Efficient Comparison Based String Matching

Dany Breslauer* — CWI

Zvi Galil† — Columbia University and Tel-Aviv University

CUCS-054-92

Abstract

We study the exact number of symbol comparisons that are required to solve the string matching problem and present a family of efficient algorithms. Unlike previous string matching algorithms, the algorithms in this family do not "forget" results of comparisons, what makes their analysis much simpler.

In particular, we give a linear-time algorithm that finds all occurrences of a pattern of length m in a text of length n in $n + \lceil \frac{4\log m+2}{m}(n-m) \rceil$ comparisons[1]. The pattern preprocessing takes linear time and makes at most $2m$ comparisons. This algorithm establishes that, in general, searching for a long pattern is easier than searching for a short one.

We also show that any algorithm in the family of the algorithms presented must make at least $n + \lfloor \log m \rfloor \lfloor \frac{n-m}{m} \rfloor$ symbol comparisons, for $m = 2^k - 1$ and any integer $k \geq 1$.

1 Introduction

String matching is the problem of finding all occurrences of a short string of length m called a *pattern* in a longer string of length n called a *text*. In this paper we study the number of comparisons performed by string matching algorithms that have access to the input strings by comparing pairs of symbols to test whether they are equal or not. We are interested in the exact number of comparisons rather then in an asymptotic bound. The study of the exact number of comparisons that are required to solve a problem is not new; there is an extensive study on order statistics problems [2, 4, 12, 16, 17, 18, 20, 21, 22, 23].

Several algorithms solve the string matching problem in linear time. For a survey on string matching algorithms see Aho's paper [1]. Most of 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. The Knuth-Morris-Pratt [19] algorithm makes at most $2n$ comparisons in its text processing step. The Boyer-Moore algorithm makes about $3n$ comparisons in the worst case as proved recently by Cole [8], but performs better in practice. A variant of that algorithm which makes $2n$ comparisons was designed by Apostolico and Giancarlo [3]. The Galil-Seiferas [15] algorithm and the Crochemore-Perrin [11] algorithm work in linear time and use only a constant additional

*Partially supported by the IBM Graduate Fellowship while studying at Columbia University and by the European Research Consortium for Informatics and Mathematics postdoctoral fellowship.

†Partially supported by NSF Grants CCR-90-14605 and CISE Institutional Infrastructure Grant CDA-90-24735.

[1]All logarithms in this paper are to the base 2 unless specified otherwise.

space. These algorithms can be implemented in linear time even on a multi-head two-way finite automaton. The Crochemore-Perrin algorithm performs at most $2n$ comparisons, but uses order comparisons, which give less-than, equal-to, or greater-than answers, in its pattern preprocessing step.

This paper is primarily concerned with the complexity of the string matching problem in the deterministic comparison model. It might be impossible to match the number of comparisons performed by a comparison model algorithm with an algorithm that works in linear time on a conventional model. For this reason, the number of comparisons performed by the algorithms which are described in this paper is analyzed first and efficient implementations in a conventional model are discussed only later.

All previous string matching algorithms are oblivious in the sense that they sometimes "forget" or do not use information that was obtained in previous comparisons. In fact, Colussi [10] developed his algorithm by using formal program correctness proof techniques to avoid performing comparisons which are implied from the results of previous comparisons. The algorithms which are described in this paper do not "forget" answers to comparisons, what makes their analysis much simpler.

Galil and Giancarlo [14] distinguish between on-line and off-line string matching algorithms. An *on-line* algorithm has access to the text through a sliding window whose length is equal to the length of the pattern. This means that such an algorithm has to report if there is an occurrence at a certain text position i before examining any text position larger than or equal to $i + m$. On the other hand, an *off-line* algorithm has access to the whole text.

Colussi [10] showed that the string matching problem can be solved using $1.5n - .5(m - 1)$ comparisons improving the best previous bound of $2n - m$ comparisons [3, 5, 11, 19]. Galil and Giancarlo [14] proved that the number of comparisons performed by Colussi's algorithm is also bounded by $n + \lfloor (n-m)\frac{\min(\pi_1, m-\pi_1)}{m} \rfloor \leq n + \lfloor \frac{n-m}{2} \rfloor$, where π_1 is the length of the shortest period of the pattern (periods are defined in Section 3). They also gave an improved version of Colussi's algorithm that makes $n + \lfloor (n-m)\min(1/3, \frac{\min(\pi_1, m-\pi_1)+2}{2m}) \rfloor \leq \frac{4}{3}n - \frac{1}{3}m$ comparisons in the worst case.

Galil and Giancarlo [13] proved lower bounds for the on-line and the off-line problems. These lower bounds match their upper bound for patterns of length 1, 2 and 3. These lower bounds were later generalized and improved by Zwick and Paterson [25].

In this paper we generalize Colussi's [10] algorithm and Galil and Giancarlo's [14] algorithm and present a family of on-line algorithms with a similar behavior. One of the algorithms in this family is shown to perform at most $n + \lceil \frac{4\log m+2}{m}(n-m) \rceil$ symbol comparisons. This algorithm and its pattern preprocessing step that makes $2m$ comparisons can be implemented in linear-time on a conventional computation model. We also show that any algorithm in the family of string matching algorithms presented must make at least $n + \lfloor \log m \rfloor \lfloor \frac{n-m}{m} \rfloor$ symbol comparisons, for $m = 2^k - 1$ and any integer $k \geq 1$.

We have learned recently that Cole and Hariharan [9] independently discovered an on-line algorithm that makes only $n + \frac{8}{3(m+1)}(n-m)$ symbol comparisons. Their algorithm can be implemented in linear-time but it requires a pattern preprocessing step that takes $\Omega(m^2)$ time. Cole and Hariharan also give tighter lower bounds for the on-line and the off-line problems.

The paper is organized as follow. In Section 2 we describe the new family of string matching algorithms. Section 3 is devoted to periodicity properties of strings and their uses in this family of algorithms. Sections 4.1 – 4.2 include the details of two algorithms in this family. In Section

5 we present a lower bound for the family of algorithms discussed. Section 6 gives the details of an implementation in a conventional model. We conclude with a brief discussion of the relation to previous work in Section 7 and a list of open problem in Section 8.

2 A Family of String Matching Algorithms

In this section we outline the structure of a family of efficient on-line string matching algorithms. Two important properties of these algorithms that make them comparison efficient and easier to analyze are:

1. They do not "forget" any comparison.
2. The comparisons are accounted for during the computation.

In the next sections we describe two algorithms in this family and analyze the number of comparisons they perform. The reader should keep in mind that the following presentation is in the comparison model where only comparisons are accounted and all other computation is free. In fact, the obvious implementation takes quadratic time. An efficient implementation on a conventional model is discussed in Section 6.

Each comparison that the algorithm make will be charged either to a text position or to a special fund. The number of comparisons charged to the special fund has to be analyzed separately for each version of the algorithm, while the number of comparisons that are charged to text positions is smaller than n as we prove in this section.

The new algorithms proceed by comparing text symbols from left to right in a "forward step" leaving some "holes", which are text positions that are not known. These holes will be "filled" or compared-to later, if necessary, in a "backward step" that compares symbols from right to left.

The algorithms maintain an integer ζ that is the current text position that is considers and a set $\Phi^\zeta = \{\phi_i^\zeta | \zeta - m < \phi_1^\zeta < \phi_2^\zeta < \cdots < \phi_{q_\zeta}^\zeta \leq \zeta\}$ of all positions of the text up to ζ that can still start occurrences of the pattern and have not been reported as occurrences yet. Each member of Φ^ζ may have a single credit assigned to it or no credit at all. The following invariants will be maintained:

1. The set Φ^ζ contains exactly all indices in the text, up to ζ, that are possible occurrences of the pattern by answers to comparisons so far.
2. If we align copies of the pattern starting at all text positions in Φ^ζ and look at the columns under text positions $\phi_1^\zeta \cdots \zeta$, the symbols in all copies of the pattern at each of these columns are identical.

Note that invariant one does not imply invariant two since some comparisons are skipped in the forward steps.

The third invariant concerns credits. Initially, each text positions has a credit which can be used for a single comparison. Each credit can be assigned and later reassigned to a member of Φ^ζ.

3. The credit of a text position can be assigned or reassigned only to a larger or equal member of Φ^ζ. In addition, the members of Φ^ζ without credits are always the first ones.

   We maintain this invariant by reassigning credits only to larger members of Φ^ζ; i.e. credits in Φ^ζ can only move to the right.

The algorithm starts with $\zeta = 1$ and $\Phi^0 = \emptyset$. We describe how the set Φ is updated in the forward step after the algorithm advances from text position $\zeta - 1$ to the next text position ζ. The set Φ^ζ is initialized by adding ζ to $\Phi^{\zeta-1}$ and then it may undergo modifications. Namely, members of Φ^ζ will be deleted until the invariants are satisfied. In the description below we assume that each time some members are deleted from Φ^ζ the remaining members are renumbered and called $\phi_1^\zeta \cdots \phi_{q_\zeta}^\zeta$.

The initial value of the set Φ^ζ obviously satisfies invariant one which will be maintained by removing from the set Φ^ζ only members for which there is evidence that they can not start an occurrence of the pattern. However, invariant two might be violated. We repeat the following until invariant two is satisfied. We distinguish between two cases:

- Invariant two is satisfied. This means that if we align copies of the pattern starting at all text positions in Φ^ζ and look at the column under text position ζ, the symbols in all copies of the pattern at that column are identical.

  If $\zeta \in \Phi^\zeta$ (initially $\zeta \in \Phi^\zeta$, but it can be removed in one of the iterations), the credit of text position ζ is assigned to $\phi_\epsilon^\zeta = \zeta$ in the set Φ^ζ satisfying invariant three.

  Note that text position ζ has not been successfully compared. We call text position ζ a *hole* and record it for a later processing. See Figure 1 for an example.

- Invariant two is violated. This means that if we align copies of the pattern starting at all text positions in Φ^ζ and look at the column under text position ζ, then there are some different symbols.

  Comparisons are performed between text position ζ and the symbol that is under ζ in a copy of the pattern that is aligned starting at one of the text positions in Φ^ζ and the set Φ^ζ is modified according to the outcome of the comparisons, until invariant two is satisfied. This loop will eventually terminate since at each step some members of Φ^ζ are removed.

  If all members of Φ^ζ except ζ have credits assigned to them, we compare text position ζ with the symbol under this position in the first copy of the pattern aligned starting at ϕ_1^ζ. We call this choice of comparison *the standard choice.*

  If some members of Φ^ζ except ζ do not have credits assigned to them, the symbol we compare to text position ζ depends on the version of the algorithm and will be discussed later.

  - The comparison results in an equal answer.
    We remove from Φ^ζ all text positions that if we align a copy of the pattern starting at these positions the symbol under text position ζ is not the same as the symbol in text position ζ. Invariants one and two are satisfied and the construction of Φ^ζ is completed. The comparison is charged to the credit of text position ζ. This text position will never be compared again since we know which symbol of the pattern is there.
    Since at least one of the members of Φ^ζ was eliminated, if any of the eliminated members had a credit and if $\zeta \in \Phi^\zeta$, that credit can be reassigned to $\phi_\epsilon^\zeta = \zeta$. Note that in this case the credit is reassigned to a larger position without violating invariant three. If no credit is assigned to $\zeta \in \Phi^\zeta$, all deleted members of Φ^ζ had no credit and at least one was deleted. In any case the number of members of Φ^ζ that have no credit is not larger

than the number of members without credit that the set Φ^{ζ} had before this comparison was made.

We now may have to shift credits in Φ^{ζ} to the right to maintain invariant three.

– The comparison results in an unequal answer.

We remove from Φ^{ζ} only the text positions that if we align a copy of the pattern starting at these positions the symbol under text position ζ is the same as the symbol that we compared to text position ζ. These are the only text positions in Φ^{ζ} for which we have evidence of not starting an occurrence of the pattern.

If all members of Φ^{ζ} except ζ had a credit we compared text position ζ to the symbol under text position ζ in the copy of the pattern aligned at ϕ_1^{ζ}. Since the comparison failed, ϕ_1^{ζ} was eliminated from Φ^{ζ} and the comparison is charged to the credit ϕ_1^{ζ} had.

If some members of Φ^{ζ} except ζ did not have a credit we compared text position ζ to some symbol depending on the version of our algorithm we describe later. In this case the comparison is charged to the special fund.

The total number of comparisons charged to the special fund depends on the version of the algorithm we use and will be analyzed separately for each version.

We now must go back to check if invariant two is satisfied.

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	ζ									
The text	**a**	**b**	**a**	**a**	b	**a**	b	**a**	**b**	**a**	.	.	.					
ϕ_1^{ζ}			a	a	b	a	b	a	a	b	a	a						
ϕ_2^{ζ}								a	a	b	a	b	a	a	b	a	a	
ϕ_3^{ζ}									a	a	b	a	b	a	a	b	a	a

Figure 1: The state of the algorithm at text position $\zeta = 9$. Holes are listed in boldface. Note that there is a hole at text position 9 since symbols at this column in the copies of the pattern aligned starting at text positions in $\Phi^{\zeta} = \{3, 8, 9\}$ are all the same. This means that no comparison is made at this text position and Φ^{10} will initially inherit all the text positions in Φ^{9}. Later, a comparison must be performed at text position $\zeta + 1 = 10$ because the column under this position contains different symbols. Since all text positions in Φ^{10} have credits, the symbol at text position 10 will be compared to 'b', the symbol aligned with it in the copy of the pattern that is aligned starting at text position 3, which is the smallest in Φ^{10}. The comparison fails, $\Phi^{10} = \{9, 10\}$, and a hole is left at text position 10. If we compared to 'a' instead, the comparison would have succeeded and we would have gotten the same set Φ^{10}, but we would not have a hole at text position 10.

If the set Φ^{ζ} is not empty after its construction has been completed, we must check if $\phi_1^{\zeta} + m = \zeta + 1$ since our algorithm is on line and it cannot examine a text position larger than ζ before reporting if there is an occurrence starting at text position ϕ_1^{ζ}.

In this case we must check if there is actually an occurrence of the pattern starting at text position ϕ_1^{ζ} and remove ϕ_1^{ζ} from Φ^{ζ}. To verify that there is such an occurrence we must go back to fill the holes we left while constructing the set Φ^{ζ}. We scan the holes that are in text positions larger than or equal to ϕ_1^ζ in decreasing order and compare the symbol at the text positions of the holes to the corresponding symbol of a copy of the pattern aligned at text position ϕ_1^ζ until a mismatch is found or all holes at text positions greater than or equal to ϕ_1^ζ are exhausted. This will be referred to as a backward step.

The comparisons performed are charged to the initial credit that the text position at each hole had. By invariant three this credit could be assigned only to a member of Φ^ζ that is larger than or equal to the text position of the hole. This credit has not been used yet since credits of members of Φ are used only when ϕ_1, the smallest member of Φ is removed from Φ and the original text position the credit belonged to will never be examined later. Credits that were assigned to members of Φ that were removed at some point and were not reassigned could not have been used either. These credits may also pay for comparing holes.

After this backward step is completed, the list of holes is emptied and all members of Φ^ζ lose their credits. This is the only way that additional members without credit are introduced to the set Φ.

In the backward step in case of a mismatch all members of Φ^ζ which are smaller than or equal to the position of the mismatch are removed from Φ^ζ. This must be done to satisfy invariant one because all copies of the pattern aligned at these text positions have the same symbol in the column of the mismatch by invariant two.

If the comparisons in all the holes succeed an occurrence of the pattern can be reported at text position ϕ_1^ζ which is deleted from Φ^ζ. In any case the algorithm resumes in a forward step with the new set Φ^ζ, and all members of Φ^ζ do not have credits. Note that all holes that are larger than or equal to the new ϕ_1^ζ have been filled by successful comparisons and these symbols will not be compared to again. This justifies emptying the list of holes. The skeleton of the algorithm is summarized in Figure 2. Note that the standard choice means chosing $\delta = 1$ in Figure 2.

Theorem 2.1 *The algorithm correctly finds and reports all occurrences of the pattern in the text. All comparisons are either charged to a text position or to the special fund. At most one comparison is charged to every text position. The number of comparisons charged to the special fund depends on the version of the algorithm and will be determined later.*

Proof: If there is a occurrence of the pattern starting at a text position ω, after $\Phi^{\omega+m-1}$ has been constructed it must include $\phi_1^{\omega+m-1} = \omega$ by invariant one. At this point holes are filled. Since an actual occurrence of the pattern starts at text position ω, all comparisons must succeed and an occurrence will be reported. Conversely, if ω is reported as an occurrence, then all the symbols must have been compared and found equal.

A comparison is charged to a text position directly only when it is successful or in a backward step. If the comparison was successful the algorithm knows what the text symbol at that position is and it will not compare that symbol again. A comparison that is unsuccessful in a backward step can be also charged to the text position compared since the set Φ^ζ is shifted ahead over the position of the mismatch and that text position will never be examined again. Other comparisons that are unsuccessful are charged either to a member of Φ or to the special fund. A comparison is charged to a member of Φ only if it is the first member and it had a credit. In this case this member will be removed from Φ and since the credit originally belonged to a text position smaller than or equal to this member, we are guaranteed that it will never be used again.

Thus except for comparisons that were charged to the special fund, all other comparisons use credits that were originally assigned to different text symbols. $\square$

– The pattern is given as $\mathcal{P}[1..m]$ and the text as $\mathcal{T}[1..n]$.
$\Phi^0 = \emptyset$
$\zeta = 1$
repeat
  $\Phi^\zeta = \Phi^{\zeta-1} \cup \{\zeta\}$
  if $\phi_1^\zeta + m > n + 1$ **then**
    STOP – No need to proceed if the text is not long enough to include more occurrences.
  while there are two members $\phi_i^\zeta, \phi_j^\zeta \in \Phi^\zeta$ such that $\mathcal{P}[\zeta - \phi_i^\zeta + 1] \neq \mathcal{P}[\zeta - \phi_j^\zeta + 1]$ **do**
    If all members of Φ^ζ, except ζ, have credits assigned to them, choose $\delta = 1$.
      Otherwise choose δ according to the version of the algorithm we are using.
    if $\mathcal{P}[\zeta - \phi_\delta^\zeta + 1] = \mathcal{T}[\zeta]$ **then** – This is an actual comparison.
      Charge the last comparison to text position ζ.
      Remove all ϕ_ϵ^ζ's such that $\mathcal{P}[\zeta - \phi_\epsilon^\zeta + 1] \neq \mathcal{P}[\zeta - \phi_\delta^\zeta + 1]$ from Φ^ζ.
    else
      If ϕ_1^ζ has credit assigned to it, the last comparison is charged to its credit.
        Otherwise the comparison is charged to the special fund.
      Remove ϕ_δ^ζ and all ϕ_ϵ^ζ's such that $\mathcal{P}[\zeta - \phi_\epsilon^\zeta + 1] = \mathcal{P}[\zeta - \phi_\delta^\zeta + 1]$ from Φ^ζ.
    end
  end
  If no successful comparison was made at last iterations of the while loop,
    then mark text position ζ as a hole.
  If $\zeta \in \Phi^\zeta$, then we need to assign a credit to ζ.
    If ζ is a hole it gets the initial credit of the text position.
    Otherwise, at least one member of Φ^ζ was removed.
      If a member that was removed had a credit, ζ inherits its credit.
      If all members that were removed did not have a credit and some members of Φ^ζ
        have credits, the credit of the smallest member of Φ^ζ that has a credit is
        reassigned to ζ.
  – The number of members without credit was not increased.
  if $\phi_1^\zeta + m = \zeta + 1$ **then**
    Let h_i be the indices of all holes at text positions larger than or equal to ϕ_1^ζ.
    scan h_i **in decreasing order**
      if $\mathcal{P}[h_i - \phi_1^\zeta + 1] \neq \mathcal{T}[h_i]$ **then** – This is an actual comparison.
        Remove all $\phi_\epsilon^\zeta \leq h_i$ from Φ^ζ.
        Stop scanning holes.
          – There is no occurrence of the pattern starting at text position ϕ_1^ζ.
      end
    end
    Unmark all holes.
    If the comparisons at all holes succeeded, report an occurrence
      of the pattern starting at text position ϕ_1^ζ and remove ϕ_1^ζ from Φ^ζ.
    – All members of Φ^ζ lose their credits.
  end
  $\zeta = \zeta + 1$
end

Figure 2: The skeleton of the new algorithms.

3 Periods in Strings

Definition 3.1 *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* $\Pi^{\mathcal{S}[1..k]} = \{\pi_i^s | 0 = \pi_0^s < \pi_1^s < \cdots < \pi_p^s = k\}$ *to be the set of all periods of* $\mathcal{S}[1..k]$*.* π_1^s*, the shortest non-trivial period of* $\mathcal{S}$ *is called* the period *of* $\mathcal{S}$*.*

The following is a well known property of periods:

Lemma 3.2 *If a string* $\mathcal{S}[1..k]$ *has two periods* π_a *and* π_b*, such that* $\pi_a < \pi_b$ *and* $\pi_a + \pi_b \leq k$*, then it also has a period* $\pi_b - \pi_a$*.*

Proof: For $1 \leq i \leq \pi_a$ we have $\mathcal{S}[i] = \mathcal{S}[i+\pi_b] = \mathcal{S}[i+\pi_b-\pi_a]$ and for $\pi_a < i \leq k-(\pi_b-\pi_a)$ we have $\mathcal{S}[i] = \mathcal{S}[i-\pi_a] = \mathcal{S}[i+\pi_b-\pi_a]$. Thus $\mathcal{S}[1..k-(\pi_b-\pi_a)] = \mathcal{S}[(\pi_b-\pi_a)+1..k]$. $\Box$

There is a close relation between the set Φ^ζ defined in the previous section and the sets of periods of prefixes of the pattern as we show in the following lemma. This relation means that most information that is required at each step of the new algorithm can be precomputed in a pattern preprocessing step.

Lemma 3.3 *If the set* Φ^ζ *is not empty after its construction is completed, then it is equal to the set of all periods of the prefix of the pattern* $\mathcal{P}[1..\zeta-\phi_1^\zeta+1]$*, except* $\zeta-\phi_1^\zeta+1$*, shifted by* ϕ_1^ζ*. That is,* $\Phi^\zeta \cup \{\zeta+1\} = \{\pi+\phi_1^\zeta | \pi \in \Pi^{\mathcal{P}[1..\zeta-\phi_1^\zeta+1]}\}$*.*

Proof: Assume the set Φ^ζ is not empty. By invariant one, it contains exactly all text positions, up to ζ, that can still start occurrences of the pattern. Furthermore, any text position larger than ζ has not been examined yet. So ϕ_1^ζ can start an occurrence of the pattern. But then all text positions in the set $\{\pi+\phi_1^\zeta | \pi \in \Pi^{\mathcal{P}[1..\zeta-\phi_1^\zeta+1]}$ *and* $\pi \neq \zeta-\phi_1^\zeta+1\}$ can also start occurrences since if we aligned copies of the pattern at these text positions the overlapping parts of these copies with the first copy aligned at ϕ_1^ζ are identical up to text position ζ. This means that Φ^ζ must include all these text positions and $\{\pi+\phi_1^\zeta | \pi \in \Pi^{\mathcal{P}[1..\zeta-\phi_1^\zeta+1]}\} \subseteq \Phi^\zeta \cup \{\zeta+1\}$.

By invariant two if we aligned copies of the pattern at all text positions in Φ^ζ the symbols at each column starting from the column under text position ϕ_1^ζ to the column under text position ζ in all copies of the pattern are identical. This means that for all $\phi_\alpha^\zeta \in \Phi^\zeta$, $\phi_\alpha^\zeta - \phi_1^\zeta$ are periods of $\mathcal{P}[1..\zeta-\phi_1^\zeta+1]$. Thus, $\Phi^\zeta \cup \{\zeta+1\} \subseteq \{\pi+\phi_1^\zeta | \pi \in \Pi^{\mathcal{P}[1..\zeta-\phi_1^\zeta+1]}\}$, since $\zeta-\phi_1^\zeta+1$ is always a period length of a string of this length. $\Box$

Corollary 3.4 *If* $\phi_\alpha^\zeta \in \Phi^\zeta$ *initially and* $\phi_\alpha^\zeta < \phi < \zeta$*, then* $\phi \in \Phi^\zeta$ *initially if and only if* $\phi - \phi_\alpha^\zeta$ *is a period of* $\mathcal{P}[1..\zeta-\phi_\alpha^\zeta]$*.*

4 Efficient Algorithms

The number of members without credit in the set Φ^ζ is crucial to the analysis of our algorithms since comparisons are charged to the special fund only if a mismatch occurs when the set Φ^ζ has members without credit. Members without credit are introduced to the set Φ only in a backward step to fill holes. We refer to these backward steps as landmarks and denote by $\overline{\phi}$ the first member of Φ that the algorithm was trying to verify an occurrence of the pattern at, in the last backward step. We also define a *round* to start in a forward step and to end after a backward step is completed or when the end of the string is reached.

Our analysis will bound the number of members without credit that the set Φ can have and show that each time a comparison is charged to the special fund the number of these members decreases. In all other cases during a round this number can not increase. To bound the total number of comparisons charged to the special fund throughout the algorithm we will show that at each round, if any comparison is charged to the special fund, then the set Φ is shifted forward by at least $\lceil m/2 \rceil$ positions; i.e. ϕ_1^ζ has increased by at least $\lceil m/2 \rceil$ since the last backward step. Note that we start with the smallest member of the set Φ equal to 1 and when it becomes larger than $n - m + 1$ the algorithm terminates.

The following lemmata provide bounds on the number of members of Φ^ζ without credits after a backward step.

Lemma 4.1 *Let $\overline{\phi}$ be the smallest member in the set Φ^ζ before a backward step to fill holes and verify an occurrence of the pattern is performed and let t be the text position of the first mismatch to verify a hole that failed and $t = \overline{\phi}$ if all comparisons succeeded. Then all text positions smaller or equal to t are removed from the set Φ^ζ and the remaining members of the set Φ^ζ correspond to the periods of the pattern that are larger than or equal to $t - \overline{\phi} + 1$, except m, shifted by $\overline{\phi}$ positions.*

Proof: The proof follows immediately from Lemma 3.3 and the modifications to the set Φ^ζ in a backward step to fill holes. $\Box$

Corollary 4.2 *Immediately after a backward step is performed, the set Φ is shifted forward by at least π_1 positions. That is, $\phi_1^\zeta - \overline{\phi} \geq \pi_1$. Furthermore, the number of members of Φ^ζ without credit is smaller than or equal to $m - \pi_1$.*

Proof: Immediate from Lemma 4.1. $\Box$

By Corollary 4.2, if $m \leq 2\pi_1$, then the set Φ is shifted forward by at least $\lceil m/2 \rceil$ positions after the backward step. If $m > 2\pi_1$, even though the initial shift is small, we will show that Φ^ζ is shifted by at least $\lceil m/2 \rceil$ positions after the first subsequent charge to the special fund.

After a backward step, if the set Φ^ζ was not shifted by at least $\lceil m/2 \rceil$ positions, the two versions of our algorithm that are described later resume executing the algorithm described in Section 2 using the standard rule (choosing $\delta = 1$) until a mismatch or the next backward step. This has an important advantage as the next two lemmas show.

Lemma 4.3 *After a backward step, if $\pi_1 < \zeta - \phi_1^\zeta + 1$ and a comparison between text position ζ and pattern position $\zeta - \phi_1^\zeta + 1$ in a forward step fails, the set Φ^ζ is modified and some members including the smallest member, ϕ_1^ζ, are removed. The new member ϕ_1^ζ satisfies $\zeta - \phi_1^\zeta + 1 \leq \pi_1$. Furthermore, after the construction of Φ^ζ is complete, the number of members of Φ^ζ without credit is smaller than or equal to π_1.*

Proof: Before the comparison is performed, the members of $\Phi^\zeta - \{\zeta\}$ correspond to periods of the prefix of the pattern $\mathcal{P}[1..\zeta - \phi_1^\zeta]$, except $\zeta - \phi_1^\zeta$, shifted by ϕ_1^ζ positions by Lemma 4.1. Assume that after the comparison has failed, some $\phi_\epsilon^\zeta < \zeta - \pi_1 + 1$ becomes the smallest member of Φ^ζ. By invariant two $\mathcal{P}[\zeta - \phi_1^\zeta + 1 - \pi_1] = \mathcal{P}[\zeta - \phi_\epsilon^\zeta + 1 - \pi_1]$. Since π_1 is a period of any prefix of the pattern also $\mathcal{P}[\zeta - \phi_1^\zeta + 1] = \mathcal{P}[\zeta - \phi_\epsilon^\zeta + 1]$. But $\mathcal{P}[\zeta - \phi_1^\zeta + 1] \neq \mathcal{T}[\zeta]$ and ϕ_ϵ^ζ must also be removed from the set Φ^ζ; a contradiction that shows that all $\phi_\epsilon^\zeta < \zeta - \pi_1 + 1$ are removed from Φ^ζ.

Since $\zeta - \phi_1^\zeta + 1 \leq \pi_1$, the total number of members in Φ^ζ is at most π_1. $\Box$

Lemma 4.4 *If $\zeta - \phi_1^\zeta + 1 \leq \pi_1$, then the set Φ^ζ was shifted forward by at least $\lceil m/2 \rceil$ positions since the last backward step. That is, $\phi_1^\zeta - \overline{\phi} \geq \lceil m/2 \rceil$.*

Proof: Before the backward step to fill holes was executed we had $\overline{\phi} + m = \zeta + 1$. Since then, ζ has been increased and also $\zeta - \phi_1^\zeta + 1 \leq \pi_1$. Therefore, the smallest member of Φ^ζ was shifted forward by $\phi_1^\zeta - \overline{\phi} \geq m - \pi_1$ positions. But by Corollary 4.2 we know that the shift is at least π_1 so we must have a shift of at least $\max(\pi_1, m - \pi_1) \geq \lceil m/2 \rceil$. $\square$

To summarize:

Lemma 4.5 *If all comparisons that are performed between text position ζ and the symbol under it in the first copy of the pattern aligned starting at text position ϕ_1^ζ until we reach the next backward step succeed, then no comparison was charged to the special fund. Otherwise, after the first comparison fails we are guaranteed that the set Φ was shifted forward at least by $\lceil m/2 \rceil$ positions since the last backward step and the number of members in Φ^ζ is at most $\lfloor m/2 \rfloor$.*

Proof: Comparisons are charged to the special fund only if they fail and some members of Φ^ζ do not have credits. If all comparisons succeed, no comparison is charged to the special fund.

After the first comparison fails, by Lemma 4.3, $\zeta - \phi_1^\zeta + 1 \leq \pi_1$. By Lemma 4.4 the set Φ^ζ was shifted by at least $\lceil m/2 \rceil$ position since the last backward step.

The number of members in Φ^ζ immediately after a backward step is smaller than or equal to $m - \pi_1$ by Corollary 4.2. If Φ^ζ was shifted by more than $\lceil m/2 \rceil$ positions then it has at most $\lfloor m/2 \rfloor$ members. Otherwise, $\pi_1 \leq \lceil m/2 \rceil$ and after the first mismatch by Lemma 4.4 the number of members of Φ^ζ is smaller than $\pi_1 \leq \lfloor m/2 \rfloor$. $\square$

At the beginning of each round after a shift of Φ^ζ by $\lceil m/2 \rceil$ text positions is guaranteed (either immediately or after the first mismatch in a forward step) we resume the execution of the algorithm in Figure 2 with a choice of δ that depends on the version of the algorithm.

4.1 Binary Patterns

In this section we propose a version of our algorithm that gives slightly better bounds when the pattern symbols are chosen from a constant-size alphabet. This version is much simpler than the general alphabet version that is given in the next section and provides an easy introduction to the arguments used. We describe how this version of the algorithm works for patterns over a binary alphabet and claim that this can be generalized for any constant-size alphabet.

Recall, that the number of comparisons charged to text positions is accounted for in Section 2 and is not larger than n. We need only to bound the number of comparisons charged to the special fund.

Our first goal is to show that the set Φ is shifted forward by at least $\lceil m/2 \rceil$ positions. This can be achieved at the cost of at most one mismatch charged to the special fund by Lemma 4.5. After this goal is achieved the number of members of Φ^ζ (without credits) is guaranteed to be at most $\lfloor m/2 \rfloor$. After the shift of $\lceil m/2 \rceil$ positions is guaranteed we proceed with the algorithm described in Figure 2 in a manner that will reduce the number of members of Φ without credit by half each time there is a mismatch.

> We look at the column under text position ζ in copies of the pattern aligned starting at all text positions in Φ^ζ that do not have credit and compare text position ζ with a symbol that appears in the largest number of copies.

Lemma 4.6 *A mismatch reduces the number of members without credit in the set Φ^ζ by at least a factor of two.*

Proof: Since the pattern alphabet is binary, if a comparison fails at least half of the members without credit are removed from Φ^ζ. $\Box$

Theorem 4.7 *This algorithm makes at most $n + \lceil \frac{2\log m+2}{m}(n-m)\rceil$ comparisons.*

Proof: Since in each round, if there is a mismatch that is charged to the special fund the set Φ is shifted forward by at least $\lceil m/2 \rceil$ positions, we can have at most $\frac{n-m}{\lceil m/2\rceil}$ such rounds.

Each round starts with at most one mismatch that is charged to the special fund. After that, there can be no more than $\log m$ comparisons charged to the special fund since each time a comparison is charged to the special fund the number of members of Φ^ζ without credit is decreased by at least a factor of 2. The total number of comparisons charged to the special fund is therefore bounded by $\lceil \frac{2\log m+2}{m}(n-m)\rceil$. If we count also the comparisons charged to text positions, the bound becomes $n + \lceil \frac{2\log m+2}{m}(n-m)\rceil$. $\Box$

If the size of the pattern alphabet is a larger constant $\sigma(\mathcal{P})$ a similar argument gives a bound of $n + \lceil \frac{2\log_{\sigma(\mathcal{P})/(\sigma(\mathcal{P})-1)} m+2}{m}(n-m)\rceil$.

4.2 General Patterns

In this section we present a version of our algorithm that works for patterns over any alphabet. Each forward step consists of up to three parts. The first and the third part use the standard rule ($\delta = 1$) to guide the choice of comparison. The second part uses the special rule defined below. The first part was discussed in Section 4. It appears in all executions of the forward step except the first one and its goal is to guarantee a shift of Φ by at least $\lceil m/2\rceil$ positions. The forward step can end during each one of the parts.

The goal of the second part is to reduce the number of members of Φ^ζ without credit to less than $\log m$. We maintain a text position χ to guide the comparisons while following the special rule. The second part starts immediately after a shift by $\lceil m/2\rceil$ positions is guaranteed after a backward step and can continue until the next backward step. We start with $\chi = \phi_1^\zeta$. The only members considered by the special rule are greater than χ. Other members will be considered in the third part.

When χ becomes the largest member of Φ^ζ we proceed to the third part. Let ϕ_ϵ^ζ be the smallest member of Φ^ζ that is larger than χ. By Lemma 3.3 we have initially $\Phi^\zeta = \{\pi + \phi_1^\zeta | \pi \in \Pi^{\mathcal{P}[1..\zeta-\phi_1^\zeta]}\}$ and during the iterations some members of Φ^ζ are deleted. Any two initial members that have the same symbol in the column under text position ζ were either both deleted or are still in Φ^ζ. By Corollary 3.4, $\overline{\pi} \equiv \phi_\epsilon^\zeta - \chi$ is a (not necessarily the smallest) period of $\mathcal{P}[1..\zeta-\chi]$.

The special rule will repeat the following step until it finds two consecutive members χ and ϕ_ϵ^ζ of Φ^ζ, such that the period $\overline{\pi} = \phi_\epsilon^\zeta - \chi$ repeats twice in $\mathcal{P}[1..\zeta-\chi]$. This step is applied in the beginning of each round after the shift of Φ by $\lceil m/2\rceil$ positions is guaranteed, starting with $\chi = \phi_1^\zeta$, and after each comparison is performed by the special rule.

> If the period $\overline{\pi} = \phi_\epsilon^\zeta - \chi$ does not repeat twice in the pattern prefix $\mathcal{P}[1..\zeta-\chi]$, that is if $\phi_\epsilon^\zeta + \overline{\pi} \geq \zeta$, we set $\chi = \phi_\epsilon^\zeta$, increment ϵ by one and repeat until either χ is the largest member of Φ^ζ or until the period is repeated twice (i.e. $\phi_\epsilon^\zeta + \overline{\pi} < \zeta$).

If χ is the largest member of Φ^ζ then we proceed with the third part, otherwise we continue with the special rule.

Lemma 4.8 *The members of the set $\Phi^\zeta = \Phi^{\zeta-1} \cup \{\zeta\}$ that are larger than χ correspond to periods of the prefix of the pattern $\mathcal{P}[1..\zeta-\chi]$. Some of these periods might continue to periods of $\mathcal{P}[1..\zeta-\chi+1]$.*

Let ϕ^ζ_ϵ be the smallest member of Φ^ζ that is larger than χ and let $\overline{\pi} = \phi^\zeta_\epsilon - \chi$. The following properties are satisfied at the beginning when $\Phi^\zeta = \Phi^{\zeta-1} \cup \{\zeta\}$ and during any step of the construction of Φ^ζ:

1. *$\phi^\zeta_\epsilon + \delta\overline{\pi} \in \Phi^\zeta$ for non-negative integers values of δ such that $\phi^\zeta_\epsilon + \delta\overline{\pi} < \zeta$.*

   The symbols in the column under text position ζ in the copies of the pattern that are aligned starting at all these text positions are the same.

2. *All other members of Φ^ζ that are larger than χ are also larger than $\zeta - \overline{\pi}$.*

Proof: The proof follows from simple properties of periods.

1. Since the prefix $\mathcal{P}[1..\zeta - \chi]$ has period length $\overline{\pi}$ it has also periods of any integral multiple of $\overline{\pi}$. Some of the periods of $\mathcal{P}[1..\zeta - \chi]$ might not extend to $\mathcal{P}[1..\zeta - \chi + 1]$.

   But since $\zeta - \phi^\zeta_\epsilon < \zeta - \chi$, we have that $\mathcal{P}[\zeta - \phi^\zeta_\epsilon - \delta\overline{\pi} + 1] = \mathcal{P}[\zeta - \phi^\zeta_\epsilon + 1]$ and all the symbols in the corresponding column are the same. Since ϕ^ζ_ϵ is still in Φ^ζ ($\delta = 0$) so are the others.

2. Assume ϕ^ζ_ν is the smallest member of Φ^ζ such that $\chi < \phi^\zeta_\nu < \zeta$ and is not of the form $\phi^\zeta_\epsilon + \delta\overline{\pi}$. By Corollary 3.4, $\mathcal{P}[1..\zeta - \chi]$ has a period $\phi^\zeta_\nu - \chi$ in addition to the period $\overline{\pi} = \phi^\zeta_\epsilon - \chi$.

   If $\phi^\zeta_\nu \leq \zeta - \overline{\pi}$, then $\phi^\zeta_\nu - \chi + \overline{\pi} \leq \zeta - \chi$ and by Lemma 3.2, $\mathcal{P}[1..\zeta - \chi]$ has also a period $\phi^\zeta_\nu - \chi - \overline{\pi}$. By Corollary 3.4, initially also $\phi^\zeta_\nu - \overline{\pi}$ was in Φ^ζ.

   But $\mathcal{P}[\zeta - \chi]$ has period $\overline{\pi}$ and therefore $\mathcal{P}[\zeta - \phi^\zeta_\nu + 1] = \mathcal{P}[\zeta - \phi^\zeta_\nu - \overline{\pi} + 1]$. These are the symbols in the column under text position ζ in the copies of the pattern aligned starting at ϕ^ζ_ν and $\phi^\zeta_\nu - \overline{\pi}$. So $\phi^\zeta_\nu - \overline{\pi}$ must still be in Φ^ζ; a contradiction to the minimality of ϕ^ζ_ν.

□

The special rule performs comparisons in the following manner:

At this point it is guaranteed that the period $\overline{\pi} = \phi^\zeta_\epsilon - \chi$ of the prefix of the pattern $\mathcal{P}[1..\zeta - \chi]$ repeats twice in this prefix.

The comparison is performed between text position ζ and the symbol aligned with this text position in the copy of the pattern that is aligned starting at ϕ^ζ_ϵ. This corresponds to the choice $\delta = \epsilon$ in Figure 2.

The set Φ^ζ is modified if necessary according to the outcome of the comparison and the loop above is repeated to update χ.

- If the comparison succeeded χ will be updated only if its current value was removed from the set Φ^ζ. In this case χ will be assigned to the previous value of ϕ^ζ_ϵ that is still in Φ^ζ because the comparison succeeded. In this case the loop above has to be repeated to compute a new χ.
  Note that if χ was not removed from Φ^ζ then the period of $\mathcal{P}[1..\zeta - \chi + 1]$, $\overline{\pi}$ still repeats twice. If χ was removed, the period may or may not repeat twice.

- If the comparison failed, χ may or may not have been removed from Φ^ζ but it will be updated in any case. The new value of χ is set to the smallest surviving member of Φ^ζ that is larger than χ. If such member does not exist we proceed to the third part. The reason χ is updated is that by Lemma 4.8 all members of the form $\phi^\zeta_\epsilon + \delta\overline{\pi}$ are removed from Φ^ζ and the next member of ϕ^ζ is larger than $\zeta - \overline{\pi}$ and the period does not repeat twice. After the update the new χ is larger than $\zeta - \overline{\pi}$.

Lemma 4.9 *The number of members of Φ^ζ that are smaller than χ is less than $\log m$ at any point while the special rule is guiding the comparisons to be performed. The number of mismatches that can occur is also smaller than $\log m$.*

Proof: After each mismatch χ is updated. Since the number of members of Φ^ζ that are smaller than χ increases only when χ is updated, we have to consider only these occasions. Note that the number of members of Φ^ζ that are smaller than χ can be reduced at any time. We prove that even if none is removed the bound still holds.

We show that each time χ is updated the potential length of a period $\overline{\pi}$ of $\mathcal{P}[1..\zeta - \chi]$ that repeats twice is halved. At the beginning the potential length for such a period is $\lfloor m/2 \rfloor$.

Immediately after a mismatch has occurred, $\zeta - \chi < \overline{\pi}$ and the longest period that repeats twice that can be found must be shorter than $\frac{\overline{\pi}}{2}$. This means that at most one member of Φ could be added to the set of members that are smaller than χ at the cost of halving the length of a potential period $\overline{\pi}$ that repeats twice.

Now, when the loop is executed to find such a period that repeats twice, each time the value of χ is modified, one members of Φ becomes smaller than χ. But since the period did not repeat twice, $\zeta - \chi$ is halved and the bound on a potential period that can repeat twice is also halved.

Therefore, the total number of mismatches while working under the special rule and the number of members of Φ^ζ that are smaller than χ are bounded by $\log m$. $\square$

The third part is applied when χ is the largest member of Φ^ζ. At the beginning when we start applying the third part, the number of members of Φ^ζ without credit is smaller than $\log m$ by Lemma 4.9.

Theorem 4.10 *The algorithm makes at most $n + \lceil \frac{4\log m + 2}{m}(n-m) \rceil$ comparisons.*

Proof: As in Theorem 4.7 there are at most $\frac{n-m}{m/2}$ rounds that have comparisons charged to the special fund. There is one mismatch charged to the special fund during the first part of a forward step. By Lemma 4.9 the number of mismatches charged to the special fund during the second part is at most $\log m$. By Lemma 4.9 the number of members of Φ^ζ without credit at the beginning of the third part is at most $\log m$ and thus the number of mismatches charged to the special fund in the third part is at most $\log m$ and at most $2\log m + 1$ during a forward step. Therefore, the number of comparisons charged to the special fund is bounded by $\lceil \frac{4\log m + 2}{m}(n-m) \rceil$ and the total number of comparisons performed by the algorithm is bounded by $n + \lceil \frac{4\log m + 2}{m}(n-m) \rceil$. $\square$

5 A Lower Bound

In this section we prove a lower bound on the number of comparisons performed by any algorithm in the family of algorithms described.

Theorem 5.1 *Consider a string matching algorithm that scans the input text string from left to right and moves to the next text symbol only when all candidates for occurrences of the pattern that have not been ruled out by comparisons made so far have the same symbol aligned with the current text position. Then, any such algorithm must make at least* $n + \lfloor \log m \rfloor \lfloor \frac{n-m}{m} \rfloor$ *symbol comparisons, for* $m = 2^k - 1$ *and any integer* k*.*

Proof: Define $\mathcal{S}_{i+1} = \mathcal{S}_i \mathcal{C}_{i+1} \mathcal{S}_i$ where the $\mathcal{C}_i$'s are different alphabet symbols. Thus, $\mathcal{S}_1$ = 'a', $\mathcal{S}_2$ = 'aba' $\mathcal{S}_3$ = '$abacaba$' etc.

Given a pattern string $\mathcal{S}_k$, $k - 1 = \lfloor \log m \rfloor$, we describe a strategy for an adversary to force a string matching algorithm which satisfies the conditions of the theorem to make at least $n + \lfloor \log m \rfloor \lceil \frac{n-m}{m} \rceil$ symbol comparisons.

The adversary chooses a text string that starts with an occurrence of the pattern $\mathcal{S}_k$. After the algorithm detects this occurrence of the pattern, there are exactly $k - 1$ potential occurrence of the pattern that overlap this occurrence and one more that starts immediately after this occurrence. If a copy of the pattern is aligned with the text starting at each of these potential occurrences, then there are k different symbols aligned with the next text position. See figure 3.

a b a c a b a d a b a c a b a **?** . . .
a b a c a b a **d** a b a c a b a
a b a **c** a b a d a b a c a b a
a **b** a c a b a d a b a c a b a
a b a c a b a d a b a c a b a

Figure 3: The lower bound for the pattern $\mathcal{S}_4$ = '$abacabadabacaba$'.

The adversary can force the algorithm to make $k - 1$ unsuccessful comparisons at this text position before the algorithm can move to the next text position. Since after $k - 1$ unsuccessful comparisons one of the candidates for occurrences of the pattern survives, the adversary can fix an occurrence of the pattern starting at this candidate. This argument can be continued and the text string can be completely covered with occurrences of the pattern. This means that the algorithm will be forced to make for a text string of length n exactly n successful comparisons and at least $k - 1 = \lfloor \log m \rfloor$ unsuccessful comparisons every m positions. □

6 Implementation Details

In this section we show that the general alphabet algorithm can be implemented in the standard random access machine model with uniform cost [2] in linear time with a linear time pattern preprocessing step that makes at most $2m$ comparisons. The comparisons made in the pattern preprocessing step of the Knuth-Morris-Pratt [19] algorithm are the only comparisons required for the pattern preprocessing of our algorithm.

Lemma 6.1 *(Knuth, Morris and Pratt [19]) Given a string* $\mathcal{P}[1..m]$*, the periods* $\pi_1^{\mathcal{P}[1..\xi]}$ *of all prefixes* $\mathcal{P}[1..\xi]$*,* $1 \le \xi \le m$*, can be computed in linear time and* $2m$ *comparisons.*

The set of all periods of each pattern prefix $\mathcal{P}[1..\xi]$ can be produced from the Knuth-Morris-Pratt pattern preprocessing when needed as shown in the next lemma. There is no need to store these sets explicitly.

Lemma 6.2 *Given $\pi_1^{\mathcal{P}[1..\gamma]}$, for $1 \le \gamma \le \xi$, the set $\Pi^{\mathcal{P}[1..\xi]}$ of all periods of the pattern prefix $\mathcal{P}[1..\xi]$ can be computed in time that is linear in its size without any additional comparisons.*

Proof: It is easy to verify that $\Pi^{\mathcal{P}[1..\xi]} = \{0\} \cup \{\pi_1^{\mathcal{P}[1..\xi]} + \pi | \pi \in \Pi^{\mathcal{P}[1..\xi - \pi_1^{\mathcal{P}[1..\xi]}]}\}$. □

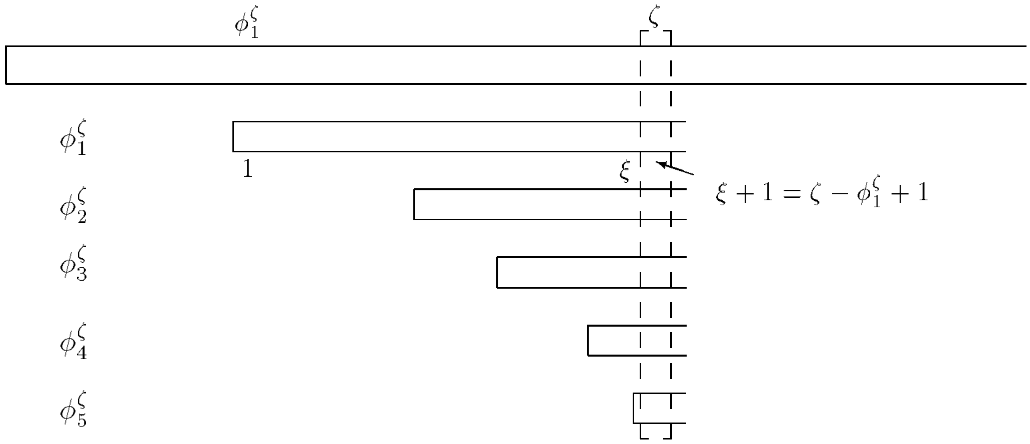

Figure 4: The algorithm at text position ζ.

Consider Figure 4. Let $\xi = \zeta - \phi_1^\zeta$. Then $\xi + 1$ is the pattern position in the column under text position ζ in a copy of the pattern that is initially aligned starting at text position ϕ_1^ζ. The preprocessing consists of the computation of three arrays in addition to the periods the prefixes of the pattern computed in Lemma 6.1.

The number of different pattern symbols initially in the column under text position ζ will be used in the implementation of the while loop in Figure 2. Define the array $\hat{\Gamma}[1..m]$, so that $\hat{\Gamma}[\xi + 1]$ is the number of different pattern symbols initially in the column under text position ζ.

Lemma 6.3 $\hat{\Gamma}[1..m]$ *can be computed in $O(m)$ time without additional comparisons.*

Proof: The array $\hat{\Gamma}[1..m]$ can be computed by the following recurrence.

$$\begin{aligned} \hat{\Gamma}[1] &= 1 \\ \hat{\Gamma}[\omega] &= \begin{cases} \hat{\Gamma}[\omega - \pi_1^{\mathcal{P}[1..\omega-1]}] & \text{if } \pi_1^{\mathcal{P}[1..\omega]} < \omega \\ \hat{\Gamma}[\omega - \pi_1^{\mathcal{P}[1..\omega-1]}] + 1 & \text{if } \pi_1^{\mathcal{P}[1..\omega]} = \omega. \end{cases} \end{aligned}$$

Obviously $\hat{\Gamma}[1] = 1$. If $\pi_1^{\mathcal{P}[1..\omega]} < \omega$, then by the definition of a period $\mathcal{P}[\omega] = \mathcal{P}[\omega - \pi_1^{\mathcal{P}[1..\omega]}]$ and $\hat{\Gamma}[\omega] = \hat{\Gamma}[\omega - \pi_1^{\mathcal{P}[1..\omega-1]}]$. Otherwise, $\mathcal{P}[\omega] \ne \mathcal{P}[\omega - \pi_\epsilon^{\mathcal{P}[1..\omega-1]}]$, for all $\pi_\epsilon^{\mathcal{P}[1..\omega-1]} \in \Pi^{\mathcal{P}[1..\omega-1]}$, $\pi_\epsilon^{\mathcal{P}[1..\omega-1]} > 0$, and $\hat{\Gamma}[\omega] = \hat{\Gamma}[\omega - \pi_1^{\mathcal{P}[1..\omega-1]}] + 1$. □

The key observation to the efficient implementation is that comparisons between pattern symbols that are in the column under the currently considered text position can be answered from the information computed in the Knuth-Morris-Pratt pattern preprocessing. By Lemma 3.3, initially $\Phi^\zeta = \{\pi + \phi_1^\zeta | \pi \in \Pi^{\mathcal{P}[1..\zeta - \phi_1^\zeta]}\}$. The algorithm in Figure 2 groups the members of Φ^ζ by the symbols

$\mathcal{P}[\xi - \pi_\alpha^{\mathcal{P}[1..\xi]} + 1]$ in the column under text position ζ. We compute the lists $\hat{\Upsilon}[1..m]$ of the smallest members of $\Pi^{\mathcal{P}[1..\xi]}$ that have a particular symbol in the column under text position ζ and generate the rest as we show in the next lemma.

Define the array $\hat{\Upsilon}[1..m]$, so that $\hat{\Upsilon}[\xi+1]$ is a set of size $\hat{\Gamma}[\xi+1]$ and for each different pattern symbol $\mathcal{P}[\xi - \pi_\beta^{\mathcal{P}[1..\xi]} + 1]$ initially in the column under text position ζ, $\xi - \pi_\alpha^{\mathcal{P}[1..\xi]} + 1 \in \hat{\Upsilon}[\xi+1]$ where $\pi_\alpha^{\mathcal{P}[1..\xi]}$ is the smallest member on $\Pi^{\mathcal{P}[1..\xi]}$ that satisfies $\mathcal{P}[\xi - \pi_\alpha^{\mathcal{P}[1..\xi]} + 1] = \mathcal{P}[\xi - \pi_\beta^{\mathcal{P}[1..\xi]} + 1]$.

Lemma 6.4 *If $\pi_\alpha^{\mathcal{P}[1..\xi]} \in \Pi^{\mathcal{P}[1..\xi]}$, then all $\pi_\beta^{\mathcal{P}[1..\xi]} \in \Pi^{\mathcal{P}[1..\xi]}$, such that $\pi_\alpha^{\mathcal{P}[1..\xi]} \leq \pi_\beta^{\mathcal{P}[1..\xi]}$ and $\mathcal{P}[\xi - \pi_\alpha^{\mathcal{P}[1..\xi]} + 1] = \mathcal{P}[\xi - \pi_\beta^{\mathcal{P}[1..\xi]} + 1]$, are given as $\{\pi_\alpha^{\mathcal{P}[1..\xi]} + \pi | \pi \in \Pi^{\mathcal{P}[1..\xi - \pi_\alpha^{\mathcal{P}[1..\xi]} + 1]}\}$. In particular, if $\pi_\alpha^{\mathcal{P}[1..\xi]} \in \hat{\Upsilon}[\xi+1]$, then all members of $\Pi^{\mathcal{P}[1..\xi]}$ that satisfy $\mathcal{P}[\xi - \pi_\alpha^{\mathcal{P}[1..\xi]} + 1] = \mathcal{P}[\xi - \pi_\beta^{\mathcal{P}[1..\xi]} + 1]$ are given by this set.*

Proof: $\pi_\beta^{\mathcal{P}[1..\xi]} - \pi_\alpha^{\mathcal{P}[1..\xi]}$ is a period of $\mathcal{P}[1..\xi - \pi_\alpha^{\mathcal{P}[1..\xi]}]$. It is a period of $\mathcal{P}[1..\zeta - \pi_\alpha^{\mathcal{P}[1..\xi]} + 1]$ if and only if $\mathcal{P}[\xi - \pi_\alpha^{\mathcal{P}[1..\xi]} + 1] = \mathcal{P}[\xi - \pi_\beta^{\mathcal{P}[1..\xi]} + 1]$. □

Lemma 6.5 *$\hat{\Upsilon}[1..m]$ can be computed in $O(m)$ time and space without additional comparisons.*

Proof: The array $\hat{\Upsilon}[1..m]$ can be computed by the following recurrence.

$$\hat{\Upsilon}[\omega] = \{\omega\} \cup \{v | v \in \hat{\Upsilon}[\omega - \pi_1^{\mathcal{P}[1..\omega-1]}] \text{ and } \mathcal{P}[\omega] \neq \mathcal{P}[v]\}$$

The condition that $\mathcal{P}[\omega] = \mathcal{P}[v]$ does not require any comparison since it is equivalent to $\omega - \pi_1^{\mathcal{P}[1..\omega]} = v$. The space required to represent the $\hat{\Upsilon}[1..m]$ array and the time it takes to compute it are $O(m)$ since $\Gamma[\omega] - 1$ of the $\Gamma[\omega]$ groups of periods of $\mathcal{P}[1..\omega-1]$ terminate at $\mathcal{P}[\omega]$. □

Define the array $\hat{\Pi}[1..m]$, so that $\hat{\Pi}[\omega]$ is the length of the shortest nonempty border of $\mathcal{P}[1..\omega]$. A *border* is a prefix which is also a suffix; a string of length ω has a border of length $\omega - \pi$ if and only if it has a period of length π. That is,

$$\hat{\Pi}[\omega] = \min\{\omega - \pi | \pi \in \Pi^{\mathcal{P}[1..\omega]} \text{ and } \pi < \omega\}.$$

The following lemma provides a way to access information that is related to each group efficiently, given any member of the group. The algorithm will use the array $\hat{\Pi}[1..m]$ to access the array $\hat{\Upsilon}[1..m]$ efficiently.

Lemma 6.6 *$\hat{\Pi}[1..m]$ can be computed in $O(m)$ time without additional comparisons. In addition, supposed that $\pi_\alpha^\xi, \pi_\beta^\xi \in \Pi^{\mathcal{P}[1..\xi]}$ and $\pi_\alpha^\xi < \pi_\beta^\xi$. Then, $\pi_\alpha^\xi, \pi_\beta^\xi$ are in the same group, that is $\mathcal{P}[\xi - \pi_\alpha^{\mathcal{P}[1..\xi]} + 1] = \mathcal{P}[\xi - \pi_\beta^{\mathcal{P}[1..\xi]} + 1]$, if and only if $\hat{\Pi}[\xi - \pi_\alpha^\xi + 1] = \hat{\Pi}[\xi - \pi_\beta^\xi + 1]$.*

Proof: The array $\hat{\Pi}[1..m]$ can be computed by the following recurrence.

$$\hat{\Pi}[\omega] = \begin{cases} \hat{\Pi}[\omega - \pi_1^{\mathcal{P}[1..\omega]}] & \text{if } \pi_1^{\mathcal{P}[1..\omega]} < \omega \\ \omega & \text{if } \pi_1^{\mathcal{P}[1..\omega]} = \omega. \end{cases}$$

If $\pi_1^{\mathcal{P}[1..\omega]} = \omega$, then the shortest nonempty border of $\mathcal{P}[1..\omega]$ is clearly $\mathcal{P}[1..\omega]$ itself. Otherwise, since any border of $\mathcal{P}[1..\omega - \pi_1^{\mathcal{P}[1..\omega]}]$ is also a border of $\mathcal{P}[1..\omega]$, the required border is the shortest border of $\mathcal{P}[1..\omega - \pi_1^{\mathcal{P}[1..\omega]}]$.

Assume that $\pi_\alpha^\xi, \pi_\beta^\xi \in \Pi^{\mathcal{P}[1..\xi]}$ and $\pi_\alpha^\xi < \pi_\beta^\xi$. If $\mathcal{P}[\xi - \pi_\alpha^{\mathcal{P}[1..\xi]} + 1] = \mathcal{P}[\xi - \pi_\beta^{\mathcal{P}[1..\xi]} + 1]$, then by the definition of a period $\pi_\beta^{\mathcal{P}[1..\xi]} - \pi_\alpha^{\mathcal{P}[1..\xi]} \in \Pi^{\mathcal{P}[1..\xi - \pi_\alpha^{\mathcal{P}[1..\xi]} + 1]}$ and $\hat{\Pi}[\xi - \pi_\alpha^\xi + 1] = \hat{\Pi}[\xi - \pi_\beta^\xi + 1]$. On the other hand, if $\hat{\Pi}[\xi - \pi_\alpha^{\mathcal{P}[1..\xi]} + 1] = \hat{\Pi}[\xi - \pi_\beta^{\mathcal{P}[1..\xi]} + 1]$, then $\mathcal{P}[1..\xi - \pi_\alpha^{\mathcal{P}[1..\xi]} + 1]$ and $\mathcal{P}[1..\xi - \pi_\beta^{\mathcal{P}[1..\xi]} + 1]$ have the same nonempty border and $\mathcal{P}[\xi - \pi_\alpha^{\mathcal{P}[1..\xi]} + 1] = \mathcal{P}[\xi - \pi_\beta^{\mathcal{P}[1..\xi]} + 1]$. $\Box$

The pattern preprocessing step can be summarized as follows.

Lemma 6.7 *The pattern preprocessing consists of computing the arrays $\hat{\Gamma}[1..m]$, $\hat{\Pi}[1..m]$, $\hat{\Upsilon}[1..m]$ and $\pi_1^{\mathcal{P}[1..\gamma]}$, for $\gamma = 1..m$. It can be done in $O(m)$ time and space using at most $2m$ comparisons.*

Theorem 6.8 *The general alphabet algorithm can be implemented in $O(n)$ time on the standard model with $O(m)$ additional space and an $O(m)$ time pattern preprocessing step that makes at most $2m$ comparisons.*

Proof: Assume that the pattern preprocessing step of Lemmas 6.7 was computed. We show that all parts of the general alphabet algorithm from Section 4.2 can be implemented in linear time and $O(m)$ space.

- **The set Φ^ζ.** We show how to maintain a somewhat redundant representation that allows to access the set Φ^ζ by a variety of operations.

  The set Φ^ζ is represented by a doubly linked list. In addition, the set Φ^ζ has an array representation that allows direct access to check if a specific member is in Φ^ζ and access its linked list representation. To avoid moving the array representation, the array will be indexed modulo m. A seperate pointer is maintained for the first member of Φ^ζ. The set Φ^ζ is initialized when the algorithm reaches text position ζ by adding ζ to the set $\Phi^{\zeta-1}$.

  Note that the algorithm can spend some constant time per each member of Φ^ζ which is removed since each text position is removed from some Φ^ζ only once during the computation. Since initially there are $\hat{\Gamma}[\xi+1]$ different pattern symbols in the column under text position ζ and $\hat{\Gamma}[\xi+1] - 1$ will be removed before the algorithm advances to the next text position, it is possible to spend $O(\hat{\Gamma}[\xi+1])$ time.

  In order to access the set $\hat{\Upsilon}[1..m]$ efficiently, the algorithm maintains an array $\Theta[1..m]$ which is initialized when the algorithm first moves to text position ζ. The initialization assigns $\Theta[\hat{\Pi}[v]] = v$, for each $v \in \hat{\Upsilon}[\xi+1]$. Note that this initialization takes $O(\hat{\Gamma}[\xi+1])$ time.

  Given ϕ_α^ζ the algorithm sometimes has to delete the group of all $\phi_\beta^\zeta \in \Phi^\zeta$ such that $\mathcal{P}[\zeta - \phi_\alpha^\zeta + 1] = \mathcal{P}[\zeta - \phi_\beta^\zeta + 1]$. By Lemma 6.6, $\hat{\Pi}[\phi_\beta^\zeta]$ is equal for every member of the same group and different for members of different groups. In particular $\hat{\Pi}[\zeta - \phi_\alpha^\zeta + 1] = \hat{\Pi}[v]$, for one member $v \in \hat{\Upsilon}[\xi+1]$, and $\Theta[\hat{\Pi}[\zeta - \phi_\alpha^\zeta + 1]] = v$. Therefore, by Lemma 6.4, the group is equal to $\{\pi_\alpha^{\mathcal{P}[1..\xi]} + \pi | \pi \in \Pi^{\mathcal{P}[1..\Theta[\hat{\Pi}[\zeta - \phi_\alpha^\zeta + 1]]}\}$. By Lemma 6.2 this group can be generated in time that is linear in its size. The elements of the group are deleted from Φ^ζ and the time to generate the group and delete its elements is charged to the deleted elements.

If a comparison to $\mathcal{T}[\zeta]$ results in unequal answer, only one group is removed. If a comparisons results in an equal answer all groups but one are removed. In the latter case, the algorithm uses the original $\hat{\Upsilon}[\xi+1]$ to access all groups of Φ^ζ which have to be removed.

- **The special rule for choosing δ.** The only part of the special rule that requires separate attention is the loop that finds a new χ such that the period $\pi_1^{P[1..\zeta-\chi]}$ is repeated twice.

  Note that by Lemma 4.5 the special rule is applied only after a mismatch and the set Φ was shifted forward by at least $\lceil m/2 \rceil$ positions. After two forward steps in which the special rule is applied the members of Φ are completely different.

  Since in each forward step that the special rule is used, χ is moved forward, any given text position in Φ is considered at most twice. Thus, the total time spent in updating χ is linear.

- **The credits.** Since the members of Φ^ζ with credits are always the last ones, it suffices to maintain an integer ϕ which is the first member of Φ^ζ that has a credit and $\phi = \infty$ if all members of Φ^ζ do not have a credit. There are two cases in which ϕ has to be updated:

  1. If ϕ was removed from Φ^ζ.
  2. If $\zeta \in \Phi^\zeta$ after the while loop has terminated, ζ does not have a credit assigned to it and none of the members with credit has been removed.

  ϕ can be updated in constant time as members of Φ^ζ are removed or after the while loop has completed, if no member with credit had been removed and $\zeta \in \Phi^\zeta$ does not have a credit.

- **The while loop.** In each iteration of the while loop, the algorithm compares $\mathcal{T}[\zeta]$ to $\mathcal{P}[\zeta - \phi_\alpha^\zeta + 1]$, for some $\phi_\alpha^\zeta \in \Phi^\zeta$. If the comparison results in an unequal answer, then some members of Φ^ζ are removed and the number of different pattern symbols in the column under text position ζ is reduces by one. The while loop terminates either when a comparison results in an equal answer or if text position ζ is left as a hole when all pattern symbols left in the column aligned with this text position are identical. The latter case happens exactly after $\hat{\Gamma}[\xi+1]-1$ iterations. Therefore, the while loop can be implemented in $\hat{\Gamma}[\xi+1]-1$ iterations with a special exit condition in case of a successful comparison.

- **The holes.** The only operations involving holes are:

  1. Marking the current text position ζ as a hole.
  2. Scanning the holes in a decreasing order. Only holes which are at text positions larger than or equal to ϕ_1^ζ need to be remembered.

  These operations can be trivially implemented if the holes are maintained as a doubly linked list. New holes are added on one side and old holes are removed as ϕ_1^ζ is updated.

Finally, we give the time analysis. Members of Φ are text positions that never rejoin Φ after they are deleted. The algorithm spends a constant time for each such member. Thus the total time is $O(n)$. $\square$

Note that by Theorem 6.8, the family of algorithms in Figure 2 except the rule for choosing δ can be implemented on the standard model within the same bounds. By using similar methods one can also implement the binary alphabet algorithm of Section 4.1.

7 Previous Work

The exposition in this paper not only generalizes the algorithms of Colussi [10] and Galil and Giancarlo [14], but can also be used to simplify the analysis of these algorithms. The differences between these algorithms and the family of algorithms which presented in this paper are described below. For a more detailed discussion see Breslauer's Thesis [6].

1. Colussi's algorithm is very similar to the family of algorithms given in Figure 2 with the standard choice $\delta = 1$ that guides the comparisons. Galil and Giancarlo's improvement of Colussi's algorithm is essentially using the choice $\delta = 2$ in special cases.

2. Colussi's algorithm associates holes with pattern positions. If a comparison fails, then the pattern template is shifted ahead and the holes are moved with it. The results of some comparisons may be "forgotten".

3. Colussi's algorithm removes from Φ^ζ only some of the first members $\phi_1^\zeta, \cdots, \phi_k^\zeta$. In many cases more members that are not numbered consecutively could be removed. Consequently, Colussi's algorithm "forgets" comparisons and some might be repeated. This phenomenon is demonstrated by the pattern '*abaa*' when the last pattern symbol is compared.

8 Open Problems

The exact complexity of string matching is not determined yet and there are several open problems left. The recent work of Cole and Hariharan [9] still leaves a small gap between the lower and the upper comparison bounds for string matching. The gap is even larger if the pattern preprocessing is accounted for in the bounds.

1. What is the exact number of comparisons required for the string matching problem if the pattern preprocessing is not accounted for? If the pattern preprocessing is accounted for? Is there a single algorithm which is optimal in both cases?

2. Galil and Giancarlo [13, 14] also consider the number of comparisons required by algorithms that only test if there is an occurrence of the pattern in the text and do not find all occurrences. What are the bounds for this problem?

3. Do algorithms that compare pairs of text symbols have an advantage over algorithms that compare only pattern symbols to text symbols? It seems easier to obtain lower bounds for algorithms that do not compare pairs of text symbols [13, 25]. Zwick [24] has recently shown that for some patterns, pairwise comparisons of text symbols can help.

Zwick and Paterson [25] call an algorithm that has access to the text through a sliding window of length $m+k$ a *k-look-ahead* algorithm (in our terminology a zero-look-ahead algorithm is *on-line* and infinite-look-ahead algorithm is *off-line*). They show that a look-ahead is useful, at least in some cases. Currently, all general comparisons efficient string matching algorithms are on-line.

4. What is the exact number of comparisons that are required for a l-look-ahead algorithm?

5. Is a finite look-ahead sufficient? Namely, is there a finite look-ahead value l that depends only on the pattern length and suffices to obtain the same comparison bounds as an off-line algorithm?

A more ambitious question would be:

6. What is the exact number of comparisons required to match a given pattern $\mathcal{P}[1..m]$.

Similar questions can be asked about other string problems. A closely related question that might help in determining the complexity of string matching including pattern preprocessing is the following.

7. What is the exact number of comparisons required to find the period of a string? All periods of a string? The periods of all prefixes of a string? In a recent work Breslauer, Colussi and Toniolo [7] give some lower bounds for the last problem.

9 Acknowledgments

We thank Raffaele Giancarlo, Laura Toniolo and Uri Zwick for 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. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA., 1974.

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

[4] S. W. Bent and J. W. John. Finding the median requires 2n comparisons. In *Proc. 17th ACM Symp. on Theory of Computing*, pages 213–216, 1985.

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

[6] D. Breslauer. *Efficient String Algorithmics.* PhD thesis, Dept. of Computer Science, Columbia University, New York, NY, 1992.

[7] D. Breslauer, L. Colussi, and L. Toniolo. Tight Comparison Bounds for the String Prefix-Matching Problem. Technical Report CUCS-049-92, Computer Science Dept., Columbia University, 1992.

[8] 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.

[9] 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.

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

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

[12] L. R. Ford and S. M. Johnson. A tournament problem. *American Mathematical Monthly*, 66:387–389, 1959.

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

[16] D. G. Kirkpatrick. Topics in the complexity of combinatorial algorithms. Technical report, Dept. of Computer Science, University of Toronto, Toronto, Canada, 1974.

[17] D. G. Kirkpatrick. A unified lower bound for selection and set partitioning problems. *J. Assoc. Comput. Mach.*, 28:150–165, 1981.

[18] S. S. Kislitsyn. On the Selection of the k^{th} Element of an Ordered Set by Pairwise Comparison. *Sibirskii Mat. Zhurnal*, 5:557–564, 1964.

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

[20] I. Pohl. A Sorting Problem and Its Complexity. *Comm. of the ACM*, 15:462–464, 1972.

[21] A. Schönhage, M. Paterson, and N. Pippenger. Finding the median. *J. Comput. System Sci.*, 13:184–199, 1976.

[22] J. Schreier. On tournament elimination systems. *Mathesis Polska*, 7:154–160, 1932.

[23] C. K. Yap. New upper bounds for selection. *Comm. of the ACM*, 19:501–508, 1979.

[24] U. Zwick. Personal communication, 1992.

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