Tight comparison bounds for the string prefix-matching problem

Dany Breslauer *

Centrum voor Wiskunde and Informatica, Postbus 4079, 1009 AB Amsterdam, The Netherlands

Livio Colussi ** and Laura Toniolo ***

Dipartimento di Matematica Pura ed Applicata, Università Degli Studi di Padova, 35131 Padova, Italy

Communicated by D. Gries
Received 14 December 1992
Revised 21 January 1993

Abstract

Breslauer, D., L. Colussi and L. Tonoilo, Tight comparison bounds for the string prefix-matching problem, Information Processing Letters 47 (1993) 51-57.

In the *string prefix-matching* problem one is interested in finding the longest prefix of a pattern string of length m that occurs starting at each position of a text string of length n. This is a natural generalization of the string matching problem where only occurrences of the whole pattern are sought. The Knuth-Morris-Pratt string matching algorithm can be easily adapted to solve the string prefix-matching problem without making additional comparisons.

In this paper we study the exact complexity of the string prefix-matching problem in the deterministic sequential comparison model. Our bounds do not account for comparisons made in a pattern preprocessing step. The following results are presented:

(1) A family of linear-time string prefix-matching algorithms that make at most $\lfloor((2m-1)/m)n\rfloor$ comparisons.

(2) A tight lower bound of $\lfloor((2m-1)/m)n\rfloor$ comparisons for any string prefix-matching algorithm that has to match the pattern 'ab^{m-1}'.

We also consider the special case when the pattern and the text strings are the same string and all comparisons are accounted. This problem, which we call the *string self-prefix* problem, is similar to the failure function that is computed in the pattern preprocessing of the Knut-Morris-Pratt string matching algorithm and used in several other comparison efficient algorithms. By using the lower bound for the string prefix-matching problem we are able to show:

(3) A lower bound of $2m-\lfloor 2\sqrt{m}\rfloor$ comparisons for the self-prefix problem.

Keywords: Algorithms; pattern matching; string matching; comparison model; exact complexity; failure function; lower bounds

1. Introduction

In the *string prefix-matching* problem one is interested in finding the longest prefix of a pattern string $\mathcal{P}[1..m]$ that starts at each position of a text string $\mathcal{T}[1..n]$. More formally, the required output of the string prefix-matching problem is an integer array $\Pi[1..n]$ $(0 \leqslant \Pi[i] \leqslant m)$ such that for each text position i, $\mathcal{T}[i..i+\Pi[i]-1]=$

Correspondence to: L. Toniolo, Dipartimento di Matematica Pura ed Applicata, Università Degli Studi di Padova, Via Belzoni 7, 35131 Padova, Italy.

* Partially supported by the European Research Consortium for Informatics and Mathematics postdoctoral fellowship.

** Partially supported by "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo" of CNR under grant number 89.00026069.

*** The work was done while the author was visiting at the Department of Computer Science in Columbia University.

$\mathscr{P}[1..\Pi[i]]$ and if $\Pi[i] < m$ and $i + \Pi[i] \leqslant n$, then $\mathscr{T}[i + \Pi[i]] \neq \mathscr{P}[\Pi[i] + 1]$.

The string prefix-matching problem is a natural generalization of the standard string matching problem where only complete occurrences of the pattern are sought. The classical linear time string matching algorithm of Knuth, Morris and Pratt [8] can be easily adapted to solve the string prefix-matching problem in the same time bounds without making additional comparisons [1]. We assume that the reader is familiar with this algorithm.

In this paper we study the exact number of comparisons performed by algorithms that have access to the input strings by pairwise symbol comparisons that test for equality. This work was motivated by recent results on the exact comparison complexity of the string matching problem [3,5–7,10]: Colussi [5] optimized the Knuth–Morris–Pratt [8] string matching algorithm, which makes $2n - m$ comparisons, using program correctness proof techniques and presented an algorithm that makes $n + \frac{1}{2}(n - m)$ comparisons. His algorithm was later improved by Galil and Giancarlo [7] and further by Breslauer and Galil [3]. Recently, Cole and Hariharan [4] discovered an algorithm that makes only $n + (c/m)(n - m)$ comparisons, but requires an expensive pattern preprocessing [2]. Cole and Hariharan [4] also improved the lower bounds given by Galil and Giancarlo [6] and Zwick and Paterson [10]. There is still a small gap between the lower and upper bounds for string matching. (Note that if the input alphabet consists of only two symbols, then the string matching problem requires at most n comparisons. Rivest [9] proved that in the worst case any string matching algorithm has to examine at least $n - m + 1$ input symbols.)

The string prefix-matching problem is obviously harder than the standard string matching problem since each text symbol must be either compared directly to the first symbol of the pattern or compared successfully to another symbol, while in the string matching problem some text symbols might not be compared at all, as shown by Boyer and Moore [2]. Interestingly, this "hardness" introduces more structure that makes the analysis of the string prefix-matching problem easier.

This paper presents matching lower and upper bounds for the string prefix matching problem. In particular we give:

(1) A family of linear-time string prefix-matching algorithms that make at most $\lfloor((2m - 1)/m)n\rfloor$ comparisons. The patterns preprocessing step of these algorithms is almost identical to that of the string matching algorithm of Knuth, Morris and Pratt [8].

This bound improves on the $2n - 1$ comparisons made by the adapted string matching algorithm of Knuth, Morris and Pratt [8].

(2) A tight lower bound of $\lfloor((2m - 1)/m)n\rfloor$ comparisons for any string prefix-matching algorithm that has to match the pattern 'ab^{m-1}'.

These results show that although the string matching and the string prefix-matching problems are closely related, their exact comparison complexities are inherently different:

- When $m \to \infty$ and $n \gg m$ the comparison complexity of the string matching problem approaches n while the comparison complexity of the string prefix-matching problem approaches $2n$.
- The lower bound proofs of the two problems require different arguments: the pattern string that we use for the lower bound is 'ab^{m-1}' while the lower bounds for the string matching problem require patterns with more complex periodicity structures [4,6,10].

Finally, we consider the special case when the text and the pattern strings are the same string and all comparisons are accounted. This problem, which we call the *string self-prefix* problem, is

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

[2] All bounds for the string matching algorithms mentioned do not account for the comparisons made in a pattern preprocessing step. The pattern preprocessing step of Cole and Hariharan's algorithm takes $O(m^2)$ time, while the other algorithms use the Knuth–Morris–Pratt pattern preprocessing step that takes linear time.

similar to the *failure function* [3] that is computed in the pattern preprocessing on the Knuth–Morris–Pratt [8] string matching algorithm using $2m - 4$ comparisons. The failure function is also used in several other string matching algorithms [3,5,7] and in the family of algorithms discussed in this paper. We prove:

(3) A lower bound of $2m - \lfloor 2\sqrt{m} \rfloor$ comparisons for the self-prefix problem.

This paper is organized as follows. Section 2 describes the family of string prefix-matching algorithms and Section 3 gives the matching lower bound. Section 4 uses this lower bound to prove a lower bound on the self-prefix problem.

2. Upper bounds

In this section we present a family of string prefix-matching algorithms that make at most $\lfloor((2m-1)/m)n\rfloor$ comparisons. The discussion below is in the comparison model where we count only comparisons and all other computation is free. We assume that the algorithms have obtained complete information about the pattern in an unaccounted pattern preprocessing step which may compare even all $\binom{m}{2}$ pairs of pattern symbols. We further assume that the algorithms do not make any comparisons that are implied by the answers to previous comparisons. These algorithms can be implemented efficiently in the standard random access machine model [1].

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

Let $\mathcal{K}^l = \{k_i^l \mid l - m < k_1^l < k_2^l < \cdots < k_{i_l}^l = l\}$ be the set of all text positions for which $\Pi[k_i^l]$ cannot be determined without examining $\mathcal{T}[l]$. That is, $\mathcal{T}[k_i^l..l-1] = \mathcal{P}[1..l-k_i^l]$ and $\mathcal{T}[l]$ must be compared to check whether $\Pi[k_i^l] = l - k_i^l$ or $\Pi[k_i^l] > l - k_i^l$. In this terminology, an on-line prefix-matching algorithm must determine whether $\mathcal{T}[l] = \mathcal{P}[l - k_i^l + 1]$, for all $k_i^l \in \mathcal{K}^l$. before examining any text position larger than l. Note that $\mathcal{K}^{l+1} \subseteq \mathcal{K}^l \cup \{l+1\}$.

Comparison efficient on-line prefix-matching algorithms are somewhat restricted with the choices of comparisons they can make. It is easy to see that they gain no advantage by comparing pairs of text symbols. Furthermore, all comparisons at text position l must be between $\mathcal{T}[l]$ and some $\mathcal{P}[l - k_i^l + 1]$ or otherwise can be answered by an adversary as unequal without giving the algorithm any useful information, provided that the alphabet is large enough. In the rest of this section we consider on-line algorithms that compare $\mathcal{T}[l]$ to $\mathcal{P}[l - k_i^l + 1]$, for some $k_i^l \in \mathcal{K}^l$. The only difference between these algorithms is the order in which the pattern symbols $\mathcal{P}[l - k_i^l + 1]$ are compared to $\mathcal{T}[l]$. These algorithms continue comparing $\mathcal{T}[l]$ until $\mathcal{T}[l] = \mathcal{P}[l - k_i^l + 1]$ for some k_i^l, or until $\mathcal{T}[l] \neq \mathcal{P}[l - k_i^l + 1]$ for all k_i^l, and only then move to the next text position. Note that by the assumption that the algorithms do not to make comparisons which are implied by answers to previous comparisons, and since the algorithms have complete information about the pattern, not all the symbols $\mathcal{P}[l - k_i^l + 1]$ have to be compared:

(1) If $\mathcal{P}[l - k_i^l + 1] = \mathcal{T}[l]$, then $\mathcal{P}[l - k_j^l + 1] = \mathcal{T}[l]$, for some $k_j^l \in \mathcal{K}^l$, if and only if $\mathcal{P}[l - k_i^l + 1] = \mathcal{P}[l - k_j^l + 1]$. In this case a comparison model algorithm "knows" which symbol is at text position l and it moves to the next text position.

(2) If $\mathcal{P}[l - k_i^l + 1] \neq \mathcal{T}[l]$, then $\mathcal{P}[l - k_j^l + 1] \neq \mathcal{T}[l]$, for all $k_j^l \in \mathcal{K}^l$, such that $\mathcal{P}[l - k_i^l + 1] = \mathcal{P}[l - k_j^l + 1]$. Ideally, a comparison model algorithm should not compare the text symbol $\mathcal{T}[l]$ to $\mathcal{P}[l - k_j^l + 1]$. However, this is not essential for the proofs in this paper as long as the algorithms do not compare some $\mathcal{P}[l - k_i^l + 1]$ more than once.

This leads to the definition of a family $\mathcal{F}$ of all on-line comparison model string prefix-matching algorithms that may compare $\mathcal{T}[l]$ only to some

[3] These are essentially different representations of the same information: one can be computed from the other in linear time without additional comparisons. Therefore, the lower bound applies also to the computations of the failure function.

$\mathscr{P}[l - k_i^l + 1]$. The data structures that are used by Breslauer and Galil [3] to implement a family of similar string matching algorithms can be used to implement all algorithms $\mathscr{A} \in \mathscr{F}$ in linear time with a pattern preprocessing step that relies on the Knuth–Morris–Pratt failure function.

Theorem 2.2. *Let $\mathscr{A} \in F$. Then, except possibly the rule which chooses the order according to which the $\mathscr{P}[l - k_i^l + 1]$'s are compared to $\mathscr{T}[l]$, $\mathscr{A}$ can be implemented in the standard model in linear time with the Knuth–Morris–Pratt linear time pattern preprocessing step that makes at most $2m - 4$ comparisons.*

The algorithms in the family $\mathscr{F}$ are comparison efficient as we show next.

Lemma 2.3. *Let $\mathscr{A} \in \mathscr{F}$. Then $\mathscr{A}$ makes at most $2n - 1$ comparisons.*

Proof. It is obvious that $\mathscr{A}$ does not need to make more than n comparisons which result in equal answers. In every comparison which results in unequal answer $\mathscr{A}$ determines that at least one prefix of the pattern which starts at some text position k_j^l terminates at text position l. Therefore, $\mathscr{A}$ does not make more than n comparisons which result in unequal answers. However, if all pattern prefixes that start at text positions in $\mathscr{K}^l$ terminate at text position l, then $\mathscr{A}$ moves to the next text position without a comparison that is answered as equal.

Consider the last text position $l = n$. It is clear that if all comparisons at this text position result in unequal answers, then $\mathscr{A}$ got at most $n - 1$ equal answers. On the other hand, if a comparison was answered as equal, then there is at least one pattern prefix which starts at some text position k_i^l and was not terminated by an inequality answer and, thus, $\mathscr{A}$ got at most $n - 1$ unequal answers. Therefore, $\mathscr{A}$ makes at most $2n - 1$ comparisons. □

The adapted Knuth–Morris–Pratt [8] prefix-matching algorithm is in the family $\mathscr{F}$. There are cases in which it would actually make $2n - 1$ comparisons; e.g. $\mathscr{P}[1..2] =$ 'ab' and $\mathscr{T}[1..n] =$ 'a^n'. Note that this algorithm compares $\mathscr{T}[l]$ to $\mathscr{P}[l - k_i^l + 1]$ in an increasing order of k_i^l. This order is the worst possible order as we show in the next theorem.

Define a family of algorithms $\hat{\mathscr{F}}$ of all $\mathscr{A} \in \mathscr{F}$ that compare $\mathscr{P}[l - k_1^l + 1]$ only last. Namely, if an algorithm $\mathscr{A} \in \hat{\mathscr{F}}$, then $\mathscr{A}$ compares $\mathscr{T}[l]$ to $\mathscr{P}[l - k_1^l + 1]$ only if an unequal answer implies that all pattern prefixes that start at text positions in $\mathscr{K}^l$ terminate at text position l. Note that if $\mathscr{P}[l - k_1^l + 1] = \mathscr{P}[l - k_i^l + 1]$, for $k_1^l \neq k_i^l$, then $\mathscr{A}$ may compare this pattern symbol at any time.

Theorem 2.4. *Let $\mathscr{A} \in \hat{\mathscr{F}}$. Then $\mathscr{A}$ makes at most $\lfloor((2m - 1)/m)n\rfloor$ comparisons.*

Proof. As in Lemma 2.3, every comparison between $\mathscr{T}[l]$ to $\mathscr{P}[l - k_i^l + 1]$ which results in an unequal answer determines that the pattern prefix which starts at text position k_i^l terminates at text position l. We charge such a comparison to text position k_i^l and charge comparisons that result in equal answers to the text position compared. Using this charging scheme it is obvious that each text position can be charged with at most two comparisons and that comparisons to $\mathscr{T}[l]$ cannot be charged to any text position that is smaller than k_1^l.

When $\mathscr{A}$ reaches text position l, the number of comparisons that are charged to the text positions $k_1^l, \ldots, l - 1$ is at most $2(l - k_1^l) - (|\mathscr{K}^l| - 1)$. This is so since each of these $l - k_1^l$ text positions has a comparison that resulted in equal answer charged to it, but at least $|\mathscr{K}^l| - 1$ of the text positions do not have a comparison that resulted in unequal answer charged to them.

We prove by induction that the number of comparisons charged to text positions smaller than k_1^l is at most $\lfloor((2m - 1)/m)(k_1^l - 1)\rfloor$. This is obviously true at the beginning when $l = 1$. The only concern is when $\mathscr{A}$ advances from l to $l + 1$ and $k_1^l < k_1^{l+1}$.

Let $d = k_1^{l+1} - k_1^l$. The number of comparisons that were charged to the text positions $k_1^l, \ldots, k_1^{l+1} - 1$ is at most $2d - 1$ since either at most d text positions were charged with comparisons that resulted in equal answers and k_1^l was not charged with an unequal answer, or k_1^l was charged with an unequal answer but then $k_1^{l+1} = l+1$ and text position l was not charged with an equal answer. But $d \leqslant m$ and by simple arithmetic,

$$\left\lfloor \frac{2m-1}{m}(k_1^l - 1) \right\rfloor + (2d-1) \leqslant \left\lfloor \frac{2m-1}{m}(k_1^{l+1} - 1) \right\rfloor.$$

When $\mathscr{A}$ reaches text position $l = n+1$, the number of comparisons satisfies [4],

$$\left\lfloor \frac{2m-1}{m}(k_1^l - 1) \right\rfloor + 2(l - k_1^l) - (|\mathscr{K}^l| - 1) \leqslant \left\lfloor \frac{2m-1}{m} n \right\rfloor. \quad \square$$

3. Lower bounds

In this section we show a lower bound on the number of comparisons required by any string prefix-matching algorithm which may have an unaccounted pattern preprocessing step. We describe an adversary that can force such an algorithm to make at least $\lfloor((2m-1)m)n\rfloor$ comparisons.

Theorem 3.1. *Any prefix-matching algorithm must make at least* $\lfloor((2m-1)m)n\rfloor$ *comparisons.*

Proof. Fix the pattern to $\mathscr{P}[1..m] =$ 'ab^{m-1}' and assume that the text alphabet has at least three symbols. We show that an adversary can answer comparisons made by any prefix-matching algorithm in a way that if the algorithm claims to have computed $\Pi[1..m]$ in less than $\lfloor((2m+1)m)n\rfloor$ comparisons, then it can be fooled.

Consider first algorithms that cannot compare pairs of text symbols. The adversary will maintain each text symbol in one of three states: *unknown*, *potential* 'a' *or* 'b', and *fixed* 'a' *or* 'b'.

Initially the adversary sets all text symbols at positions i, such that $i \equiv 1 \bmod m$, to be potential 'a's and all other text symbols to be unknown. A comparison between an unknown text symbol to 'a' or to 'b' is answered as unequal and the text symbol is set to be a potential 'b' or 'a', respectively. A potential 'a' or 'b' is revealed to the algorithm at the cost of one comparison after which it becomes fixed.

If an algorithm claims it has computed $\Pi[1..n]$ before all text symbols are fixed, the adversary has the freedom of setting one of the unknown or potential symbols to an alphabet symbol other than 'a' and 'b'. Let u be a text position that is not fixed and assume that all other text symbols become fixed. If $\mathscr{T}[u]$ is a potential 'b', then there exists v such that $u - m < v < u$ and $\mathscr{T}[v..u-1] =$ 'ab^{u-v-1}', and the adversary can alter $\Pi[v]$ by fixing $\mathscr{T}[u]$ to 'b' or 'c'. Similarly, the adversary can alter $\Pi[u]$ if $T[u]$ is unknown or a potential 'a'. Thus, any algorithm must make two comparisons at each text position except at the text positions that are set initially to be potential 'a's, where it has to make only one comparison. The total number of comparisons is at least $\lfloor((2m-1)/m)n\rfloor$.

When pairwise comparisons of text symbols are permitted, the lower bound arguments are slightly more complicated. To keep track of the comparisons the adversary maintains a graph with $n+2$ vertices that correspond to the n text symbols and the pattern symbols 'a' and 'b'. The edges of the graph correspond to comparisons and are labeled with their outcome ("equal" or "unequal").

The adversary maintains a two-level representation of the edges. This representation satisfies the following invariants:

(1) A subgraph that contains the edges that are labeled "unequal" and all vertices.

We refer to the connected components in this subgraph as *components*. The adversary will maintain the property that components are bipartite graphs.

(2) A subgraph that contains the edges that are labeled "equal" and all vertices.

We refer to the connected components in this subgraph as *super-vertices*. By transitivity, all vertices in a super-vertex correspond to equal symbols. The adversary will maintain the property

[4] Note that $|\mathscr{K}^l| = 1$ if $k_1^l = n+1$ and $|\mathscr{K}^l| \geqslant 2$ otherwise.

that vertices which are in the same super-vertex are always in the same side of a single component.

Initially, the graph has $1 + \lfloor n/m \rfloor$ edges: between the pattern symbol 'a' and the pattern symbol 'b' and between the pattern symbol 'b' and every text position i, such that $i \equiv 1 \bmod m$. These edges are labeled "unequal"; the invariants are clearly satisfied. The adversary answers comparisons as follows:

- A comparison between symbols which correspond to vertices that belong to different components is answered as unequal.
  The two components are merged into a single component which is still bipartite.
- A comparison between symbols which correspond to vertices that belong to the same component is answered as equal if and only if the two vertices are on the same side of the component.
  This may cause two super-vertices to be merged into one. Note that comparisons between vertices that belong to the same component but are on different sides and comparisons between two vertices in the same super-vertex do not contribute anything to the component or super-vertex structure and are practically answered for free.

The invariants are obviously maintained after each comparison is answered. Note that vertices which are in the same super-vertex as one of the pattern symbols correspond to fixed symbols; vertices which are in the same component as the pattern symbols correspond to potential symbols and vertices which are in other components correspond to unknown symbols.

A prefix-matching algorithm can terminate correctly only when there is one component and two super-vertices. Since every connected component with r vertices must have at least $r - 1$ edges, there are at least $n + 1$ edges labeled "unequal" and at least n edges labeled "equal" at termination. Thus, the total number of comparisons is at least

$$2n + 1 - \left(1 + \left\lfloor \frac{n}{m} \right\rfloor\right) = \left\lfloor \frac{2m-1}{m} n \right\rfloor. \qquad \Box$$

4. Lower bounds for the self-prefix problem

In this section we consider the special case where the pattern and the text strings are the same string and all comparisons are accounted. This problem is solved in the preprocessing step of the Knuth–Morris–Pratt [8] string matching algorithm in linear time and $2m - 4$ comparisons.

Theorem 4.1. *Fix a positive integer constant h. Then, any self-prefix algorithm that is given an input string of length m, such that $m \geqslant h$, must make at least $\lfloor((2h-1)/h)m\rfloor - h$ comparisons.*

Proof. The adversary fixes the first h symbols of the string to 'ab^{h-1}' and reveals them to the algorithm for $h - 1$ comparisons. Note that any self-prefix algorithm must compare these symbols eventually. By Theorem 3.1 the algorithm must make at least $\lfloor((2h-1)/h)(m-h)\rfloor$ more comparisons. But,

$$\left\lfloor \frac{2h-1}{h}(m-h) \right\rfloor + h - 1$$
$$= \left\lfloor \frac{2h-1}{h} m \right\rfloor - h. \qquad \Box$$

If the lenght of the input string is known to the adversary in advance, it can maximize the lower bound as the next corollary shows. In the on-line case, where the string is given a symbol at a time and its length not known in advance, there seems to be a tradeoff between maximizing the number of comparisons in the short term and in the long term.

Corollary 4.2. *The lower bound in Theorem 4.1 has a maximal value of $2m - \lfloor 2\sqrt{m} \rfloor$.*

Proof. It is easy to verify that the maximum is achieved for $h = \lfloor\sqrt{m}\rfloor$ and also for $h = \lfloor\sqrt{m}\rfloor$. $\Box$

Acknowledgment

We thank Matt Franklin, Raffaele Giancarlo and Moti Yung for comments on early versions of this paper.

References

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

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

[3] D. Breslauer and Z. Galil, Efficient comparison based string matching, *J. Complexity* (1993) to appear.

[4] R. Cole and R. Hariharan, Tighter bounds on the exact complexity of string matching, in: *Proc. 33rd IEEE Symp. on Foundations of Computer Science* (1992) 600–609.

[5] L. Colussi, Correctness and efficiency of string matching algorithm, *Inform. and Control* 95 (1991) 225–251.

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

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

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

[9] R.V. Rivest, On the worst case behavior of string-searching algorithms. *SIAM J. Comput.* **6** (1977) 669–674.

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