such that $\pi_\alpha^{\mathcal{S}} \le k - \lfloor k/\sqrt{2}^{c} \rfloor$, by specifying only c period lengths. Furthermore, it is possible to compute this representation from the periods of $\mathcal{S}[1..k]$ and to compute the periods from this representation in linear time and constant space.*

Proof: We show how to construct the representation $\pi_{r_1}^{\mathcal{S}} \cdots \pi_{r_c}^{\mathcal{S}}$ inductively. The construction uses constant space in addition to the c memory registers that store the representation. The idea is to use Lemma 2.4 to generate larger period lengths from small ones. Periods which can be generated by smaller periods do not need to be stored.

The theorem obviously holds for $c = 0$. Initially define $\pi_{r_0}^{\mathcal{S}} = \pi_0^{\mathcal{S}} = 0$ and let $\pi_{r_1}^{\mathcal{S}} = \pi_1^{\mathcal{S}}$ and $c = 1$. Since the induction is in double steps, the base must show that the theorem holds also for $c = 1$. By Lemma 2.4, if $\pi_\alpha^{\mathcal{S}}$ is a period of $\mathcal{S}[1..k]$ which is not a multiple of $\pi_1^{\mathcal{S}}$, then $\pi_\alpha^{\mathcal{S}} \ge \pi_1^{\mathcal{S}} + 1$ and $\pi_\alpha^{\mathcal{S}} > k - \pi_1^{\mathcal{S}}$. Therefore, $\pi_\alpha^{\mathcal{S}} > \lceil \frac{k}{2} \rceil \ge k - \lfloor \frac{k}{\sqrt{2}} \rfloor$.

Assume that the remaining periods of $\mathcal{S}[1..k]$ are given in an increasing order starting with $\pi_2^{\mathcal{S}}$. Let $\pi_\alpha^{\mathcal{S}}$ be the next period length.

- If $\pi_{\alpha-1}^{\mathcal{S}} - \pi_{\alpha-2}^{\mathcal{S}} = \pi_\alpha^{\mathcal{S}} - \pi_{\alpha-1}^{\mathcal{S}}$, 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.

  Thus, all period lengths of $\mathcal{S}[1..k]$ which are smaller than or equal to $\pi_\alpha^{\mathcal{S}}$ are specified by the c periods $\pi_{r_1}^{\mathcal{S}} \cdots \pi_{r_c}^{\mathcal{S}}$.

- Otherwise, $\pi^{\mathcal{S}}_{\alpha}$ is not given by the period lengths $\pi^{\mathcal{S}}_{\alpha-2}$ and $\pi^{\mathcal{S}}_{\alpha-1}$. Note, that it might be given by previous periods, but it is more convenient not to check for this condition; e.g. the string '*aabaaabaa*' 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. 0 and k are always period lengths of a string $\mathcal{S}[1..k]$ and do not need to be specified in the representation.

  By Lemma 2.4, $\pi^{\mathcal{S}}_{\alpha} > k - (\pi^{\mathcal{S}}_{\alpha-1} - \pi^{\mathcal{S}}_{\alpha-2})$. But $\pi^{\mathcal{S}}_{\alpha} \geq \pi^{\mathcal{S}}_{\alpha-1} + 1$ and by the induction hypothesis $\pi^{\mathcal{S}}_{\alpha-2} \geq \pi^{\mathcal{S}}_{r_{c-1}} > k - \lfloor k/\sqrt{2}^{c-2} \rfloor$. Therefore,

$$2\pi^{\mathcal{S}}_{\alpha} > \pi^{\mathcal{S}}_{\alpha-1} + 1 + k - (\pi^{\mathcal{S}}_{\alpha-1} - \pi^{\mathcal{S}}_{\alpha-2}) > 2k - \lfloor k/\sqrt{2}^{c-2} \rfloor + 1$$

  and $\pi^{\mathcal{S}}_{\alpha} > k - \lfloor k/\sqrt{2}^{c} \rfloor$. Thus, all period lengths which are smaller than or equal to $k - \lfloor \frac{k}{\sqrt{2}^c} \rfloor$ are given by the c periods $\pi^{\mathcal{S}}_{r_1} \cdots \pi^{\mathcal{S}}_{r_c}$.

  If there is space to store more periods, then set $\pi^{\mathcal{S}}_{r_{c+1}} = \pi^{\mathcal{S}}_{\alpha}$, and increment c by one.

Given $\pi^{\mathcal{S}}_{r_1} \cdots \pi^{\mathcal{S}}_{r_c}$, one can obviously generate all periods $\pi^{\mathcal{S}}_{\alpha}$, such that $\pi^{\mathcal{S}}_{\alpha} \leq k - \lfloor k/\sqrt{2}^{c} \rfloor$, in an increasing order, in time that is linear in the number of periods generated and using constant space. Sometimes, it is possible to generate larger periods, but when generating periods that are larger than $\pi^{\mathcal{S}}_{r_c}$, other periods which are not specified by this representation might be skipped. □

The bounds in the last theorem are obviously not tight. It is easy to see that if $c = 1$, then all periods $\pi^{\mathcal{S}}_{\alpha} \leq \lceil k/2 \rceil$ are represented and if $c = 2$, then all periods $\pi^{\mathcal{S}}_{\alpha} \leq \lceil 2k/3 \rceil$ are represented. It is also possible to show that all periods $\pi^{\mathcal{S}}_{\alpha} \leq k - \lfloor (\frac{2}{3})^c k \rfloor$ are represented.

Corollary 5.2 *All periods of a string $\mathcal{S}[1..k]$ can be represented by $\lfloor 2 \log k \rfloor$ periods.*

Remark. The compact representation of periods of a string is not new. Galil and Seiferas [15] used similar arguments in a variant of the Knuth-Morris-Pratt string matching algorithm that uses only $O(\log m)$ space. Guibas and Odlyzko [19] characterized all possible periodicity structures of a string of length k and showed that there are $\Theta(k^{\log k})$ 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 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 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 5.2, the periods can be represented in $O(\log m)$ memory registers. By Theorem 5.1, 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.

However, τ increases each time an occurrence of the pattern suffix 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. $\square$

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 $n + \lfloor \frac{\sqrt{2}^{c-1}}{sq^{c+1}-1}(n-m) \rfloor$ comparisons.*

Proof: Since the period $\pi_1^{\mathcal{P}}$ is used in the original algorithm, the c registers are used to store other periods. Thus, by Theorem 5.1 all period lengths $\pi_\alpha^{\mathcal{P}} \leq m - \lfloor \frac{m}{\sqrt{2}^{c+1}} \rfloor$ can be represented.

Recall the proof of Theorem 4.1. If $\Gamma^\phi \leq m - \lfloor \frac{m}{\sqrt{2}^{c+1}} \rfloor$, then the algorithm can proceed as in Theorem 6.1. The problem arises when if $\sigma < \tau$ and $\Gamma^\phi > m - \lfloor \frac{m}{\sqrt{2}^{c+1}} \rfloor$. Since the algorithm cannot maintain the invariant that $\mathcal{T}[\sigma..\tau-1] = \mathcal{P}[1..\tau-\sigma]$ it will behave as the original algorithm of Section 3.

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

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

establishing the claimed bound. $\square$

7 The Pattern Preprocessing

The pattern preprocessing step of the Crochemore and Perrin algorithm takes linear time, uses constant space and make at most $5m$ 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 periods of the pattern. In fact, if the period of the pattern is longer than half of its length, then the Crochemore-Perrin pattern preprocessing algorithm 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 [17] and Crochemore and Rytter [12] show that their linear-time constant-space 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 period lengths in an increasing order as required in Theorem 5.1. The construction of the economic representation of the periods proceeds as the periods are found.

The number of comparisons made is obviously linear (the constant is not very large). □

8 Open Problems

There are several 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 this context. Two problems which are related to this work are:

1. What is the exact number of comparisons required by a constant-space string matching algorithm? Is there a space vs. comparisons tradeoff for this problem?

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

9 Acknowledgments

I am indebt to Alberto Apostolico for several discussions and comments on early versions of this paper.

References

[1] A. V. Aho. Algorithms for finding pattern 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. Technical Report CUCS-054-92, Computer Science Dept., Columbia University, 1992.

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

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

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

[12] M. Crochemore and W. Rytter. Periodic prefixes in texts. In *Proc. of Sequences '91.* 1991. To appear.

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

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

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

[16] 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.

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

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

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

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

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

[22] 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.

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

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