Fast String Matching by Using Probabilities:
On an Optimal Mismatch Variant of Horspool's Algorithm

Markus E. Nebel
Institut für Informatik, Johann Wolfgang Goethe-Universität Frankfurt
Robert Mayer Str. 11-15, 60325 Frankfurt, Germany
nebel@sads.informatik.uni-frankfurt.de

Abstract

The string matching problem, i.e. the task of finding all occurrences of one string as a substring of another one, is a fundamental problem in computer science. Recently, this problem received a great deal of attention due to numerous applications in computational biology. In this paper we present a modified version of Horspool's string matching algorithm using the probabilities of the different symbols for speeding up. We show that the modified algorithm has a linear average running time; a precise asymptotical representation of the running time will be proven. A comparison of the average running time of the modified algorithm with the well-known results for the original method shows that a substantial speed up for most of the symbol distributions has been achieved. Finally, we show that the distribution of the symbols can be approximated to a high precision using a random sample of sublinear size.

1 Introduction

In a lot of applications it is necessary to find one string being a substring of another one, e.g. when a word in a text needs to be changed or deleted. In computer science this task is called the *string matching problem*. Quite a number of algorithms for solving this problem are well-known (for a detailed presentation see [ChL04]). Since in computational biology a lot of objects are modeled as strings (nucleotide sequences of DNA/RNA, amino acid sequences of proteins,...), the problem has recently attracted new attention. As the size of these objects often is rather large, highly efficient string matching algorithms are needed to build software-tools that provide results within an appropriate time.

In this paper we present a modified version of Horspool's string matching algorithm [Hor80] making use of the probabilities of the symbols within the string that has to be searched for a substring. Provided that we have different probabilities for different symbols like for natural languages or within biological data[1], it is possible to decrease the number of comparisons needed to figure out that a string does not occur as a substring in a certain position. Consequently, we will obtain a faster string matching algorithm. This fact is proven by a precise average-case analysis of the method. A quite similar modification of the Boyer-Moore algorithm [Boy77] which is called *optimal mismatch algorithm* has been published by Sunday in [Sun90]. In his article Sunday presents simulations based on the man pages files of a unix system to give empirical evidence for the advantages of his modification.

The present paper is structured as follows: In section 2 we give a formal definition of the string matching problem and describe algorithmic solutions like a naive algorithm or the Horspool algorithm. In section 3 we present the modified algorithm in detail and perform an average-case analysis for its running time. Based on this analysis we compare the new method with the original

[1]For instance, the four different nucleotides in tRNA and rRNA are non-uniformly distributed [CKKS05], this also holds for many DNA sequences (see section 4). In addition, we usually observe non-uniform dinucleotide frequencies [ClB00].

```
s:=0;
while s<=n-m do
    j:=m;
    while (j>0) and (P[j]=T[s+j]) do j:=j-1;
    if j=0 then
        print("Pattern occurs at position",s+m);
    s:=s+1;
```

Figure 1: The naive string matching algorithm.

algorithm of Horspool. In Section 4 we present some simulation results, section 5 contains the concluding remarks. In the appendix we discuss the possibility of approximating the symbol distribution to a high precision using a random sample of sublinear size only.

2 The String Matching Problem

Suppose we have a given alphabet Σ, a text $T \in \Sigma^n$ and a pattern $P \in \Sigma^m$, $m, n \in \mathbb{N}$, $m \leq n$. Then the string matching problem is to find all occurrences of P in T, i.e. all $i \in \mathbb{N}$ with $(\exists u \in \Sigma^\star \wedge \exists v \in \Sigma^{n-i})(T = uPv)$. If $T = uPv$ with $v \in \Sigma^{n-i}$ holds, we say P occurs at position i, i.e. the position of an occurrence is defined relative to the location of P's rightmost character within T. We use $P[i]$ (resp. $T[i]$) to denote the i-th symbol of P (resp. T); the notation $P[i..j]$ (resp. $T[i..j]$), $i \leq j$, is used to represent the substring of P (resp. T) which starts with the i-th and ends with the j-th symbol of P (resp. T).

The most apparent algorithmic solution to the string matching problem is presented in Figure 1. This naive algorithm considers all possible positions $m, m+1, \ldots, n$ and compares the symbols of the pattern to the corresponding symbols of the text one by one, each time starting with the rightmost symbol of P. This obviously leads to a worst-case running time in $\Theta((n-m+1) \cdot m)$, e.g. for $\Sigma = \{A\}$, $T = A^n$, $P = A^m$. This example can also be used to explain the weakness of the algorithm: After we have found the occurrence of P at position m, the algorithm considers position $m+1$. At this position only the comparison of $P[m]$ and $T[m+1]$ is necessary; knowing P and knowing that P occurs at position m we can conclude that $P[1..m-1] = T[2..m]$ holds.

More sophisticated algorithms like the Knuth-Morris-Pratt (KMP) algorithm [KMP77] make use of this observation. In a preprocessing step they extract all the information on the pattern which is needed to avoid multiple comparisons of symbols, thus, yielding a linear running time for recognizing all occurrences. In the case of the KMP algorithm, the preprocessing phase is linear in the length of the pattern using an amortized analysis such that the worst-case running time of the complete algorithm is in $\mathcal{O}(m+n)$ and therefore optimal. Surprisingly, the naive algorithm is nearly as good as the KMP algorithm on the average [Sed92]. Note, that in contrast to our naive algorithm, the KMP algorithm scans the pattern from left to right.

A third class of algorithms exists being much faster than the KMP algorithm (on the average) but which has the same poor worst-case behavior as the naive algorithm. An example is the Hoorspool algorithm which makes use of the following idea to speed up the algorithm presented in Figure 1: Suppose that $T[j] = A$ holds and that $P[i] \neq A$, $1 \leq i \leq m$. Then, when checking position j for the occurrence of P, a mismatch is observed by the first comparison. In this situation it makes no sense to consider any of the positions $j+1, j+2, \ldots, j+m-1$ since in all cases $T[j] = A$ cannot agree with the corresponding symbol of the pattern (P contains no symbol A). Thus, in order to avoid those preassigned mismatches, the algorithm proceeds as follows: After finishing the comparison of P and T at any position k (either because a mismatch has been observed or because the pattern has been found) it will not consider a position for which $T[k]$ will be compared to a symbol of P which differs from $T[k]$. This behavior is implemented by changing `s:=s+1` of

```
s:=0;
while s<=n-m do
    j:=m;
    while (j>0) and (P[j]=T[s+j]) do j:=j-1;
    if j=0 then
        print("Pattern occurs at position",s+m);
    s:=s+d(T[s+m]);
```

Figure 2: The Horspool algorithm.

the naive algorithm into `s:=s+d(T[s+m])` with

$$d(x) := \min_{1 \le i \le m} \{i \mid i = m \vee P[m-i] = x\},\ x \in \Sigma.$$

The function d is called the *bad character heuristic* which is computed in a preprocessing step in time $\Theta(m)$.

Example: $P = ACACGGAC \rightarrow d(A) = 1,\ d(C) = 4,\ d(G) = 2,\ d(U) = 8.$

$$T = \cdots AAUGCUUAGACUCAGG \cdots$$
$$ACACGGAC$$
$$ACACGGAC$$
$$d(C) = 4$$

An implementation of the Horspool algorithm (without the preprocessing phase) is given in Figure 2.

3 The Modified Horspool Algorithm

In this section we describe a modified version of the Horspool algorithm and show its improved average running time.

3.1 The Algorithm

Like for Sunday's Optimal Mismatch algorithm, the idea of the modified algorithm considered here is to change the order in which the symbols of the pattern are compared to the symbols of the text such that the probability for a mismatch is maximized. Assuming that we know the distribution of the symbols in the text, we compare least probable symbols first in order to maximize this probability. To proceed this way it is necessary to perform an additional preprocessing step which will be described next.

For the modified algorithm we use an array `v:array[1..m] of integer` such that `v[k]=i` implies that $P[i]$ is compared to the corresponding symbol of the text in k-th order for any position considered. As already mentioned, we assume that the probability distribution of the symbols in the text is known; we will use the relative number of occurrences of the different symbols to represent these probabilities. For this purpose we use the array

```
w:array['a'..'Z'] of record
                        number:integer;
                        locations:pointer to block;
                    end;

type block = record
                position:integer;
```

```
    next:pointer to block;
end;
```

Here, the notation `array['a'..'Z']` is used to represent an array whose elements can be addressed using the symbols in Σ; `locations` is used to build up a list of all occurrences of the corresponding symbol in P. At the beginning of the preprocessing, all the pointers `locations` of array `w` are set to NIL (running time $\mathcal{O}(|\Sigma|)$); we will see that all the pointers of `w` are NIL again after the preprocessing such that subsequent searches don't need to re-initialize the data structure. The first step is to run through P from left to right, inserting each symbol $P[i]$ with its relative number of occurrences as its priority into a min-heap. Using well-known implementations for a min-heap (see e.g. [CLR90]) results in a running time proportional to $\log(m)$ for each symbol. Furthermore, the occurrence of $P[i]$ is registered within the list `locations` of `w[P[i]]`, setting `position:=i` in a new block which is created at the head of the list `locations`. This can be done in constant time per symbol leading to an overall running time for this phase in $\mathcal{O}(m \log(m))$.
In a second phase we use the min-heap to set the values of `v` according to our needs. For this purpose, we delete the smallest element stored in the heap, say symbol a. If a occurs l times in P, then the list `locations` of `w[a]` contains l elements which are used to compute `v[1]` to `v[l]`; we set[2] `v[i]:=w[a].locations^.position` for i from 1 to l, deleting each time the corresponding block from the head of the list. We have processed P from left to right, inserting elements at the head of the lists, which implies that `v[1]` corresponds to the rightmost occurrence of the least probable symbol, `v[2]` to the second rightmost occurrence of the least probable symbol (if it exists) and so on. In this way our algorithm will process all the occurrences of a symbol within the pattern from right to left. We continue in exactly the same way, deleting elements from the min-heap and processing their lists `locations` in order to set up array `v`. When the heap is empty, we are done with our preprocessing. Each symbol of the pattern will be deleted from one of the lists (in constant time) exactly once. Thus the resulting running time is in $\mathcal{O}(m)$. The deletion of the smallest element stored in a min-heap can be implemented in $\log(s)$ time for s the size of the heap; therefore the overall running-time of both preprocessing phases is on $\mathcal{O}(|\Sigma| + m \log(m))$.

Remark: It is possible to speed up this preprocessing by using a linear sorting algorithm like *distribution counting*. However, in order to apply this algorithm, we need to know the order of all symbols sorted with respect to their relative number of occurrences. If we work with a fixed distribution of the symbols, e.g. when processing natural languages with known probabilities for the letters, this order can be computed once and the distribution counting would improve the running time of our preprocessing. However, for realistic values of m, the advantage obtained is marginal. In cases of an unknown distribution of the symbols (see section 6), it makes no sense to sort the entire alphabet in order to slightly speed up the construction of array `v`.

Changing the fourth line of the algorithm presented in Figure 2 into

```
while (j>0) and (P[v[j]]=T[s+v[j]]) do j:=j-1;
```

results in an implementation of the modified method. We will investigate this algorithm in detail in order to see if it is faster than the original one and to quantify the influence of the symbol distribution to its running time.

3.2 An Average-Case Analysis

When designing a software program we usually have several algorithms at hand which we can use to solve those fundamental problems (like searching or sorting) that are part of the tasks our program has to perform. We may choose the algorithms according to the complexity of their coding, preferring the easiest implementation possible. However, we should use the fastest of the algorithms for the key functions of our program in order to get an efficient solution. When considering the string matching problem and the three solutions presented in this paper, it makes

[2]Here, `p^` is used to represent the variable addressed by pointer `p`.

sense to use the number of comparisons of symbols in order to evaluate the algorithm's running time. In a worst-case scenario, i.e. when considering only the input which results in the worst possible running time, all three algorithms have the same bad behavior; for $T = A^n$ and $P = A^m$ we perform $(n - m + 1) \cdot m$ comparisons since $d(A) = 1$ holds.
We will perform an average-case analysis along the lines of [MSR96] in order to see which algorithm performs best under a typical (average) input. In this way it will be possible to proof that the modified algorithm is faster than the original one for the vast majority of symbol distributions.

For the analysis we assume that we are given a fixed pattern P of length m and a random text T of length n where a random text is generated according to a multinomial distribution, i.e. at each position of T symbol $x \in \Sigma$ occurs independently of the other positions with probability π_x. Of course, this model is rather unrealistic with respect to most applications but it is typically the first one to consider for an average-case analysis of string matching algorithms because even for this simple model the analysis leads to a deeper understanding of the algorithm's benefits, disadvantages and of the algorithm itself.

3.3 The Number of Positions Considered

By using the bad character heuristic, the Horspool algorithm does not consider all positions of the text; in many cases the pattern is shifted over at least by one position due to a preassigned mismatch. Since the original and the modified algorithm use the same heuristic for relocating the pattern, both algorithms consider the same positions for a given input. As a consequence, well-known results concerning the number of positions considered by the Horspool algorithm can be applied to our modification. The corresponding results from [MSR96] will be recalled in the sequel.

Theorem 1 *Let $f_P(z) := \sum_{a\in\Sigma} \pi_a z^{d(a)}$, $\mu_P := \frac{1}{f'(1)}$ and $\sigma_P^2 := \frac{f'(1)+f''(1)-(f'(1))^2}{(f'(1))^3}$, f' resp. f'' the first resp. second derivative of f. For a fixed pattern P the original and the modified Horspool algorithm consider the same number of positions $H_n^{[P]}$ when searching for P in a random text of size n. We have*

$$\frac{H_n^{[P]} - \mu_P n}{\sqrt{n}} \xrightarrow{\mathcal{D}} \mathcal{N}(0, \sigma_P^2).$$

Moreover,

$$\mathbb{E}[H_n^{[P]}] = \mu_P n + \mathcal{O}(1), \text{ and}$$
$$\mathbb{V}ar[H_n^{[P]}] = \sigma_P^2 n + \mathcal{O}(1).$$

As usual, $\mathcal{N}(\mu, \sigma^2)$ is used to represent the normal distribution with mean μ and variance σ^2; $\xrightarrow{\mathcal{D}}$ denotes the convergence in distribution.

3.4 The Average Running-Time

Let X_j denote the random variable describing the number of comparisons made when position j of the text is considered by the modified algorithm. Then for $\mathbf{1}_j$ an indicator which is one if position j is actually considered by the algorithm[3], zero otherwise, the (average) number of comparisons $C_n^{[P]}$ made by the modified algorithm is given by

$$C_n^{[P]} = \sum_{j=m}^{n} X_j \mathbf{1}_j.$$

This is a sum of dependent random variables since only position m (the first position) is considered in all cases. If positions $j > m$ are considered depends on which positions $< j$ are considered and

[3]Since $d(x)$ may have values larger than 1, it is possible that some positions of the text are not considered by the algorithm.

which value of d is applied. Nevertheless, we can obtain precise results for the expected number of comparisons by direct manipulation of this formula. First we use $X_j = X_j \mathbf{1}_j + X_j(1-\mathbf{1}_j)$ in order to split the sum into two. Then dividing both sides of the equation by n and taking expectations yields

$$\mathbb{E}\left[\frac{1}{n}C_n^{[P]}\right] = \frac{1}{n}\sum_{j=m}^{n}\mathbb{E}[X_j] - \frac{1}{n}\sum_{j=m}^{n}\mathbb{E}[X_j(1-\mathbf{1}_j)]. \tag{1}$$

This representation has a nice interpretation: The first sum is the expected number of comparisons performed by the naive algorithm when comparisons are made in the optimized order of our modification. The second sum represents the number of comparisons which we saved by using the bad character heuristic since $(1-\mathbf{1}_j)$ is one if and only if position j is not considered by our algorithm. Lets take a look at the first sum. Since all the X_j are identically distributed for fixed P, we find $\frac{1}{n}\sum_{j=m}^{n}\mathbb{E}[X_j] = \frac{1}{n}(n-m+1)\mathbb{E}[X_m] \sim \mathbb{E}[X_m]$. Here we make the realistic assumption that n is much larger than m. But how does $\mathbb{E}[X_m]$ behave? Let $P_{[j]}$ denote the symbol in P with the j largest probability[4] (i.e. $P_{[m]}$ is the symbol which is compared first to the text) and $\pi_{[j]} := \pi_{P_{[j]}}$. Then we obviously have

$$\begin{aligned}\mathbb{E}[X_m] &= 1(1-\pi_{[m]}) + 2\pi_{[m]}(1-\pi_{[m-1]}) + \ldots \\ &\quad +(m-1)\pi_{[m]}\cdots\pi_{[3]}(1-\pi_{[2]}) + m\pi_{[m]}\cdots\pi_{[2]} \\ &= 1 + \pi_{[m]} + \pi_{[m]}\pi_{[m-1]} + \ldots + \pi_{[m]}\pi_{[m-1]}\cdots\pi_{[2]}.\end{aligned}$$

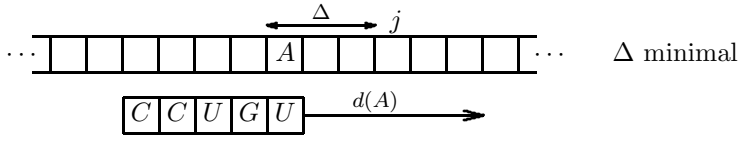
Introducing the notation

$$t_j := \begin{cases} 1 & \text{if } j = 1, \\ \pi_{[m]}\cdots\pi_{[m-j+2]} & \text{if } 2 \le j \le m, \end{cases}$$

leads to

$$\mathbb{E}[X_m] = \sum_{j=1}^{m} t_j.$$

Now, we have to consider the second sum, i.e. positions which are not considered by the algorithm since P is shifted over them by means of d. The following graphic describes the situation where position j is skipped. For minimal Δ, position $j-\Delta$ is the last position left of position j which is considered by the algorithm; the bad character heuristic implies a shift larger than Δ, thus position j is skipped:



We can use this view in order to find a representation for $(1-\mathbb{1}_j)$. First of all, in a random text position $j-\Delta$ may be any of the symbols in Σ. Thus we have to sum all the corresponding possibilities (to sum over all $x \in \Sigma$). For a fixed symbol x in the text, the bad character heuristic $d(x)$ must be larger than Δ since otherwise P would not be shifted over position j. Thus Δ may have the values 1 to $d(x)-1$. Finally, we must assure that position $j-\Delta$ is considered by the algorithms; this can be expressed using $\mathbf{1}_{j-\Delta}$. All in all this leads to

$$(1-\mathbf{1}_j) = \sum_{x\in\Sigma}\sum_{\Delta=1}^{d(x)-1}\mathbf{1}_{j-\Delta}\delta_{T[j-\Delta]=x},$$

[4]This definition of $P_{[j]}$ is only sound if all symbols of P are different. In cases where this is not fulfilled, $P_{[j]}$ is meant to be the symbol in P which is compared at $(m-j+1)$-st rank to the text.

P	$\pi_A = \frac{4}{10}$, $\pi_C = \frac{3}{10}$, $\pi_G = \frac{2}{10}$, $\pi_U = \frac{1}{10}$	$\pi_A = \frac{10}{34}$, $\pi_C = \frac{9}{34}$, $\pi_G = \frac{8}{34}$, $\pi_U = \frac{7}{34}$
$AAAAA$	$0.54955\ldots$	$0.39920\ldots$
$AAACG$	$0.52772\ldots$	$0.45682\ldots$
$ACACG$	$0.52098\ldots$	$0.45506\ldots$
$UCACG$	$0.46374\ldots$	$0.47236\ldots$
$UCCCG$	$0.32735\ldots$	$0.38235\ldots$
$UCGCG$	$0.38023\ldots$	$0.44801\ldots$
$UCCGG$	$0.36876\ldots$	$0.44471\ldots$
$UUUGG$	$0.30710\ldots$	$0.41833\ldots$
$UUUUU$	$0.24395\ldots$	$0.31380\ldots$

Table 1: Some numerical values of ρ_P.

where $\delta_{T[j-\Delta]=x}$ is used to enforce symbol x at position $j-\Delta$ of T. Inserting this into the second sum of (1) leads to

$$\sum_{j=m}^{n} \mathbb{E}[X_j(1-\mathbf{1}_j)] = \sum_{j=m}^{n} \sum_{x\in\Sigma} \pi_x \sum_{\Delta=1}^{d(x)-1} \mathsf{Pr}[\mathbf{1}_{j-\Delta}=1]\mathbb{E}[X_j \mid T[j-\Delta]=x \wedge \mathbf{1}_{j-\Delta}=1].$$

For $\Delta < d(x)$ the conditional expectation $\mathbb{E}[X_j \mid T[j-\Delta]=x \wedge \mathbf{1}_{j-\Delta}=1]$ can be written as

$$\mathbb{E}[X_j \mid T[j-\Delta]=x \wedge \mathbf{1}_{j-\Delta}=1] = \sum_{i=1}^{k(\Delta,P)} t_i,$$

where $k(\Delta,P)-1$ denotes the number of symbols of P that are compared to the text prior to symbol $P[m-\Delta]$ by the modified algorithm. This equation holds because of the preassigned mismatch which at position j exists for symbol $P[m-\Delta]$ (position j is skipped by the bad character heuristic $d(T[j-\Delta])$) such that the comparison of pattern and text is finished after $P[m-\Delta]$ has been considered. Thus it only remains to find a representation for $\mathsf{Pr}[\mathbf{1}_{j-\Delta}=1]$. Let $\varphi = \sum_{x\in\Sigma} \pi_x d(x)$ be the average bad character heuristic, i.e. assuming large texts ($n \to \infty$) the average step size used to slide P over T. Then the algorithms obviously only considered every φ's position. Consequently, (in accordance with Theorem 1) the probability for an arbitrary position to be considered is $\varphi^{-1} := \mu_P$. Collecting all the partial results presented we have proven:

Theorem 2 *The expected number of comparisons performed by the modified Horspool-algorithm for pattern P of length m and for a random text T of length n is asymptotically given by*

$$n \cdot \underbrace{\mu_P \sum_{x\in\Sigma} \pi_x \left(d(x) \sum_{j=1}^{m} t_j - \sum_{\Delta=1}^{d(x)-1} \sum_{i=1}^{k(\Delta,P)} t_i \right)}_{=:\rho_P},$$

$n \to \infty$.

Thus, we observe a linear number of comparisons where the fraction ρ_P of symbols considered depends on P and the distribution of the symbols. Table 1 shows some examples for ρ for $\Sigma = \{A,C,G,U\}$. A decreased probability for the pattern leads in most cases to a decreased value for ρ_P. However, there are exceptions from this rule like e.g. the patterns $UCCCG$ and $UCGCG$. Here the additional symbol G in the middle of P changes the bad character heuristic from $d(G)=5$ to $d(G)=2$ and the increased probability for a mismatch (symbol G is less likely than symbol C) cannot compensate the resulting decrease for φ.

Note that the representation for the expected number of comparisons given in Theorem 2 holds for any modification to the Horspool algorithm concerning only changes to the order of comparisons; only notation t_j has to be adopted to the algorithm in question. For example, setting $t_j = \pi_m \cdots \pi_{m-j+2}$ for $2 \leq j \leq m$ and $k(\Delta, P) = \Delta + 1$ leads to the expected number of comparisons made by the original algorithm.

In order to decide whether or not our modification of the Horspool algorithm leads to an improved running time, we can compare the result of Theorem 2 with corresponding results for the original algorithm [MSR96]. In doing so, we observe that both algorithms are incomparable with respect to the average number of comparisons made for searching a fixed pattern in a random text. For the different probability distributions we always find patterns for which one algorithm is faster and patterns for which the other algorithm needs fewer comparisons. We can work around this dilemma by changing our model, searching now for a random pattern of fixed size m. This means to consider $n \cdot \sum_{P \in \Sigma^m} \pi_P \rho_P$. In order to get an appropriate representation for the resulting average number of comparisons we need to get rid of the notations μ_P, $k(\Delta, P)$, $d(x)$ and t_j within the representation of ρ_P given in Theorem 2. In general this is not an easy task but for $\Sigma = \{0, 1\}$ we can proceed in the following way. Assuming without loss of generality that $\pi_1 > \pi_0$ holds, we can conclude that the modified algorithm will first compare all the zeros of the pattern with the text before it processes the first one. Therefore, partitioning the set of all patterns in Σ^m into those consisting of k zeros, we can conclude that

$$\begin{aligned} \sum_{j=1}^{m} t_j &= 1 + \sum_{j=2}^{k+1} \pi_0^{j-1} + \pi_0^k \sum_{j=k+1}^{m} \pi_1^{j-(k+1)} \\ &= \frac{\pi_0^{k+1} - 1}{\pi_0 - 1} + \pi_0^k \frac{\pi_1^{m-k} - \pi_1}{\pi_1 - 1} \end{aligned}$$

holds (this result has been used for the lengthy computation mentioned below). To continue it is advantageous to partition the set of all patterns according to their $d(0)$ and $d(1)$ values. Let $P(\alpha, \beta)$ denote the set of all $P \in \Sigma^m$ with $d(0) = \alpha$ and $d(1) = \beta$. Then obviously we have

$$n \cdot \sum_{P \in \Sigma^m} \pi_P \rho_P = n \cdot \left(\sum_{\alpha=2}^{m} \sum_{P \in P(\alpha,1)} \pi_P \rho_P + \sum_{\beta=2}^{m} \sum_{P \in P(1,\beta)} \pi_P \rho_P \right).$$

For $P \in P(\alpha, 1)$ (resp. $P \in P(1, \beta)$) we have $\mu_P^{-1} = \pi_0 \alpha + \pi_1$ (resp. $\mu_P^{-1} = \pi_0 + \pi_1 \beta$). Furthermore, a pattern with $\alpha = 1$ and $\beta = k$ (resp. $\alpha = k$ and $\beta = 1$) will be of the shape $\{0,1\}^\star 1 \underbrace{0 \cdots 0}_{k-1} \{0,1\}$ (resp. $\{0,1\}^\star 0 \underbrace{1 \cdots 1}_{k-1} \{0,1\}$). Thus we can conclude that

$$k(\Delta, P) = \begin{cases} \Delta & P \in P(1,\beta) \text{ and } P_m = 1,\ \Delta \in \{1, 2, \ldots, \beta - 1\}, \\ \Delta + 1 & P \in (1,\beta) \text{ and } P_m = 0,\ \Delta \in \{1, 2, \ldots, \beta - 1\}, \\ \#0 + \Delta & P \in P(\alpha,1) \text{ and } P_m = 0,\ \Delta \in \{1, 2, \ldots, \alpha - 1\}, \\ \#0 + \Delta + 1 & P \in P(\alpha,1) \text{ and } P_m = 1,\ \Delta \in \{1, 2, \ldots, \alpha - 1\}, \end{cases}$$

holds, where $\#0$ is used to denote the number of zeros in P. These formulæ can be used to get rid of all the special notations in our representation of ρ_P. Performing a lengthy but straight forward computation yields the following representation for the expected number of comparisons performed by the modified algorithms for a random text of size n and a random pattern of size m ($p := \pi_0$):

$$\begin{aligned} n \cdot \Bigg[& \sum_{\alpha=2}^{m-1} (1-p)^{\alpha-1} \sum_{j=0}^{m-\alpha-1} \binom{m-\alpha-1}{j} p^j (1-p)^{m-\alpha-1-j} f_1(\alpha, j, m, p) \\ & + \sum_{\beta=2}^{m-1} p^{\beta-1} \sum_{j=0}^{m-\beta-1} \binom{m-\beta-1}{j} p^j (1-p)^{m-\beta-1-j} f_2(\beta, j, m, p) \end{aligned} \tag{2}$$

$$+\frac{1}{p(p-1)^2}\left(\frac{2(1-p)^{m+1}((1-p)^3+p)}{1+(m-1)p}-\frac{(1-p)^{2m}(2+(m-2)p)(1+2(p-1)p)}{1+(m-1)p}\right.$$
$$\left.-\frac{p^m(1-p^2(2+(p-2)p)+(p-1)p^m(p^2-mp+m+1))}{m(p-1)-p}\right)\Bigg]$$

with

$$f_1(\alpha,j,m,p) = \frac{1}{1+(\alpha-1)p}\left((1-p)^{-j-2}p((1-p)^{j+1}+p^j((1-p)^m(p-1-\alpha p)(1+2p(p-1))\right.$$
$$\left.-(1-p)^{j+1}(1+(1-p)(-3+(1-p)^\alpha+2(p-1)p(p+(1-p)^\alpha-2)))))\right)$$

and

$$f_2(\beta,j,m,p) = (1-p)^{-\beta-j}p^{\beta+j-2}(-(1-p)^{\beta+j}(2p-1)(1+(p-1)p)-(1-p)^{m+1}$$
$$\times(1+2(p-1)p))+\frac{(1+(p-1)p)(p^\beta-p)-1}{\beta(p-1)-p}.$$

For any given value for m, this representation simplifies to an easy rational functions, e.g. to

$$n\cdot\frac{p^5-6\,p^4+6\,p^3-7\,p^2+6\,p-4}{(1+p)\,(p-2)},\ \text{for } m=2 \text{ and}$$

$$n\cdot\frac{14\,p^9-71\,p^8+99\,p^7+2\,p^6-106\,p^5+111\,p^4-49\,p^3+24\,p^2-24\,p+18}{(1+2\,p)\,(1+p)\,(-3+2\,p)\,(p-2)}\ \text{for } m=3.$$

However, neither the representation as a twofold sum nor the rational functions for given values of m provide a clear view on how the running-time of the algorithm depends on m and p. Therefore, we compute an appropriate asymptotical representation.

Theorem 3 *For $\Sigma=\{0,1\}$ and $p:=\pi_0<\frac{1}{2}$ the expected number of comparisons performed by the modified Horspool-algorithm for a random pattern P of fixed length m and for a random text T of length n is asymptotically given by*

$$n\cdot\begin{cases}2 & \text{if } m=1\\ 2-\frac{11}{2}p+\frac{45}{4}p^2+\mathcal{O}(p^3) & \text{if } m=2\\ m+\left(\frac{1}{2}+m-2m^2\right)p+\left(\frac{13}{12}-\frac{7}{3}m-m^2+\frac{7}{3}m^3\right)p^2+\mathcal{O}(p^3) & \text{if } m\geq 3\end{cases},$$

$n\to\infty$, $p\to 0$.

Proof: We first consider the case $m\geq 3$. In order to prove the corresponding asymptotic, we analyze the sum

$$\sum_{\alpha=2}^{m-1}(1-p)^{\alpha-1}\sum_{j=0}^{m-\alpha-1}\binom{m-\alpha-1}{j}p^j(1-p)^{m-\alpha-1-j}f_1(\alpha,j,m,p)$$
$$+\sum_{\beta=2}^{m-1}p^{\beta-1}\sum_{j=0}^{m-\beta-1}\binom{m-\beta-1}{j}p^j(1-p)^{m-\beta-1-j}f_2(\beta,j,m,p)$$
$$=\sum_{\alpha=2}^{m-1}\sum_{j=0}^{m-\alpha-1}\binom{m-\alpha-1}{j}\underbrace{p^j(1-p)^{m-\alpha-j-1}\left[(1-p)^{\alpha-1}f_1(\alpha,j,m,p)+p^{\alpha-1}f_2(\alpha,j,m,p)\right]}_{=:\tau}.$$

Expanding τ we observe terms of three different patterns:

1. Terms of the pattern $c \cdot p^\gamma (1-p)^\delta$ with the following values for the parameters c, γ and δ:

   $c = -2$, $\gamma = 2j + 2\alpha$, $\delta = m - \alpha - j - 1$; $c = -3$, $\gamma = 2j + 2\alpha - 2$, $\delta = m - \alpha - j - 1$;
   $c = 3$, $\gamma = 2j + 2\alpha - 1$, $\delta = m - \alpha - j - 1$; $c = 1$, $\gamma = 2j + 2\alpha - 3$, $\delta = m - \alpha - j - 1$;
   $c = -1$, $\gamma = 2j + 2\alpha - 3$, $\delta = 2m - 2\alpha - 2j - 1$; $c = -4$, $\gamma = 2j + 2\alpha - 1$, $\delta = 2m - 2\alpha - 2j - 1$;
   $c = 3$, $\gamma = 2j + 2\alpha - 2$, $\delta = 2m - 2\alpha - 2j - 1$; $c = 2$, $\gamma = 2j + 2\alpha$, $\delta = 2m - 2\alpha - 2j - 1$;

   We number these settings left to right from top to button. Obviously, $c \cdot p^\gamma (1-p)^\delta = c \cdot \sum_{k \geq 0} \binom{\delta}{k} (-1)^k p^{k+\delta}$ holds. Since $j \geq 0$ and $\alpha \geq 2$ hold according to our summation, γ is at least 1 in all cases such that there is no contribution to the constant term in the expansion of $c \cdot p^\gamma (1-p)^\delta$ around $p = 0$ for any of the eight parameter settings. For the coefficient at p there is only the choice $\gamma = 1$ and $k = 0$ which is fulfilled by the fourth and the fifth setting for $\alpha = 2$ and $j = 0$. However, with these choices for the parameters both contributions cancel out. For the coefficient at p^2 we can combine $k = 0$ with $\gamma = 2$ and $k = 1$ with $\gamma = 1$. For $k = 0$ only the second and the seventh setting might contribute with $\alpha = 2$ and $j = 0$; again the resulting expressions cancel out. For $k = 1$ the fourth and the fifth setting contribute for $\alpha = 2$ and $j = 0$ giving rise to the expansion

   $$(m-2)p^2 + \mathcal{O}(p^3).$$

2. Terms of the pattern $\frac{c \cdot p^\gamma (1-p)^{m-\alpha-j-1}}{\alpha p - \alpha - p}$ with the following values for the parameters c and γ:

   $c = 1$, $\gamma = j + \alpha + 1$; $c = -1$, $\gamma = j + \alpha$; $c = -1$, $\gamma = j + \alpha + 2$;
   $c = 1$, $\gamma = j + 2\alpha - 1$; $c = -1$, $\gamma = j + \alpha - 1$; $c = -1$, $\gamma = j + 2\alpha$;
   $c = 1$, $\gamma = j + 2\alpha + 1$;

   Again, we number these settings left to right from top to button. Computing a series expansion at $p = 0$ yields

   $$\frac{c \cdot p^\gamma (1-p)^{m-\alpha-j-1}}{\alpha p - \alpha - p} = c \cdot p^\gamma \cdot \left(-\frac{1}{\alpha} + \frac{(m-\alpha-j-2)\alpha + 1}{\alpha^2} p + \mathcal{O}(p^2) \right).$$

   Now $\alpha \geq 2$ and $j \geq 0$ implies $\gamma \geq 1$ for all the parameter settings such that no contribution for the constant term exists. The coefficient $\frac{1}{2}$ at p is given by the sole possible contribution which results from the choices $\alpha = 2$ and $j = 0$ for the fifth setting. The coefficient at p^2 is made up of several contributions by the fifth setting ($\alpha = 2$, $j = 0$) for $\gamma = 1$ combined with $\frac{(m-\alpha-j-2)\alpha+1}{\alpha^2} p$ and by the second ($\alpha = 2$, $j = 0$) and fifth ($\alpha = 3$, $j = 0$ and $\alpha = 2$, $j = 1$) setting for $\gamma = 2$ combined with $-\frac{1}{\alpha}$. Note that we have to take the binomial coefficient $\binom{m-\alpha-1}{j}$ into account whenever $j \neq 0$ holds. Altogether, this leads to the expansion

   $$\frac{1}{2} p + \left(\frac{13}{12} - \frac{1}{3} \delta_{m,3} \right) p^2 + \mathcal{O}(p^3).$$

   Here, δ is Kronecker's symbol which shows up since $\alpha = 3$ is impossible for $m = 3$.

3. Terms of the pattern $\frac{c \cdot p^\gamma (1-p)^\delta}{1 + \alpha p - p}$ with the following values for the parameters c, γ and δ:

   $c = 1$, $\gamma = j + 1$, $\delta = m - j - 4$; $c = -1$, $\gamma = j + 2$, $\delta = m - j - 4$;
   $c = 3$, $\gamma = 2j + 2$, $\delta = 2m - 2j - 4$; $c = 2$, $\gamma = 2j + 4$, $\delta = 2m - 2j - 4$;
   $c = -4$, $\gamma = 2j + 3$, $\delta = 2m - 2j - 4$; $c = -1$, $\gamma = 2j + 1$, $\delta = 2m - 2j - 4$;
   $c = -\alpha$, $\gamma = 2j + 2$, $\delta = 2m - 2j - 4$; $c = -2\alpha$, $\gamma = 2j + 4$, $\delta = 2m - 2j - 4$;
   $c = 2\alpha$, $\gamma = 2j + 3$, $\delta = 2m - 2j - 4$; $c = 2$, $\gamma = 2j + 1$, $\delta = m - j - 4$;
   $c = -1$, $\gamma = 2j + 1$, $\delta = m - j - 4 + \alpha$; $c = -18$, $\gamma = 2j + 4$, $\delta = m - j - 4$;
   $c = -7$, $\gamma = 2j + 3$, $\delta = m - j - 4 + \alpha$; $c = 17$, $\gamma = 2j + 3$, $\delta = m - j - 4$;
   $c = 4$, $\gamma = 2j + 2$, $\delta = m - j - 4 + \alpha$; $c = -9$, $\gamma = 2j + 2$, $\delta = m - j - 4$;
   $c = 10$, $\gamma = 2j + 5$, $\delta = m - j - 4$; $c = 6$, $\gamma = 2j + 4$, $\delta = m - j - 4 + \alpha$;
   $c = -2$, $\gamma = 2j + 6$, $\delta = m - j - 4$; $c = -2$, $\gamma = 2j + 5$, $\delta = m - j - 4 + \alpha$;

Let c_i resp. γ_i resp. δ_i be the parameter c resp. γ resp. δ of the i-th parameter setting numbered from left to right, top to button, $1 \le i \le 20$. With $j \ge 0$ all the γ_i are at least 1 such that the expansion

$$\frac{c \cdot p^\gamma (1-p)^\delta}{1+\alpha p - p} = c \cdot p^\gamma \cdot \big(1 + (1-\alpha-\delta)p + \mathcal{O}(p^2)\big)$$

implies that we have no constant term. For $j = 0$ and arbitrary α we observe $\gamma_k = 1$ for $k \in \{1, 6, 10, 11\}$. This leads to the contribution

$$\sum_{\alpha=2}^{m-1} \binom{m-\alpha-1}{0} (c_1 + c_6 + c_{10} + c_{11})p = (m-2)p.$$

There are several possibilities for a contribution to the coefficient at p^2. First, we can combine $\gamma = 1$ with $(1-\alpha-\delta)p$ which leads to

$$\sum_{\alpha=2}^{m-1} \binom{m-\alpha-1}{0} \sum_{k \in \{1,6,10,11\}} c_k(1-\alpha-\delta_k)p^2 = 5(m-2)p^2.$$

Second, we can combine $\gamma = 2$ with 1; we find $\gamma_k = 2$ for $k \in \{2, 3, 7, 15, 16\}$ and $j = 0$ leading to

$$\sum_{\alpha=2}^{m-1} \binom{m-\alpha-1}{0} (c_2 + c_3 + c_7 + c_{15} + c_{16})p^2 = \left(-\frac{5}{2}m - \frac{1}{2}m^2 + 7\right)p^2.$$

Additionally, the choice $j = 1$ implies $\gamma_1 = 2$ for arbitrary α, which leads to the contribution

$$\sum_{\alpha=2}^{m-1} \binom{m-\alpha-1}{1} c_1 p^2 = \left(\frac{1}{2}m^2 - \frac{5}{2}m + 3\right)p^2.$$

Adding all the contributions for the two alternatives $\gamma = 1$ and $\gamma = 2$ leads to

$$\left[5(m-2) + \left(-\frac{5}{2}m - \frac{1}{2}m^2 + 7\right) + \left(\frac{1}{2}m^2 - \frac{5}{2}m + 3\right)\right]p^2 = 0.$$

Therefore we finally find the following expansion of our sum for fixed $m \ge 3$ and $p \to 0$:

$$\frac{2m-3}{2}p + \left(\frac{13}{12} - \frac{1}{3}\delta_{m,3} + (m-2)\right)p^2 + \mathcal{O}(p^3).$$

It remains to determine the contribution of

$$\nu := \frac{1}{p(p-1)^2}\left(\frac{2(1-p)^{m+1}((1-p)^3+p)}{1+(m-1)p} - \frac{(1-p)^{2m}(2+(m-2)p)(1+2(p-1)p)}{1+(m-1)p}\right.$$
$$\left. - \frac{p^m(1-p^2(2+(p-2)p) + (p-1)p^m(p^2-mp+m+1))}{m(p-1)-p}\right)$$

to the expansion at $p = 0$. For this purpose we proceed in the same way as before. Expanding ν we observe terms of two different patterns.

1. Terms of the pattern $\frac{c \cdot p^\gamma (1-p)^\delta}{(p-1)^2(1+mp-p)}$ with

$$\begin{array}{ccc} c=2,\ \gamma=-1,\ \delta=m; & c=-6,\ \gamma=0,\ \delta=m; & c=10,\ \gamma=1,\ \delta=m; \\ c=-8,\ \gamma=2,\ \delta=m; & c=2,\ \gamma=3,\ \delta=m; & c=-2,\ \gamma=-1,\ \delta=2m; \\ c=-8,\ \gamma=1,\ \delta=2m; & c=6,\ \gamma=0,\ \delta=2m; & c=-m,\ \gamma=0,\ \delta=2m; \\ c=-2m,\ \gamma=2,\ \delta=2m; & c=2m,\ \gamma=1,\ \delta=2m; & c=4,\ \gamma=2,\ \delta=2m; \end{array}$$

Again, we use c_i resp. γ_i resp. δ_i to denote the parameter c resp. γ resp. δ of the i-th parameter setting numbered from left to right, top to button, $1 \leq i \leq 12$. Then the expansion

$$\frac{(1-p)^\delta}{(p-1)^2(1+mp-p)} = 1 + (3-\delta-m)p + \left(6 - \frac{7}{2}\delta + \frac{1}{2}\delta^2 + m(\delta - 4 + m)\right)p^2$$

$$+ \left(10 - \frac{1}{6}\delta^3 + 2\delta^2 - \frac{47}{6}\delta - 10m + \frac{9}{2}\delta m - \frac{1}{2}\delta^2 m + 5m^2 - \delta m^2 - m^3\right)p^3 + \mathcal{O}(p^4)$$

makes it possible to collect all the contributions to the expansion in question. Since $c_1 + c_6 = 0$ we have no contribution to p^{-1}. For the constant term we have contributions by the k-th parameter setting, $k \in \{1, 2, 6, 8, 9\}$, which sum up to m. The coefficient at p results from contributions by the k-th setting, $k \in \{1, 2, 3, 6, 7, 8, 9, 11\}$, which sum up to $2 - 2m^2$. The coefficient at p^2 results from $k \in \{1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12\}$ and is given by $2 + \frac{7}{3}m^3 - m^2 - \frac{10}{3}m$. Thus, the expansion of all the terms of ν belonging to the first pattern is given by

$$m + (2 - 2m^2)p + \left(2 + \frac{7}{3}m^3 - m^2 - \frac{10}{3}m\right)p^2 + \mathcal{O}(p^3).$$

2. Terms of the pattern $\frac{c \cdot p^\gamma}{(p-1)^2(mp-m-p)}$ with

$$\begin{array}{llll} c = -1,\ \gamma = m-1; & c = 2,\ \gamma = m+1; & c = 1,\ \gamma = m+3; & c = -2,\ \gamma = m+2; \\ c = -1,\ \gamma = 2m+2; & c = m,\ \gamma = 2m+1; & c = -2m,\ \gamma = 2m; & c = -1,\ \gamma = 2m; \\ c = 1,\ \gamma = 2m+1; & c = m,\ \gamma = 2m-1; & c = 1,\ \gamma = 2m-1; & \end{array}$$

Since γ grows with m for all eleven parameter settings and since

$$\frac{c \cdot p^\gamma}{(p-1)^2(mp-m-p)} = -\frac{1}{m} - \frac{3m-1}{m^2}p - \frac{6m^2-4m+1}{m^3}p^2 + \mathcal{O}(p^3)$$

holds, the only contribution to the expansion in question (recall that we are discussing the case $m \geq 3$) results from choosing $m = 3$. For this case all parameter settings together contribute $\frac{1}{3}p^2 + \mathcal{O}(p^3)$. For $m \geq 4$, the contributions of all terms of the pattern considered here are in $\mathcal{O}(p^3)$ and thus beyond the precision of our asymptotic.

Adding all the contributions made by the terms of τ and ν finally yields the expansion stated in the theorem. The cases $m = 1$ and $m = 2$ can be proved easily by an inspection of (2) for these specific values of m. □

Remarks:

1. The assumption of m being fixed is essential for the asymptotic given in the previous theorem. Like the coefficients of our asymptotic expansion the constant of the $\mathcal{O}$-term grows with m. As a consequence, our asymptotic is rather precise for small values of m even for p relatively close to $\frac{1}{2}$. However, with growing m we need p close to zero for our asymptotic to provide values being similar to the precise number of comparisons.

2. Considering $\Sigma = \{0, 1\}$ does not provide simplified results for the original Horspool algorithm. Since the pattern is processed in a fixed order from right to left, it is not possible to simplify the corresponding summations, e.g. by partitioning the set of patterns. From a mathematical point of view, the fixed order of processing implies that for almost all distributions the sums[5] $\sum_{j=1}^{m} t_j$ equal just for two patterns P_1, P_2 with $P_1[2..m] = P_2[2..m]$.

Based on the expected number of comparisons for searching a random pattern in a random text, it is now possible to compare the modified algorithm with the original one. The plot in Figure 3 shows the quotient $\mathbb{E}[C_n^{\mathsf{mod}}(m)]/\mathbb{E}[C_n^{\mathsf{org}}(m)]$, i.e. the expected number of comparisons

[5]Here we assume, that the definition of t_j is adapted to the case of the original algorithm as described on page 8.

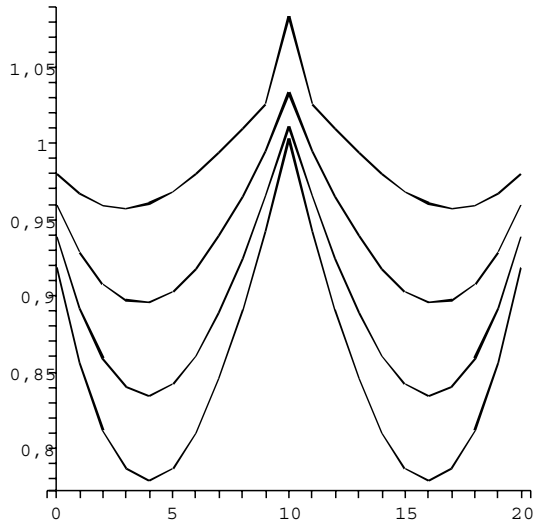


Figure 3: The expected number of comparisons made by the modified algorithm divided by the same number for the original one. The different curves correspond to different lengths $m \in \{2, 3, 4, 5\}$ of a random pattern. The x-axis determines the probability of symbol 1 according to $\pi_1 := \frac{x+1}{22}$.

made by the modified algorithm divided by the corresponding number for the original method, for $\Sigma = \{0, 1\}$, $\pi_1 := \frac{x+1}{22}$ and $m \in \{2, 3, 4, 5\}$. The highest curve corresponds to the case $m = 2$, the lowest to the case $m = 5$. The original algorithm is slightly better for a uniform distribution only. For every asymmetric distribution our modified algorithm runs faster. The longer the pattern becomes, the larger the potential advantage of the modified algorithm gets. In case of a random pattern of length 5 we need less than 80% of the running time of the original algorithm for the most beneficial distribution. The same behavior can be observed for alphabets of larger sizes. In Figure 4 you can find a plot of the same quotient for an alphabet of size 3 and a random pattern of size 5; the x and y axes represent the probabilities for two of the three symbols. Again, the original algorithm is faster for the uniform distribution. However, the potential advantage of our algorithm is even higher than in the case of a binary alphabet, needing less than 75% of the original algorithm's running time in the best case.

4 Simulations

In this section, we provide some simulation results in order to

- show the accuracy of our asymptotic formulae,
- to study the effect of using approximated probabilities (that result from sampling the text randomly) for the modified algorithm, and
- to see how the modified algorithm performs on real world data which is not generated according to our simple probability model.

For the first item we have generated a random text T of size 1000000 over the alphabet $\Sigma = \{A, C, G, U\}$ according to the probability distribution $\pi_A = \frac{9}{20}$, $\pi_C = \frac{1}{10}$, $\pi_G = \frac{1}{5}$ and $\pi_U = \frac{1}{4}$ and

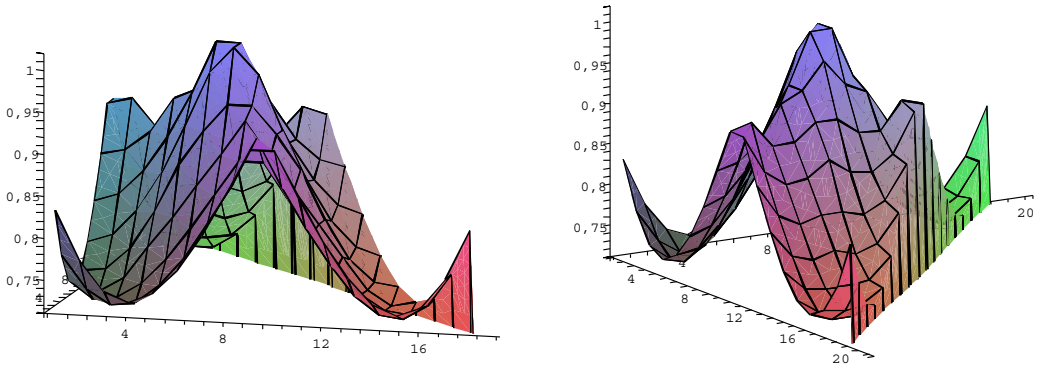


Figure 4: The expected number of comparisons made by the modified algorithm divided by the same number for the original one. The plots show this quotient for a random pattern of size 5 over an alphabet of size 3 for two different viewpoints. The x and y axes determine the probabilities of the symbols 1 and 2 according to $\pi_1 = \frac{x}{20}$ and $\pi_2 = \frac{y}{20}$.

searched for several patterns of different lengths. Table 2 shows the observed results. As we can see, the asymptotical number of comparisons given by Theorem 2 is rather precise with a relative error of only about one per thousand. Furthermore, we observe that there are patterns for which both algorithms have the same behavior. One example is the pattern $UUUGG$ for which both algorithms need the same number of comparisons because $\pi_G < \pi_U$ holds, such that the order in which the modified algorithm processes this pattern is identical to the one used by the original method. Finally, with the size of the pattern increasing, we observe a greater potential for cutting down on the number of comparisons for the modified algorithm.

In order to study the effect of using approximated probabilities for the modified Horspool algorithm we ran the following experiments: We generated random texts of size 250000 according to four different probability distributions. For each of these texts we used the modified algorithm so search for different patterns using random samples of different sizes to approximate the probabilities. These experiments (fixed text T and fixed pattern P) were repeated 50 times; the resulting average number of comparisons needed together with the number of comparisons needed when using the exact distribution of the symbols is shown in Table 3. We observe that using only an approximation of the probability distribution may be either an advantage or an disadvantage. For instance, pattern $AAACG$ needs a smaller number of comparisons for the random text according to distribution Π_1 for all sizes of the random sample considered. However, the same pattern needs more comparisons for all sample sizes and probability distribution Π_2. If we sum the average number of comparisons for all patterns and all distributions, we observe that the largest sample size $((|\Sigma|+1)\sqrt{|T|})$ leads to the best result, the smallest sample size $(\frac{1}{2}\sqrt{|T|})$ to the worst. This contradicts somehow the fact that the usage of the real probabilities (last column) leads to the largest total number of comparisons since we should assume that a larger sample size leads to a better approximation of the probabilities; but then, using the largest sample size should be closest to using the exact distribution. Furthermore, if we take the running time for sampling the text into account (assuming that the running time of one comparison of two symbols is similar to that of sampling a single symbol of the text), we get the converse situation: Adding the amount of 250 to each of the average numbers of the fifth column results to a total amount of 4219162.02 in contrast to a total number of 4228339.50 (resp. 4306816.08) comparisons which follow from the addition of 500 (resp. 2500) to each entry of the third (resp. fourth) column. As a consequence, we should expect that using a larger sample size does not pay off over a long run.

P	$\#P$ in T	comparisons observed		asymptotic formulae		as./ob. modified
		original	modified	original	modified	
$AAAAA$	18491	643567	643567	644970	644970	1.002180037
$AAACG$	1804	391173	388644	390920	387843	0.997938988
$ACACG$	426	388496	375071	388206	375606	1.001426397
$UCACG$	237	420538	405468	420557	406492	1.002525477
$UCCCG$	40	286655	281813	286055	281021	0.997189626
$UCGCG$	108	333183	324505	333259	326005	1.004622425
$UCCGG$	101	351441	331699	351584	328789	0.991226986
$UUUGG$	605	378200	378200	377609	377609	0.998437335
$UUUUU$	992	353235	353235	352783	352783	0.998720399
$UAGACGCA$	9	386239	301838	386114	302373	1.001772474
$AGGUAUAC$	26	438142	414726	438301	410599	0.990048852
$CAACUAGCAUACGAU$	0	614298	492315	614712	474548	0.963911317

Table 2: The number of comparisons observed when simulating the original (third column) and the modified (forth column) Horspool algorithm searching for different patterns P. The columns five and six show the corresponding asymptotic numbers of comparisons according to our formulae. The second column shows the number of occurrences of P in the random text T used for the simulations which can be assumed to be the relative probability of that pattern. The last column provides the quotient of the asymptotical divided by the observed number of comparisons for the modified Horspool algorithm.

For the third item we use the entire genome of Methanococcus jannaschii taken from the NCBI-database (sequence NC000909) as our text. This genome consists of 1664970 symbols (nucleotides) of which a few are letters not taken from the alphabet $\{A, C, G, T\}$ being placeholders for different possibilities for a nucleotide. We simply have omitted these letters which provided a text of length 1664957. For this text (genome) it is well-known that it possesses the following non-uniform diletter (dinucleotide) frequencies (see e.g. [ClB00]):

	A	C	G	T
A	0.134	0.039	0.060	0.111
C	0.055	0.033	0.008	0.059
G	0.057	0.027	0.034	0.039
T	0.098	0.056	0.055	0.134

Thus it obviously is no instance of our probability model from the average-case analysis. Nevertheless the four different letters are non-uniformly distributed according to the probabilities (computed as relative numbers of occurrence) $\mathsf{Pr}[A] = 0.34$, $\mathsf{Pr}[C] = 0.16$, $\mathsf{Pr}[G] = 0.16$ and $\mathsf{Pr}[T] = 0.34$ (all rounded to the second decimal digit). As our subsequent results will show, the modified Horspool algorithm will be superior to the original one even for this kind of data.
To compare the original to the modified algorithm with this sort of input we ran two different kinds of simulations. For the first kind we have searched for 1000 random patterns of length l, $l = 5, 10, 15, 20, 25, 50$, where each random pattern has been generated according to a multinomial distribution with $\mathsf{Pr}[A] = 0.34$, $\mathsf{Pr}[C] = 0.16$, $\mathsf{Pr}[G] = 0.16$ and $\mathsf{Pr}[T] = 0.34$. For the second kind, the same simulations have been performed, now choosing random patterns according to the uniform distribution. In both cases and for both algorithms we have determined the to total number of wins, i.e. the total number of patterns for which the algorithm outperforms its competitor, together with the average number of comparisons needed by the algorithm to search the entire text for all random patterns of the same size l. The corresponding results are represented in Table 4. Please note that the number of wins does not always sum to 1000 since in some cases both algorithms needed the same number of comparisons thus giving rise for a tie. As we can see, the advantage of the modified algorithms over the original one does not depend on the simplifying assumptions made for random texts within our analysis. Additionally, it is not necessary that random patterns are taken from the same distribution as the text (as assumed for our average-case

P	Π	sample size $\sqrt{\lvert T\rvert} = 500$	$(\lvert\Sigma\rvert+1)\sqrt{\lvert T\rvert} = 2500$	$\frac{1}{2}\sqrt{\lvert T\rvert} = 250$	ex. distribution
$AAAAA$	Π_1	87707.00	87707.00	87707.00	87707
	Π_2	97568.00	97568.00	97568.00	97568
	Π_3	137401.00	137401.00	137401.00	137401
	Π_4	160773.00	160773.00	160773.00	160773
$AAACG$	Π_1	116365.64	117722.60	116502.48	125920
	Π_2	115984.40	114040.24	117457.72	112722
	Π_3	131757.00	131757.00	132392.54	131757
	Π_4	82683.00	82683.00	82683.00	82683
$ACACG$	Π_1	117537.88	119290.76	117642.80	124426
	Π_2	117309.00	113162.92	122104.58	112072
	Π_3	129745.00	129745.00	129979.40	129745
	Π_4	79949.00	79949.00	79949.00	79949
$UCACG$	Π_1	130588.52	128542.74	129399.34	133371
	Π_2	126723.72	127101.98	127785.82	127021
	Π_3	121376.00	121376.00	121396.64	121376
	Π_4	86280.50	86280.00	86279.00	86292
$UCCCG$	Π_1	105378.92	106728.98	105674.32	110334
	Π_2	101747.66	100993.36	100014.62	100836
	Π_3	84052.00	84052.00	84052.00	84052
	Π_4	60822.72	60822.92	60823.08	60822
$UCGCG$	Π_1	127387.42	129384.80	125661.40	133019
	Π_2	121952.32	122208.36	121376.38	121758
	Π_3	99157.82	99074.00	99157.82	99074
	Π_4	72581.06	72582.68	72583.04	72587
$UCCGG$	Π_1	126307.72	127522.70	127972.42	131834
	Π_2	121356.74	121579.20	120691.76	121284
	Π_3	95870.00	95870.00	95936.70	95870
	Π_4	76562.80	76564.08	76564.08	76582
$UUUGG$	Π_1	115178.12	114219.96	114219.96	120448
	Π_2	107047.40	108004.30	105953.80	108141
	Π_3	77860.00	77860.00	77860.00	77860
	Π_4	113878.68	115359.60	112397.76	120296
$UAGACGCA$	Π_1	108223.18	109088.40	109358.86	113183
	Π_2	111639.54	110760.78	111648.42	110580
	Π_3	102248.42	102191.00	102580.92	102191
	Π_4	67577.00	67571.96	67573.64	67556
$AGGUAUAC$	Π_1	98453.86	95495.72	96455.44	101408
	Π_2	94157.88	94568.64	94392.80	94556
	Π_3	75645.00	75645.00	75652.00	75645
	Π_4	103504.58	103567.40	103539.48	103707
Σ		4208339.50	4206816.08	4209162.02	4254406

Table 3: The average number of comparisons needed to search for pattern P in a random text of size 250000 generated according to the probability distributions $\Pi_1 = \{\pi_A = \frac{1}{4}, \pi_C = \frac{1}{4}, \pi_G = \frac{1}{4}, \pi_U = \frac{1}{4}\}, \Pi_2 = \{\pi_A = \frac{10}{34}, \pi_C = \frac{9}{34}, \pi_G = \frac{8}{34}, \pi_U = \frac{7}{34}\}, \Pi_3 = \{\pi_A = \frac{4}{10}, \pi_C = \frac{3}{10}, \pi_G = \frac{2}{10}, \pi_U = \frac{1}{10}\}, \Pi_4 = \{\pi_A = \frac{9}{20}, \pi_C = \frac{1}{20}, \pi_G = \frac{1}{4}, \pi_U = \frac{1}{4}\}$. Each entry from the third to the fifth column corresponds to this average taken over 50 independent runs for the given pattern and a fixed text using a random sample of the corresponding size in order to approximate the probability distribution. The rightmost column provides the number of comparisons needed by the modified Horspool algorithm using the exact probability distribution. The last row provides the sum of all comparisons needed.

l	multinomial		uniform	
	orig. algorithm	mod. algorithm	orig. algorithm	mod. algorithm
5	*214* 889955.17	*718* 838769.23	*217* 806068.45	*726* 761028.73
10	*102* 743853.53	*896* 657868.17	*209* 684720.97	*791* 632732.47
15	*112* 707489.40	*888* 618483.51	*214* 658627.78	*786* 606088.09
20	*92* 715433.77	*908* 620506.15	*210* 666914.43	*790* 612703.16
25	*117* 699424.07	*883* 606884.82	*215* 665760.25	*785* 610888.14
50	*96* 703774.44	*904* 608933.88	*224* 674893.91	*776* 618344.42

Table 4: The results of our simulations performed on the genome of M.jannaschii and 1000 random patterns of size l taken from a multinomial and from a uniform distribution. The italic number represents the number of wins of the corresponding algorithm, the number in roman is the average number of comparisons needed to find all 1000 patterns of the corresponding length.

analysis) in order to benefit from non-uniformly distributed symbols in the average.

5 Conclusions

In this paper we have presented a modification of Horspool's string matching algorithm similar to Sunday's optimal mismatch algorithm (which is a modification of the Boyer-Moore algorithm). For both, the idea to speed up the algorithm is to use statistical properties of the (random) input which may be derived from knowledge on the source of the input, e.g. when processing natural languages with well-known symbol distributions, or from a random sampling performed as a preprocessing. For the simple model of random texts according to a multinomial distribution we have proven that this modification leads to a speedup of the Horspool algorithm in the average. Additionally, our simulations on DNA data provides some evidence that this remains also valid when the input is equipped with *intersymbol dependencies* (like non-uniform dinucleotide frequencies).

The example of the modified string matching algorithm shows that we can benefit for our algorithms from statistical knowledge on the input; in the opinion of the author, it is worth investigating if this strategy can be advantageously applied to other problems and their algorithmic solutions.

References

[Boy77] R. S. Boyer and J. S. Moore, *A Fast String Searching Algorithm*, Communication of the ACM **20**, 767-772, 1977.

[ChL04] C. Charras, T. Lecroq, *Handbook of Exact String Matching Algorithms*, King's College London Publications, 2004.

[ClB00] P. Clote and R. Backofen, *Computational Molecular Biology*, Wiley, 2000.

[CKKS05] P. Clote, E. Kranakis, D. Krizanc and L. Stacho, *Asymptotics of Random RNA*, in Discrete Applied Mathematics, Special Issue on Com-

putational Biology, P. Pevzner and S. Istrail, editors, to appear (see http://www.scs.carleton.ca/∼kranakis/pub.html).

[CLR90] T. Cormen, C. Leiserson and R. Rivest, *Introduction to Algorithms*, MIT Press, 1990.

[HTF01] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*, Springer Series in Statistics, 2001.

[Hor80] R. Horspool, *Practical Fast Searching in Strings*, Software Practice and Experience **10**, 501-506, 1980.

[KMP77] D. Knuth, J. Morris and V. Pratt, *Fast Pattern Matching in Strings*, SIAM Journal on Computing **6**, 323-350, 1977.

[MSR96] H. Mahmoud, R. Smith and M. Régnier, *Analysis of Boyer-Moore-Horspool String-Matching Heuristic*, Random Structures & Algorithms **10**, 169-186, 1997.

[Sed92] R. Sedgewick, *Algorithms*, Addison-Wesley, 1992.

[Sun90] D. Sunday, *A Very Fast Substring Search Algorithm*, Communications of the ACM **33**, 132-143, 1990.

6 Appendix

If the string matching problem has to be solved in the context of natural languages, we should use well-known statistics for the probabilities of the different symbols in order to run our algorithm. However, we will show in this section that it is possible to get precise estimates of those probabilities for arbitrary texts in time sublinear in the length of the text. For this purpose we use methods from statistical learning theory as described in [HTF01]. We will construct a maximum likelihood estimator for the probabilities based on random samples of the text. Assuming a multinomial distribution of the symbols, the likelihood $L(p_1, \ldots, p_n)$ is given by

$$L(p_1, \ldots, p_n) = p_1^{r_1} p_2^{r_2} \cdots p_{n-1}^{r_{n-1}} p_n^{r_n}$$

where p_i denotes the (unknown) probability of the i-th symbol in Σ and r_i is the number of occurrences of this symbol in the random sample, $1 \leq i \leq n$. To ensure that the probabilities sum to one we set $p_n = 1 - p_1 - p_2 - \cdots - p_{n-1}$ such that $L(p_1, \ldots, p_n) = p_1^{r_1} p_2^{r_2} \cdots p_{n-1}^{r_{n-1}} (1 - p_1 - p_2 - \cdots - p_{n-1})^{r_n}$ holds. To determine the maximum likelihood estimator we have to set the partial derivative of $\ln(L(p_1, \ldots, p_n))$ with respect to p_i equal to zero, $1 \leq i \leq n-1$, and solve the resulting equations. We find

$$\frac{\partial}{\partial p_i} \ln(L(p_1, \ldots, p_n)) = \frac{r_i}{p_i} - \frac{r_n}{1 - p_1 - \ldots - p_{n-1}}, \; 1 \leq i < n,$$

with the solution

$$p_i = \frac{r_i}{r_1 + r_2 + \ldots + r_n} = \frac{r_i}{N}, \; 1 \leq i \leq n,$$

for N the size of our random sample. Note that the solution for p_n results from

$$p_n = 1 - \sum_{j=1}^{n-1} \frac{r_j}{\sum_{k=1}^{n} r_k} = \frac{\sum_{k=1}^{n} r_k - \sum_{j=1}^{n-1} r_k}{\sum_{k=1}^{n} r_k} = \frac{r_n}{r_1 + r_2 + \ldots + r_n}.$$

Thus, the maximum likelihood estimator for p_i is given by the number of occurrences of the i-th symbol in our sample divided by the sample size.

It is well-known that a maximum likelihood estimator is consistent and thus it converges to the real probabilities. In our case this is obvious for $N = n$ since then we use the exact relative number of occurrences as an *estimator* for the probabilities. However, we do not want to sample the entire text and therefore it is necessary to get a representation for the error made when considering samples of smaller sizes only. For this purpose we have to determine $I(P)_{i,j} := -\frac{\partial^2 \ln(L(p_1, \ldots, p_n))}{\partial p_1 \partial p_j}$. We find

$$I(P)_{i,j} = \begin{cases} \frac{r_i}{p_i^2} + \frac{r_n}{p_n^2} & \text{for } i = j \\ \frac{r_n}{p_n^2} & \text{otherwise} \end{cases}.$$

From this we obtain the so called Fisher information $F(P)$ by taking expectations of $I(P)$ over all possible random samples (over all possible r_i), i.e. $F(P) = \mathbb{E}[I(P)]$. In our case, the assumed multinomial distribution for the symbols implies $\mathbb{E}[r_i] = N p_i$ such that

$$F(P)_{i,j} = \mathbb{E}[I(P)_{i,j}] = \begin{cases} \frac{N}{p_i} + \frac{N}{p_n} & \text{for } i = j \\ \frac{N}{p_n} & \text{otherwise} \end{cases}$$

holds. To get an approximation for the standard errors made by our estimators we need the inverse of matrix $(F(P))$, for which we find

$$\left((F(P))^{-1} \right)_{i,j} = \begin{cases} \frac{r_i (N - r_i + r_n)}{N^2 (N + r_n)} & \text{for } i = j \\ -\frac{r_i \cdot r_j}{N^2 (N + r_n)} & \text{otherwise} \end{cases}.$$

The entries of the main diagonal of this inverse matrix are the variances of the multivariate normal distribution to which the sampling distribution of the maximum likelihood estimators converges.

Therefore an $\alpha\%$ confidence interval for the real probability of the i-th symbol is approximately given by

$$\left(\frac{r_i}{N} - z^\alpha \sqrt{\left((F(P))^{-1}\right)_{i,i}}, \frac{r_i}{N} + z^\alpha \sqrt{\left((F(P))^{-1}\right)_{i,i}}\right),$$

for z^α the α percentile of the standard normal distribution. For example, for $\alpha = 0.95$ (i.e. for a 95% confidence interval) we have $z^\alpha = 1.96$. The size of the confidence interval is therefore given by

$$2z^\alpha \sqrt{\left((F(P))^{-1}\right)_{i,i}} = \frac{2z^\alpha}{N}\sqrt{r_i - \frac{r_i^2}{N + r_n}} \leq \frac{2z^\alpha}{N}\sqrt{r_i} \stackrel{\mathbb{E}}{=} \frac{2z^\alpha}{N}\sqrt{p_i N}. \tag{3}$$

Here, the last equality holds in expectation only; it tells us that the most probable symbols need the largest number of samples to get a good approximation for their probability. As a consequence of equation (3) the size of the confidence interval decreases in expectation at least like $1/\sqrt{N}$ to zero with an increasing sample size N for all the p_i and for arbitrary fix α. It is therefore sufficient to use a sublinear sample of size $n^{1-\epsilon}$ for $\epsilon > 0$ to get an approximation for the distribution of the symbols of high precision. For the application of our algorithm this implies a sublinear contribution to the total running time in cases where the distribution of the symbols is not known. Thus for large texts, our algorithm will be faster than the original Horspool algorithm for most of the inputs (distributions) even if we first have to compute estimates for the symbol's probabilities.