Designing optimal-and fast-on-average pattern matching algorithms

Gilles Didier, Laurent Tichit

Designing optimal- and fast-on-average pattern matching algorithms

Gilles Didier and Laurent Tichit
Aix-Marseille Université, CNRS, Centrale Marseille, I2M UMR7373, Marseille, France
E-mail: {gilles.didier,laurent.tichit}@univ-amu.fr

May 2, 2016

Abstract

Given a pattern w and a text t, the speed of a pattern matching algorithm over t with regard to w, is the ratio of the length of t to the number of text accesses performed to search w into t. We first propose a general method for computing the limit of the expected speed of pattern matching algorithms, with regard to w, over iid texts. Next, we show how to determine the greatest speed which can be achieved among a large class of algorithms, altogether with an algorithm running this speed. Since the complexity of this determination make it impossible to deal with patterns of length greater than 4, we propose a polynomial heuristic. Finally, our approaches are compared with 9 pre-existing pattern matching algorithms from both a theoretical and a practical point of view, i.e. both in terms of limit expected speed on iid texts, and in terms of observed average speed on real data. In all cases, the pre-existing algorithms are outperformed.

1 Introduction

We focus on algorithms solving the online string matching problem, which consists in reporting all, and only the occurrence positions of a pattern w in a text t (*online* meaning that no pre-processing of the text is allowed). As one of the oldest problems addressed in computer science, it has been extensively studied. We refer to [10] for a comprehensive list and an evaluation of all the pattern matching algorithms developed so far. By the authors' count, more than 80 algorithms have already been proposed, among which more than a half were published during the last ten years. This fact sounds quite paradoxical, since the Morris-Pratt algorithm, which is optimal in terms of worst case analysis, dates back to 1970.

A possible explanation is that there is wide gap between the worst case complexity of algorithms and their computation times on real data. For instance, there are pattern matching algorithms with non-linear worst case complexities, which perform much better than Morris-Pratt on English texts. Basically, the average case analysis is way more suited to assess the relevance of a pattern matching algorithm from a practical point of view. The average case analysis of some pattern matching algorithms, notably Boyer-Moore-Horspool and Knuth-Morris-Pratt, has already been carried out from various points of view [27, 12, 4, 3, 16, 21, 22, 25]. We provide here a general method for studying the limit average behavior of a pattern algorithm over iid texts. More precisely, following [18], we consider the limit expectation of the ratio of the text length to the number of text accesses performed by an algorithm for searching a pattern w in iid texts. This limit expectation is called the asymptotic speed of the algorithm with regard to w under the iid model. The computation of the asymptotic speed is based on w-matching machines which are automata-like structures able to simulate the behavior of a pattern matching algorithm while searching the pattern w. The underlying idea is the same as in [18, 19, 20, 17] and can be seen as a generalization of the string matching automaton [7].

In the companion paper, G. Didier provided a theoretical analysis of the asymptotic speed of pattern matching algorithms over iid texts [8]. In particular, he showed that, for a given pattern w, the greatest asymptotic speed among a large class of pattern matching algorithms, is achieved by a w-matching machine in which the states are essentially subsets of positions of w. Such machines are called *strategies* below.

We provide here a brute force algorithm computing the *Fastest* strategy for a given pattern w and the frequencies of an iid model. The algorithm is based on an original structure associated to the pattern w and called its position lattice, which gives a full representation of the overlap relations between the subsets of positions of w.

Since the brute force algorithm cannot be applied on patterns of length greater than 4, because of its (very high) time-complexity, we propose a polynomial *K-Heuristic*, in which the polynomial order K may be chosen by the user.

The Fastest and K-Heuristic approaches are finally compared with 9 several pre-existing pattern matching algorithms:

- from a theoretical point of view, by computing their limit expected speeds with regard to various patterns and iid models,
- from a practical point of view, by computing their average speeds over two sources (an English text and a DNA sequence).

In both cases, the Fastest and K-Heuristic (with K large enough) approaches outperform the pre-existing algorithms.

The software and the data used to perform the tests are available at `https://github.com/gilles-didier/Matchines.git`.

The rest of the paper is organized as follows. Section 2 presents the notations and recalls some concepts and results from [8]. It is followed by two sections which introduce the central objects of this work: the strategies and the position lattice of a pattern. In particular, we provide an algorithm computing the position lattice of a given pattern. Section 5 shows how to use the position lattice of a pattern to obtain the Fastest strategy with regard to this pattern and an iid model. In Section 6, we provide a polynomial heuristic allowing to compute fast strategies. Section 7 presents the results of various comparisons between 9 pre-existing pattern matching algorithms, the K-Heuristic and, each time it is possible, the Fastest strategy. The results are discussed in the last section.

2 Notations and definitions

2.1 Notations and general definition

For all finite sets $\mathcal{S}$, $\mathcal{P}(\mathcal{S})$ is the power set of $\mathcal{S}$ and $|\mathcal{S}|$ is its cardinal. An *alphabet* is a finite set $\mathcal{A}$ of elements called *letters* or *symbols*.

A *word*, a *text* or a *pattern* on $\mathcal{A}$ is a finite sequence of symbols of $\mathcal{A}$. We put $|v|$ for the length of a word v. Words are indexed from 0, i.e. $v = v_0 v_1 \dots v_{|v|-1}$. We write $v_{[i,j]}$ for the subword of v starting at its position i and ending at its position j, i.e. $v_{[i,j]} = v_i v_{i+1} \dots v_j$. The *concatenate* of two words u and v is the word $uv = u_0 u_1 \dots u_{|u|-1} v_0 v_1 \dots v_{|v|-1}$.

For any length $n \geq 0$, we note $\mathcal{A}^n$ the set of words of length n on $\mathcal{A}$, and $\mathcal{A}^\star$, the set of finite words on $\mathcal{A}$, i.e. $\mathcal{A}^\star = \bigcup_{n=0}^{\infty} \mathcal{A}^n$.

Unless otherwise specified, all the texts and patterns considered below are on a fixed alphabet $\mathcal{A}$.

A *pattern matching algorithm* takes a pattern w and a text t as inputs an reports all, and only the occurrence positions of w in t. For all patterns w, we say that two pattern matching algorithms are *w-equivalent* if, for all texts t, they access exactly the same positions of t on the input (w, t).

2.2 Matching machines and the generic algorithm [8]

For all patterns w, a *w-matching machine* is 6-uple $(Q, o, F, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\gamma})$ where

- Q is a finite set of states,
- $o \in Q$ is the initial state,
- $F \subset Q$ is the subset of pre-match states,
- $\boldsymbol{\alpha} : Q \to \mathbb{N}$ is the next-position-to-check function, which is such that for all $q \in F$, $\boldsymbol{\alpha}(q) < |w|$,
- $\boldsymbol{\delta} : Q \times \mathcal{A} \to Q$ is the transition state function,
- $\boldsymbol{\gamma} : Q \times \mathcal{A} \to \mathbb{N}$ is the shift function.

By convention, the set of states of a matching machine always contains a *sink state* $\odot$, which is such that, for all symbols $x \in \mathcal{A}$, $\boldsymbol{\delta}(\odot, x) = \odot$ and $\boldsymbol{\gamma}(\odot, x) = 0$.

The *order* O_Γ of a matching machine $\Gamma = (Q, o, F, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\gamma})$ is defined as $O_\Gamma = \max_{q \in Q}\{\boldsymbol{\alpha}(q)\}$.

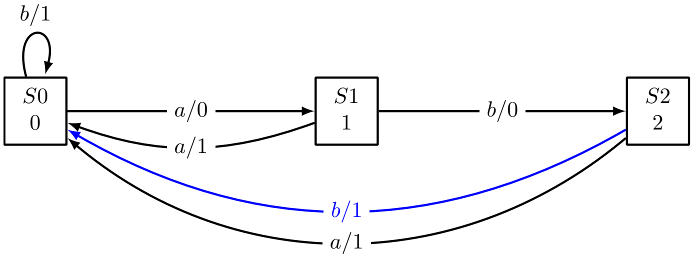

Figure 1: *abb*-matching machine of the naive algorithm. The next-position-to check are displayed below all states $S0$, $S1$ and $S2$. Edges from states Si are labelled with "$x/\boldsymbol{\gamma}(Si, x)$" for all symbols x. The transition associated to a match is blue-colored.

The w-matching machines carry the same information as the *Deterministic Arithmetic Automatons* defined in [19, 20].

The generic algorithm takes a w-machine machine and a text t as inputs and outputs positions of t (Algorithm 1).

```
input  : a $w$-matching machine $(Q, o, F, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\gamma})$ and a text $t$
output: all the occurrence positions of $w$ in $t$ (hopefully)

$(q, p) \leftarrow (o, 0)$
while $p \leq |t| - |w|$ do
    if $q \in F$ and $t_{p+\boldsymbol{\alpha}(q)} = w_{\boldsymbol{\alpha}(q)}$ then
        print " occurrence at position $p$ "
    $(q, p) \leftarrow (\boldsymbol{\delta}(q, t_{p+\boldsymbol{\alpha}(q)}), p + \boldsymbol{\gamma}(q, t_{p+\boldsymbol{\alpha}(q)}))$
```

Algorithm 1: The generic algorithm

A w-matching machine Γ is *valid* if, for all texts t, the execution of the generic algorithm on the input (Γ, t) outputs all, and only the occurrence positions of w in t.

We claim that for all the pattern matching algorithms developed so far and all patterns w, there exists a w-matching machine Γ which is such that, for all texts t, the generic algorithm and the pattern matching algorithm access exactly the same positions of t on the inputs (Γ, t) and (w, t) respectively [8]. For instance, Figure 1 displays a *abb*-matching machine which is *abb*-equivalent to the naive algorithm.

2.3 Full-memory expansion – standard matching machines [8]

For all $n \in \mathrm{N}$, R_n is the set of subsets H of $\{0, \dots, n\} \times \mathcal{A}$ verifying that, for all $i \in \{0, \dots, n\}$, there exists at most one pair in H with i as first entry.

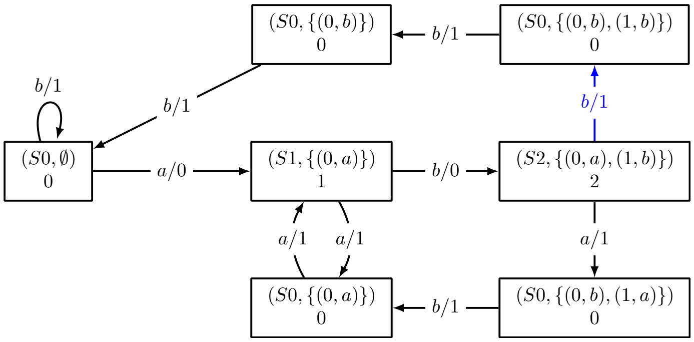

Figure 2: Full memory expansion of the *abb*-matching machine of Figure 1.

For all $H \in R_n$, we put $\mathbf{f}(H)$ for the set comprising all the first entries of the pairs in H, namely

$$\mathbf{f}(H) = \{i \mid \exists x \in \mathcal{A} \text{ with } (i,x) \in H\}.$$

For all $k \in \mathrm{N}$ and $H \in R_n$, the *k-shifted* of H is

$$\overset{k}{\overleftarrow{H}} = \{(u-k,y) \mid (u,y) \in H \text{ with } u \geq k\},$$

i.e. the subset of R_n obtained by subtracting k from the first entries of the pairs in H and by keeping only the pairs with non-negative first entries.

The *full memory expansion* of a w-matching machine $\Gamma = (Q, o, F, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\gamma})$ is the w-matching machine $\Gamma^\star$ obtained by removing the unreachable states of $\Gamma' = (Q', o', F', \boldsymbol{\alpha}', \boldsymbol{\delta}', \boldsymbol{\gamma}')$, defined as:

- $Q' = Q \times R_{O_\Gamma}$
- $o' = (o, \emptyset)$
- $\boldsymbol{\alpha}'((q,H)) = \boldsymbol{\alpha}(q)$
- $\boldsymbol{\gamma}'((q,H),x) = \boldsymbol{\gamma}(q,x)$
- $F' = F \times R_{O_\Gamma}$
-

$$\begin{aligned}&\boldsymbol{\delta}'((q,H),x)\\ &\quad = \begin{cases} (\boldsymbol{\delta}(q,x), \overset{\boldsymbol{\gamma}(q,x)}{\overleftarrow{H \cup \{(\boldsymbol{\alpha}(q),x)\}}}) & \text{if } \forall a \in \mathcal{A}, (\boldsymbol{\alpha}(q),a) \notin H \\ \odot & \text{if } \exists a \neq x \text{ s.t. } (\boldsymbol{\alpha}(q),a) \in H \\ (\boldsymbol{\delta}(q,x), \overset{\boldsymbol{\gamma}(q,x)}{\overleftarrow{H}}\ ) & \text{if } (\boldsymbol{\alpha}(q),x) \in H \end{cases}\end{aligned}$$

By construction, at all iterations of the generic algorithm on the input $(\Gamma^\star, t)$, if the current state and position are (q, H) and p, respectively, then the positions of $\{j + p \mid j \in \mathbf{f}(H)\}$ are exactly the positions of t greater than p accessed so far (the second entries of the corresponding elements of H give the symbols read).

For all texts t, the generic algorithm access the same positions of t on the inputs (Γ, t) and $(\Gamma^\star, t)$ [8].

A w-matching machine Γ is *standard* if each state q of Γ appears in a unique pair/state of its full memory expansion. For all states q of a standard matching machine Γ, we put $\mathbf{h}_\Gamma(q)$ for the second entry of the unique pair/state of $\Gamma^\star$ in which q appears.

We implemented a basic algorithm computing the full-memory expansion $\Gamma^\star = (Q^\star, o^\star, F^\star, \boldsymbol{\alpha}^\star, \boldsymbol{\delta}^\star, \boldsymbol{\gamma}^\star)$ of a w-matching machine $\Gamma = (Q, o, F, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\gamma})$ in $O(|w|.|Q^\star|)$ time. We have $|Q^\star| \leq (\mathcal{A} + 1)^{|w|}|Q|$ but the size of $Q^\star$ may vary a lot with regard to the matching machine/algorithm considered.

2.4 iid and Markov models

An *independent identically distributed (iid)* model (aka *Bernoulli* model) is fully specified by a probability distribution π on the alphabet (i.e. π_x is the probability of the symbol x in the model). Such a model will be simply referred to as "π" below. Under π, the probability of a text t is

$$\mathrm{p}_\pi(t) = \prod_{i=0}^{|t|-1} \pi(t_i).$$

A *Markov* model M over a given set of states Q is a 2-uple (π_M, δ_M), where π_M is a probability distribution on Q (the initial distribution) and δ_M associates a pair of states (q, q') with the probability for q to be followed by q' (the transition probability). Under a Markov model $M = (\pi_M, \delta_M)$, the probability of a sequence s of states is

$$\mathrm{p}_M(s) = \pi_M(s_0) \prod_{i=0}^{|s|-1} \delta_M(s_0, s_1).$$

Theorem 1 ([8])**.** *Let $\Gamma = (Q, o, F, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\gamma})$ be a w-matching machine. If a text t follows an iid model and Γ is standard then the sequence of states parsed by the generic algorithm on the input (Γ, t) follows a Markov model (π_M, δ_M).*

Proof. Whatever the text model and the machine, the sequence of states always starts with o with probability 1. We have $\pi_M(o) = 1$ and $\pi_M(q) = 0$ for all $q \neq o$.

The probability $\delta_M(q, q')$ that the state q' follows the state q during an execution of the generic algorithm, is equal to:

- 1, if there exists a symbol x such that $\boldsymbol{\delta}(q, x) = q'$ and $(\boldsymbol{\alpha}(q), x) \in \mathbf{h}_\Gamma(q)$),

- $\displaystyle\sum_{x \text{ s.t. } \boldsymbol{\delta}(q,x)=q'} \pi(x)$, otherwise,

independently of the previous states. □

2.5 Asymptotic speed

Let $\mathcal{M}$ be a text model and $\mathbf{A}$ be an algorithm. The *w-asymptotic speed* of $\mathbf{A}$ under $\mathcal{M}$ is the limit expectation, under $\mathcal{M}$, of the ratio of the text length to the number of text accesses performed by $\mathbf{A}$ [8]. Namely, by putting $a_{\mathbf{A}}(t)$ for the number of text accesses performed by $\mathbf{A}$ to parse t and $p_{\mathcal{M}}(t)$ for the probability of t with regard to $\mathcal{M}$, the asymptotic speed of $\mathbf{A}$ under $\mathcal{M}$ is

$$\mathrm{AS}_{\mathcal{M}}(\mathbf{A}) = \lim_{n\to\infty} \sum_{t\in\mathcal{A}^n} \frac{|t|}{a_{\mathbf{A}}(t)} p_{\mathcal{M}}(t).$$

The asymptotic speed $\mathrm{AS}_{\mathcal{M}}(\Gamma)$ of a w-matching machines Γ is that the generic algorithm with Γ as first input. From Theorem 5 of [8], the asymptotic speed of a standard w-matching machine $\Gamma = (Q, o, F, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\gamma})$ under an iid model π exists and is given by

$$\mathrm{AS}_{\pi}(\Gamma) = \sum_{q\in Q} \alpha_q E(q), \tag{1}$$

where $(\alpha_q)_{q\in Q}$ are the limit frequencies of the states of the Markov model associated to Γ and π in Theorem 1, and

$$E(q) = \begin{cases} \boldsymbol{\gamma}(q,x) & \text{if } (\boldsymbol{\alpha}(q), x) \in \mathbf{h}_{\Gamma}(q)), \\ \sum_x \boldsymbol{\gamma}(q,x)\pi_x & \text{otherwise.} \end{cases}$$

Computing the asymptotic speed of a pattern matching algorithm, with regard to a pattern w and an iid model π is performed by following the stages below.

1. We get a w-matching machine Γ which simulates the behavior of the algorithm while looking for w (Figure 1). The transformation of the 9 algorithms presented in Section 7 (and a few others, see our GitHub repository) into w-matching machines, given w, has been implemented.

2. We obtain the full-memory extension $\Gamma^\star$ of Γ (Figure 2, Section 2.3).

3. We compute the limit frequencies of the Markov model associated to $\Gamma^\star$ and π in Theorem 1. This mainly needs to solve a system of linear equations of dimension $|Q^\star|$.

4. We finally obtain the asymptotic speed of the algorithm from these limit frequencies, π and $\Gamma^\star$ by using Equation 1.

The most time-consuming stage is the computation of the limit frequencies, which has $O(|Q^\star|^3)$ time complexity, where $|Q^\star|$, the number of states of the full memory expansion, is smaller than $(\mathcal{A}+1)^{|w|}|Q|$.

3 Strategies

For all sets $\mathcal{I} \subset \mathbb{N}$ and $k \in \mathbb{N}$, we define the k-left-shifted of $\mathcal{I}$ as

$$\Delta(\mathcal{I}, k) = \{i - k \mid \exists i \in \mathcal{I} \text{ and } i \geq k\}.$$

A *w-strategy* $S = (Q, o, F, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\gamma})$ is a w-matching machine such that

- $Q \subseteq \mathcal{P}(\{0, \dots, |w| - 1\}) \setminus \{\{0, \dots, |w| - 1\}\}$ and $\emptyset \in Q$,
- $o = \emptyset$,
- $F = \{s \in Q \mid |s| = |w| - 1\}$,
- $\boldsymbol{\alpha} : Q \to \{0, 1, \dots, |w| - 1\}$ is such that for all $s \in Q$, $\boldsymbol{\alpha}(s) \notin s$ and $|s| < |w| - 1 \Rightarrow s \cup \{\boldsymbol{\alpha}(s)\} \in Q$,
- $\boldsymbol{\gamma} : Q \times \mathcal{A} \to \{0, 1, \dots, |w|\}$ is such that for all states s and all symbols x,

$$\boldsymbol{\gamma}(q, x) \\ = \begin{cases} \min\{k \geq 1 \mid w_{\boldsymbol{\alpha}(s)-k} = x \text{ if } \boldsymbol{\alpha}(s) \geq k \text{ and } w_j = w_{j+k} \text{ for all } j \in \Delta(s, k)\} & \text{if } q \in F, \\ \min\{k \geq 0 \mid w_{\boldsymbol{\alpha}(s)-k} = x \text{ if } \boldsymbol{\alpha}(s) \geq k \text{ and } w_j = w_{j+k} \text{ for all } j \in \Delta(s, k)\} & \text{otherwise,} \end{cases}$$

- $\boldsymbol{\delta} : Q \times \mathcal{A} \to Q$ is such that for all $q \in Q$ and all symbols x,

$$\boldsymbol{\delta}(s, x) = \Delta(s \cup \{i\}, \boldsymbol{\gamma}(s, x)).$$

Proposition 1. *A w-strategy is a standard, valid and non-redundant w-matching machine.*

Proof. By construction, a w-strategy is both standard and non redundant. The validity of a w-strategy follows from Theorem 1 of [8]. □

Proposition 2. *There is a w-strategy which achieves the greatest asymptotic speed among all the w-matching machines of order $|w| - 1$.*

Proof. The Corollary 2 of [8] implies that there exists a w-matching machine which achieves the greatest asymptotic speed among those of order of order $|w| - 1$ and which is

1. standard,
2. compact,
3. valid,
4. in which all the states are relevant (i.e. such that they may lead to a match without any positive shift [8]),
5. such that there is no pair of states (q, q') with $q \neq q'$ and $\mathbf{h}_{\Gamma_\pi}(q) = \mathbf{h}_{\Gamma_\pi}(q')$.

Let us verify that a w-matching machine $\Gamma = (Q, o, F, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\gamma})$ of order $|w| - 1$ satisfying the properties above is (isomorphic to) a w-strategy. Since it verifies in particular the properties 4 and 5, its set of states Q is in bijection with a subset of $\mathcal{P}(\{0, \dots, |w| - 1\})$. Let us identify all states q of Q with $\mathbf{f}(\mathbf{h}_\Gamma(q))$, its corresponding element of $\mathcal{P}(\{0, \dots, |w| - 1\})$. Since Γ is standard, compact and of order $|w| - 1$, we don't have $\{0, \dots, |w| - 1\} \in Q$. Moreover, since Γ is standard, we have $\boldsymbol{\delta}(s, x) = \Delta(s \cup \{i\}, \boldsymbol{\gamma}(s, x))$. Last, by construction, if

$$\begin{aligned}&\boldsymbol{\gamma}(q, x)\\ &> \begin{cases} \min\{k \geq 1 \mid w_{\boldsymbol{\alpha}(s)-k} = x \text{ if } \boldsymbol{\alpha}(s) \geq k \text{ and } w_j = w_{j+k} \text{ for all } j \in \Delta(s,k)\} & \text{if } q \in F,\\ \min\{k \geq 0 \mid w_{\boldsymbol{\alpha}(s)-k} = x \text{ if } \boldsymbol{\alpha}(s) \geq k \text{ and } w_j = w_{j+k} \text{ for all } j \in \Delta(s,k)\} & \text{otherwise,} \end{cases}\end{aligned}$$

then Γ is not valid, and if

$$\begin{aligned}&\boldsymbol{\gamma}(q, x)\\ &< \begin{cases} \min\{k \geq 1 \mid w_{\boldsymbol{\alpha}(s)-k} = x \text{ if } \boldsymbol{\alpha}(s) \geq k \text{ and } w_j = w_{j+k} \text{ for all } j \in \Delta(s,k)\} & \text{if } q \in F,\\ \min\{k \geq 0 \mid w_{\boldsymbol{\alpha}(s)-k} = x \text{ if } \boldsymbol{\alpha}(s) \geq k \text{ and } w_j = w_{j+k} \text{ for all } j \in \Delta(s,k)\} & \text{otherwise,} \end{cases}\end{aligned}$$

then $\boldsymbol{\delta}(q, x)$ is not relevant. □

4 Position lattices

The position lattice of a pattern w is the 3-uple $L^{[w]} = (Q^{[w]}, (\delta_s^{[w]})_{s \in Q^{[w]}}, (\gamma_s^{[w]})_{s \in Q^{[w]}})$ where, by putting $\overline{s}$ for $\{0, \dots, |w| - 1\} \setminus s$,

- $Q^{[w]} = \mathcal{P}(\{0, \dots, |w| - 1\}) \setminus \{\{0, \dots, |w| - 1\}\}$, i.e. the set made of all the subsets of positions of w but $\{0, \dots, |w| - 1\}$,
- for all $s \in Q^{[w]}$, $\gamma_s^{[w]}$ is a map from $\overline{s} \times \mathcal{A}$ to $\{0, \dots, |w|\}$,
- for all $s \in Q^{[w]}$, $\delta_s^{[w]}$ is a map from $\overline{s} \times \mathcal{A}$ to $Q^{[w]}$,

where, for all $s \in Q^{[w]}$, all $i \in \overline{s}$ and all $x \in \mathcal{A}$, we have

$$\begin{aligned}&\gamma_s^{[w]}(i, x)\\ &= \begin{cases} \min\{k \geq 1 \mid w_{\boldsymbol{\alpha}(s)-k} = x \text{ if } \boldsymbol{\alpha}(s) \geq k \text{ and } w_j = w_{j+k} \text{ for all } j \in \Delta(s,k)\} & \text{if } |s| = |w| - 1,\\ \min\{k \geq 0 \mid w_{\boldsymbol{\alpha}(s)-k} = x \text{ if } \boldsymbol{\alpha}(s) \geq k \text{ and } w_j = w_{j+k} \text{ for all } j \in \Delta(s,k)\} & \text{otherwise,} \end{cases}\end{aligned}$$

and

$$\delta_s^{[w]}(i, x) = \Delta(s \cup \{i\}, \gamma_s^{[w]}(i, x)).$$

In particular, if $x = w_i$ and $|s| < |w| - 1$ then we have $\gamma_s^{[w]}(i, x) = 0$ and $\delta_s^{[w]}(i, x) = s \cup \{i\}$.

Let us remark that, since $\max(s) \leq |w| - 1$ for all $s \in Q^{[w]}$, we have, for all $i \in \overline{s}$ and all $x \in \mathcal{A}$, $\Delta(s \cup \{i\}, |w|) = \emptyset$, thus $\gamma_s^{[w]}(i, x) \leq |w|$ which is consistent with the definition of $\gamma_s^{[w]}$.

The edges of $L^{[w]}$ are the pairs $(s, \delta_s^{[w]}(i, x))$ for all $s \in Q^{[w]}$, all $i \in \overline{s}$ and all $x \in \mathcal{A}$ (see Figure 3).

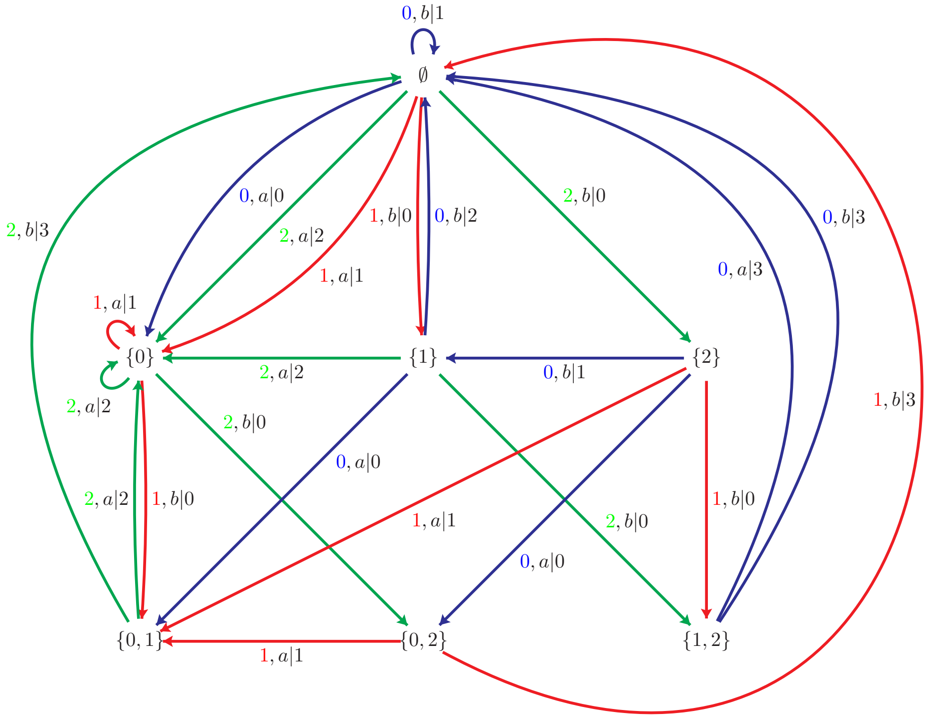

Figure 3: Position lattice of the pattern abb. Vertices represent the states of $L^{[abb]}$. For all states s, there is an outgoing edge for all pairs (i, x) with $i \in \{0, \ldots, |abb| - 1\} \setminus s$ and $x \in \mathcal{A}$. This outgoing edge is labeled with "$i, x|\gamma_s^{[abb]}(i, x)$", is colored according to i, and goes to $\delta_s^{[abb]}(i, x)$.

Remark 1. *The position lattice of w contains $2^{|w|} - 1$ states and $|\mathcal{A}|.|w|.2^{|w|-1}$ edges.*

Remark 2. *Let s be a state of $Q^{[w]}$, i and j be two positions in $\overline{s}$ such that $i \neq j$ and x and y be two symbols of $\mathcal{A}$. We have*

$$\gamma^{[w]}_{\delta^{[w]}_s(i,x)}(j-\gamma^{[w]}_s(i,x),y)+\gamma^{[w]}_s(i,x)=\gamma^{[w]}_{\delta^{[w]}_s(j,y)}(i-\gamma^{[w]}_s(j,y),x)+\gamma^{[w]}_s(j,y), \quad \textit{and}$$
$$\delta^{[w]}_{\delta^{[w]}_s(i,x)}(j-\gamma^{[w]}_s(i,x),y)=\delta^{[w]}_{\delta^{[w]}_s(j,y)}(i-\gamma^{[w]}_s(j,y),x).$$

By considering the particular case where $x = w_i$, we get

$$\gamma^{[w]}_{s\cup\{i\}}(j,y)=\gamma^{[w]}_{\delta^{[w]}_s(j,y)}(i-\gamma^{[w]}_s(j,y),w_i)+\gamma^{[w]}_s(j,y), \quad \textit{and}$$
$$\delta^{[w]}_{s\cup\{i\}}(j,y)=\delta^{[w]}_{\delta^{[w]}_s(j,y)}(i-\gamma^{[w]}_s(j,y),w_i).$$

Let prec_w be the table indexed on $\{0,\dots,|w|-1\}\times\mathcal{A}$ and in which, for all positions i of w and all symbols x of $\mathcal{A}$, the entry $\mathrm{prec}_w[i,x]$ is defined as

$$\mathrm{prec}_w[i,x]=\begin{cases}\max\{j\le i \mid w_j=x\} & \text{if } \{j\le i \mid w_j=x\}\neq\emptyset,\\ \mathrm{NULL} & \text{otherwise.}\end{cases}$$

Lemma 1. *Let s be a state of $Q^{[w]}$, i a position in $\overline{s}$ and x a symbol of $\mathcal{A}$.*

1. *If $x = w_i$ then*
   - *if $|s| = |w|-1$ then $\delta^{[w]}_s(i,x)=\{0,\dots,\mathrm{B}-1\}$ and $\gamma^{[w]}_s(i,x)=|w|-\mathrm{B}$, where B is the length of the longest proper suffix of w which is a prefix of w;*
   - *otherwise $\delta^{[w]}_s(i,x)=s\cup\{i\}$ and $\gamma^{[w]}_s(i,x)=0$.*
2. *If $x \neq w_i$,*

   (a) if $s=\emptyset$ then

$$\delta^{[w]}_\emptyset(i,x)=\begin{cases}\{\mathrm{prec}_w[i,x]\} & \textit{if } \mathrm{prec}_w[i,x]\neq\mathrm{NULL},\\ \emptyset & \textit{otherwise,}\end{cases}$$
$$\gamma^{[w]}_\emptyset(i,x)=\begin{cases}i-\mathrm{prec}_w[i,x] & \textit{if } \mathrm{prec}_w[i,x]\neq\mathrm{NULL},\\ i+1 & \textit{otherwise,}\end{cases}$$

   (b) if $s\neq\emptyset$ then for all $\ell\in s$, we have

$$\delta^{[w]}_s(i,x)=\delta^{[w]}_{\delta^{[w]}_{s\setminus\{\ell\}}(i,x)}(\ell-\gamma^{[w]}_{s\setminus\{\ell\}}(i,x),w_\ell),$$
$$\gamma^{[w]}_s(i,x)=\gamma^{[w]}_{\delta^{[w]}_{s\setminus\{\ell\}}(i,x)}(\ell-\gamma^{[w]}_{s\setminus\{\ell\}}(i,x),w_\ell)+\gamma^{[w]}_{s\setminus\{\ell\}}(i,x).$$

Proof. The only case which does not immediately follow from the definition of $L^{[w]}$, is when $x \neq w_i$ and $s \neq \emptyset$ which is given by Remark 2. □

The relation $\preccurlyeq$ on $Q^{[w]}$ is defined as follows. For all sets s and s' in $Q^{[w]}$, we have $s \preccurlyeq s'$ if one of the following properties holds:

- $|s| < |s'|$,
- $|s| = |s'|$, $s \neq s'$ and $\min(s \ominus s') \in s$, where $s \ominus s'$ is the symmetric difference of s and s',
- $s = s'$.

The relation $\preccurlyeq$ defines a total order on $Q^{[w]}$. We write " $s \prec s'$ " for " $s \preccurlyeq s'$ and $s \neq s'$ ".

Lemma 2. *Let s be a state of $Q^{[w]}$ with $|s| > 1$, i a position in $\overline{s}$ and x a symbol of $\mathcal{A}$. If $x \neq w_i$ then $\delta^{[w]}_{s\setminus\{\max s\}}(i, x) \prec s$.*

Proof. Under the assumption that $|s| > 1$, we have $\min s = \min(s \setminus \{\max s\})$. By construction, the fact that $x \neq w_i$ implies that $\gamma^{[w]}_{s\setminus\{\max s\}}(i, x) > 0$.

If we have
$$\min((s \setminus \{\max s\}) \cup \{i\}) < \gamma^{[w]}_{s\setminus\{\max s\}}(i, x)$$
then $|\delta^{[w]}_{s\setminus\{\max s\}}(i, x)| < |s|$, thus $\delta^{[w]}_{s\setminus\{\max s\}}(i, x) \prec s$.

Otherwise, we have $|\delta^{[w]}_{s\setminus\{\max s\}}(i, x)| = |s|$ but since necessarily
$$\min \delta^{[w]}_{s\setminus\{\max s\}}(i, x) \leq \min|s| - \gamma^{[w]}_{s\setminus\{\max s\}}(i, x) < \min|s|,$$
we get again $\delta^{[w]}_{s\setminus\{\max s\}}(i, x) \prec s$. □

Theorem 2. *Algorithm 2 computes the position lattice of the pattern w in $O(|w|2^{|w|})$ time by using the same amount of memory.*

Proof. Let us first show that Algorithm 2 determines the shifts and the transitions of the state s before those of the state s' if and only if $s \prec s'$. The loop at Lines 3-9 computes the shifts and the transitions of $\emptyset$. Next, the loop at Lines 10-24 computes the shifts and the transitions of the singletons from $\{0\}$ to $\{|w| - 1\}$. The last loop (Lines 25-43) determines the shifts and the transitions of the states corresponding to the subsets of increasing cardinals ℓ from 2 to $|w| - 1$. Inside the last loop, the way in which the next subset s' is computed from the current subset s, both of cardinal ℓ, ensures that $s \prec s'$ (Lines 39-43).

For all iterations i of the loop at Lines 3-9 and all symbols x, we have $\mathrm{last}[x] = \mathrm{prec}_w[i, x]$ at the beginning of the inner loop (Line 5). From Lemma 1 (Cases 1 and 2a), the transitions $\delta^{[w]}_{\emptyset}(i, x)$ and the shifts $\gamma^{[w]}_{\emptyset}(i, x)$ for all positions i of w and all symbols x, are correctly computed at the end of the loop.

The loop at Lines 10-24 computes the shifts and the transitions from the singleton states. For all pairs of positions (i, j) and all symbols x, determining $\delta^{[w]}_{\{i\}}(j, x)$ and $\gamma^{[w]}_{\{i\}}(j, x)$ is performed by distinguishing between two cases.

```
$\mathrm{B} \leftarrow$ length of the longest proper suffix of $w$ which is also a prefix;
for $x \in \mathcal{A}$ do  $\mathrm{last}[x] \leftarrow \mathrm{NULL}$ ;
for $i = 0$ to $|w| - 1$ do
    $\mathrm{last}[w_i] \leftarrow i$;
    for $x \in \mathcal{A}$ do
        if $\mathrm{last}[x] \neq \mathrm{NULL}$ then
            $\gamma_{\emptyset}^{[w]}(i,x) \leftarrow i - \mathrm{last}[x]$;  $\delta_{\emptyset}^{[w]}(i,x) \leftarrow \{\mathrm{last}[x]\}$;
        else
            $\gamma_{\emptyset}^{[w]}(i,x) \leftarrow i + 1$;  $\delta_{\emptyset}^{[w]}(i,x) \leftarrow \emptyset$;
for $i = 0$ to $|w| - 1$ do
    for $j = 0$ to $i - 1$ do
        for $x \in \mathcal{A}$ do
            $\gamma_{\{i\}}^{[w]}(j,x) \leftarrow \gamma_{\emptyset}^{[w]}(j,x) + \gamma^{[w]}_{\delta_{\emptyset}^{[w]}(j,x)}(i - \gamma_{\emptyset}^{[w]}(j,x), w_i)$;
            $\delta_{\{i\}}^{[w]}(j,x) \leftarrow \delta^{[w]}_{\delta_{\emptyset}^{[w]}(j,x)}(i - \gamma_{\emptyset}^{[w]}(j,x), w_i)$;
    for $j = i + 1$ to $|w| - 1$ do
        for $x \in \mathcal{A}$ do
            if $x = w_j$ then
                if $|w| = 2$ then
                    $\gamma_{s}^{[w]}(i,x) \leftarrow |w| - \mathrm{B}$;  $\delta_{s}^{[w]}(i,x) \leftarrow \{0, \dots, \mathrm{B} - 1\}$;
                else
                    $\gamma_{\{i\}}^{[w]}(j,x) \leftarrow 0$;  $\delta_{\{i\}}^{[w]}(j,x) \leftarrow \{i, j\}$;
            else
                $\gamma_{\{i\}}^{[w]}(j,x) \leftarrow \gamma^{[w]}_{\delta_{\emptyset}^{[w]}(i-1,w_i)}(j - \gamma_{\emptyset}^{[w]}(i-1,w_i), x)$;
                $\delta_{\{i\}}^{[w]}(j,x) \leftarrow \delta^{[w]}_{\delta_{\emptyset}^{[w]}(i-1,w_i)}(j - \gamma_{\emptyset}^{[w]}(i-1,w_i), x)$;
for $\ell = 2$ to $|w| - 1$ do
    for $j = 0$ to $\ell - 1$ do  $\mathrm{S}[j] \leftarrow j$ ;
    repeat
        $s \leftarrow \{\mathrm{S}[0], \dots, \mathrm{S}[\ell - 1]\}$;  $s' \leftarrow \{\mathrm{S}[0], \dots, \mathrm{S}[\ell - 2]\}$;
        for $i \in \{0, \dots, |w| - 1\} \setminus s$ do
            for $x \in \mathcal{A}$ do
                if $x = w_i$ then
                    if $\ell = |w| - 1$ then
                        $\gamma_{s}^{[w]}(i,x) \leftarrow |w| - \mathrm{B}$;  $\delta_{s}^{[w]}(i,x) \leftarrow \{0, \dots, \mathrm{B} - 1\}$;
                    else
                        $\gamma_{s}^{[w]}(i,x) \leftarrow 0$;  $\delta_{s}^{[w]}(i,x) \leftarrow s \cup \{i\}$;
                else
                    $\gamma_{s}^{[w]}(i,x) \leftarrow \gamma_{s'}^{[w]}(i,x) + \gamma^{[w]}_{\delta_{s'}^{[w]}(i,x)}(\mathrm{S}[\ell-1] - \gamma_{s'}^{[w]}(i,x), w_{\mathrm{S}[\ell-1]})$;
                    $\delta_{s}^{[w]}(i,x) \leftarrow \delta^{[w]}_{\delta_{s'}^{[w]}(i,x)}(\mathrm{S}[\ell - 1] - \gamma_{s'}^{[w]}(i,x), w_{\mathrm{S}[\ell-1]})$;
        $j \leftarrow \ell - 1$;
        while $j \geq 0$ and $\mathrm{S}[j] \geq |w| - \ell + j$ do  $j \leftarrow j - 1$ ;
        if $j \geq 0$ then
            $\mathrm{S}[j] \leftarrow \mathrm{S}[j] + 1$;
            for $k = j + 1$ to $\ell - 1$ do  $\mathrm{S}[k] \leftarrow \mathrm{S}[k - 1] + 1$ ;
    until $j < 0$;
```

Algorithm 2: Computation of the position lattice.

- If $i > j$, then $\delta_{\emptyset}^{[w]}(j,x) \prec \{i\}$ and its shifts and transitions were already computed. Formula of Remark 2 gives us those of $\{i\}$ (Lines 13-14).
- If $i < j$, we distinguish between two subcases according to the symbol x considered. If $x = w_j$ then the shift and the transition state are given in Lemma 1 - Case 1. Otherwise, we remark that, since $\gamma_{\{i\}}^{[w]}(j,x)$ is positive, we have that $\gamma_{\{i\}}^{[w]}(j,x) = \min\{k \geq 1 \mid w_{j-k} = x \text{ if } j \geq k \text{ and } w_{i-k} = w_i\}$. This implies that $\gamma_{\{i\}}^{[w]}(j,x) = \gamma_{\delta_{\emptyset}^{[w]}(i-1,w_i)}^{[w]}(j,x)$. We have $\delta_{\emptyset}^{[w]}(i-1,w_i) \prec s$, thus both the shifts and the transitions of the state $\delta_{\emptyset}^{[w]}(i-1,w_i)$ are computed before s (Lines 23-24).

The last loop, lines 25-43, computes the shifts and the transitions of the states corresponding to the subsets of cardinals 2 to $\ell - 1$. For all states s with $2 \leq |s| \leq |w| - 1$, all positions $i \in \overline{s}$ and all symbols $x \neq w_i$, the corresponding shift and transition $\gamma_s^{[w]}(i,x)$ and $\delta_s^{[w]}(i,x)$ are computed from the shifts and transitions of the state $\delta_{s\setminus\{\max s\}}^{[w]}(i,x)$ following Lemma 1 - Cases 2b (in Algorithm 2, we put s' for $s \setminus \{\max s\}$). Lemma 2 ensures that $\delta_{s\setminus\{\max s\}}^{[w]}(i,x) \prec s$, thus that the shifts and transitions of $\delta_{s\setminus\{\max s\}}^{[w]}(i,x)$ are computed before those of s. For all states s with $2 \leq |s| \leq |w| - 1$, all positions $i \in \overline{s}$, the shift and transition $\gamma_s^{[w]}(i,w_i)$ and $\delta_s^{[w]}(i,w_i)$ are given in Lemma 1 - Case 1.

The time complexity is $O(\sum_{k=0}^{|w|-1}(k + |w| - k)\binom{|w|}{k})$ (loop Lines 25-43), i.e. $O(|w|2^{|w|})$. We do not use more memory than needed to store the lattice, which is, from Remark 1, $O(|w|2^{|w|})$. □

5 The Fastest w-strategy

Determining the fastest w-strategy, which, from Proposition 2, has the greatest asymptotic speed among all the w-matching machines of order $|w| - 1$, may be performed by computing the asymptotic speed of all the w-strategies and by returning the fastest one.

In order to enumerate all the w-strategies, let us remark that they are all contained in the position lattice of w in the sense that:

- the set of states of a w-strategy is included in that of the position lattice;
- all the w-strategies $\Gamma = (Q, o, F, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\gamma})$ verify $\boldsymbol{\delta}(s,x) = \delta_s^{[w]}(\boldsymbol{\alpha}(s),x)$ and $\boldsymbol{\gamma}(s,x) = \gamma_s^{[w]}(\boldsymbol{\alpha}(s),x)$ for all $s \in Q$ and all symbols x.

Reciprocally, to any map ϕ from $Q^{[w]}$ to $\{0,\ldots,|w|-1\}$ such that $\phi(s) \in \overline{s}$ for all states $s \in Q^{[w]}$, there corresponds the unique w-strategy $S = (Q, o, F, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\gamma})$ for which the next-position-to-check function $\boldsymbol{\alpha}$ coincides with ϕ on Q.

Finally, our brute force algorithm

1. takes as input a pattern w and an iid model π,

2. computes the position lattice of w,

3. enumerates all the maps ϕ such that $\phi(s) \in \overline{s}$ for all states $s \in Q^{[w]}$,

4. for each ϕ, gets the corresponding w-strategy by keeping only the states of $Q^{[w]}$ reachable from $\emptyset$, with the next-position-to-check function ϕ,

5. computes the asymptotic speed of all the w-strategies under π,

6. returns the w-strategy with the greatest speed.

The time complexity of the brute force algorithm is

$$O\left(\left(\prod_{k=1}^{|w|-1} k^{\binom{|w|}{k}}\right) 2^{3|w|}\right),$$

where the first factor stands for the number of functions ϕ and the second one for the computation of the asymptotic speed of a w-strategy, which needs to solve a linear system of size equal to the number of states, which is $O(2^{|w|})$. Its memory space complexity is $|w|2^{|w|-1}$, i.e. what is needed to store the position lattice of w.

Under its current implementation, the brute force determination of the fastest w-strategy is unfeasible for patterns of length greater than 4.

6 A polynomial heuristic

There are two points which make the complexity of the brute force algorithm that high:

1. the size of the position lattice, which is exponential with the length of the pattern,

2. determining the fastest strategy in the position lattice, which needs a time exponential with its size.

Our heuristic is based on two independent stages, each one aiming to overcome one of these two points. Both of them start from the general idea that, since, for any current position of the text, the probability that no mismatch occurs until the n^{th} text access decreases geometrically with n, the first relative positions accessed by a strategy (or more generally by a pattern algorithm) are those which have the greatest influence on its asymptotic speed.

6.1 n-sets sublattices

A sufficient condition for a sublattice $\mathcal{U} \subseteq Q^{[w]}$ to contain a w-strategy is that, for all $s \in \mathcal{U}$, there exists at least a position $i \in \overline{s}$ with $\delta_s^{[w]}(i, x) \in \mathcal{U}$ for all $x \in \mathcal{A}$. A sublattice $\mathcal{U}$ verifying this condition will be said to be *complete*.

Let us introduce some additional notations here. For all sets $\mathcal{S}$ of positions, the *prefix* of $\mathcal{S}$ is defined as $P(\mathcal{S}) = \max\{i \in \mathcal{S} \mid j \in \mathcal{S} \text{ for all } 0 \leq j \leq i\}$ and its *rest* is $R(\mathcal{S}) = \mathcal{S} \setminus \{0, \dots, P(\mathcal{S})\}$

For all positive integers n, the n-sets sublattice of w is the sublattice $\mathcal{U}$ of $Q^{[w]}$ which contains all and only the subsets of $Q^{[w]}$ with a rest containing less than n positions, i.e. the subsets of the form $\{0, \dots, p\} \cup \mathcal{X}$ with $p < |w| - 1$ and $|\mathcal{X}| \leq n$.

By construction, the n-sets sublattice of w is complete. It contains $O(|w|^n)$ states and $O(|w|^{n+1})$ transitions.

We adapted Algorithm 2 to compute the n-sets sublattice of w in $O(|w|^{n+1})$ time with the same amount of memory space.

6.2 ℓ-shift expectation

We are now interested in a fast way for finding an efficient w-strategy in a given complete sublattice.

For all integers ℓ and all states s of a sublattice $\mathcal{U}$, the ℓ-shift expectation of s is defined as the greatest shift expectation one could possibly get in ℓ steps in $\mathcal{U}$ by starting from s. Namely, the ℓ-shift expectation is computed following the recursive formula:

- $\mathrm{ES}_0^{[w]}[s] = 0$,
- for all $k > 0$,

$$\mathrm{ES}_k^{[w]}[s] = \max_{i \in \mathrm{Tr}(s)} \sum_{x \in \mathcal{A}} \left(\gamma_s^{[w]}(i, x) + \mathrm{ES}_{\ell-1}^{[w]}[\delta_s^{[w]}(i, x)] \right)$$

where $\mathrm{Tr}(s) = \{i \in \overline{s} \mid \delta_s^{[w]}(i, x) \in \mathcal{U} \text{ for all } x \in \mathcal{A}\}$.

The ℓ-shift expectation of a complete sublattice $\mathcal{U}$ is well defined and can be computed in $O(\ell T)$ time, where T is the number of transitions of the sublattice $\mathcal{U}$ and by using $O(|\mathcal{U}|)$ memory space.

We finally extract a w-strategy from $\mathcal{U}$ by setting the next-position-to-check of all states $s \in \mathcal{U}$ to

$$\operatorname*{arg\,max}_{i \in \mathrm{Tr}(s)} \sum_{x \in \mathcal{A}} \left(\gamma_s^{[w]}(i, x) + \mathrm{ES}_{\ell-1}^{[w]}[\delta_s^{[w]}(i, x)] \right).$$

6.3 K-Heuristic

The K-Heuristic combines the two approaches above in order to compute a w-strategy in a time polynomial with the length of the pattern.

Being given an order $K \geq 1$, we start by computing the K-sets sublattice of w, thus in $O(|w|^{K+1})$ time. In order to select a w-strategy from the K-sets sublattice, we next compute the $(K+1)$-shift expectation of all its states and extract a w-strategy as described just above. This computation is performed in

$O(K|w|^{K+1})$ time, since the number of transition of the sublattice is $O(|w|^{K+1})$, by using $O(|w|^K)$ memory space.

Let us remark that the order ℓ of the ℓ-shift expectation does not have, a priori, to be strongly related to the order K of the K-sets sublattice on which it is computed. By experimenting various situations, we observed that considering an order greater than $K+1$ generally does not improve much the performances, whereas the strategies obtained from ℓ-expectations with ℓ smaller than K may be significantly slower.

The K-Heuristic returns a w-strategy in $O(K|w|^{K+1})$ time by using $O(|w|^K)$ memory space. We insist on the fact that the K-Heuristic generally does not return the fastest strategy, even if $K > |w|$. However, we will see in the next section that it performs quite well in practice.

7 Evaluation

We shall compare the approaches introduced in Sections 5 and 6 with selected pattern matching algorithms. The comparison is performed, first, from a theoretical point of view, by computing their asymptotic speeds under iid models, and second, in practical situations, by measuring their average speed over real data. The *average speed* with regard to a pattern w, of an algorithm or a matching machine on a text t is the ratio of $|t|$ to the number of text accesses performed by the algorithm to search w in t.

We are also interested in to what extent taking into account the frequencies of the letters of an iid model or a text, for determining the Fastest and the K-Heuristic strategies, actually improves their asymptotic or their average speeds. To this purpose, we compute the Fastest and the K-Heuristic strategies from the uniform iid model. Next, we test their efficiency in terms of asymptotic speed under a non-uniform iid model and in terms of average speeds on data with non-uniform frequencies of letters.

7.1 Pre-existing pattern matching algorithms

More than forty years of research have already led to the development of dozens algorithms. We selected the 9 ones below for our evaluation:

1. Naive [6],
2. Morris-Pratt [6],
3. Knuth-Morris-Pratt [15],
4. Quicksearch [23],
5. Boyer-Moore-Horspool [13],
6. TVSBS [24], a“right-to-left” algorithm in which shifts are given by a bad-character rule [5, 23] taking into account the two letters at distances $|w|-1$ and $|w|$ from the current position of the text,

7. EBOM [9], a version of the Backward Oracle Matching algorithm [1] which also uses a “bad two-characters” rule,

8. HASHq [26], which implements the Boyer-Moore algorithm on blocks of length q by using efficient hashing techniques [14] (our tests are performed with $q = 3$),

9. FJS [11], which combines the ideas of Knuth-Morris-Pratt [15] and Sunday [23] algorithms.

Algorithms 1 to 5 are classics. The last four ones were chosen for being known to be efficient on short patterns and small alphabets [10], a situation in which the determination of the fastest strategy is feasible.

Let us remark that the order of the w-matching machine associated to TVSBS is equal to $|w|$, thus greater than that of the Fastest strategy that we compute.

The transformation into matching machines was implemented for a few other pattern matching approaches, for instance the SA algorithm (the Baeza-Yates-Gonnet algorithm) based on bitwise operations [2], or the string-matching automaton [7]. Since the asymptotic and average speeds of these two algorithms are exactly 1, whatever the pattern, the model and the text, there is no point in displaying them.

7.2 Results

We shall evaluate:

- the pre-existing pattern matching algorithms presented in Section 7.1,
- the 1- 2- and 3-Heuristics and
- the Fastest strategy (each time it is possible).

7.2.1 Asymptotic speed

The asymptotic speeds are computed for texts and patterns on the binary alphabet $\{\mathtt{a}, \mathtt{b}\}$.

Table 1 displays the asymptotic speeds for all the patterns of length 4 on iid texts drawn from the uniform distribution. As expected, the strategy computed with the brute force algorithm (last column) is actually the fastest, but the speeds of the 1-,2- and 3-Heuristics are very close. The pre-existing algorithms are outperformed by all our approaches (even by the 1-Heuristic) for all the patterns. We observe that the Naive, Morris-Pratt and Knuth-Morris-Pratt algorithms have asymptotic speeds always smaller than 1. One cannot expect them to be faster since, by construction, they access all the positions of a text at least once. In the following, we will not display their speeds, nor that of Quicksearch, for they are always smaller than at least one of the other pre-existing algorithms. The full tables can easily be re-computed by using our software.

	Naive	Morris-Pratt	Knuth-Morris-Pratt	Quicksearch	Horspool	FJS	TVSBS	EBOM	Hashq	1-Heuristic	2-Heuristic	3-Heuristic	Fastest
aaaa	0.53	0.70	1.00	0.98	1.18	0.78	0.72	0.93	0.52	1.50	1.69	1.80	**1.83**
aaab	0.53	0.76	0.94	0.51	1.18	0.69	0.39	0.73	0.52	1.37	1.52	**1.60**	**1.60**
aaba	0.53	0.76	0.89	0.51	0.73	0.59	0.54	0.62	0.53	1.19	1.33	1.35	**1.37**
aabb	0.53	0.76	0.84	0.69	0.73	0.71	0.50	0.69	0.53	1.30	1.43	1.54	**1.56**
abaa	0.53	0.73	0.80	0.63	0.73	0.66	0.61	0.62	0.54	1.23	1.34	**1.38**	**1.38**
abab	0.53	0.70	0.80	0.50	0.73	0.56	0.50	0.73	0.54	1.22	1.33	1.36	**1.43**
abba	0.53	0.70	0.73	0.55	0.94	0.56	0.39	0.67	0.53	1.27	1.31	**1.34**	**1.34**
abbb	0.53	0.70	0.70	0.75	0.94	0.76	0.62	0.73	0.53	1.47	1.59	1.64	**1.69**
baaa	0.53	0.70	0.70	0.75	0.94	0.76	0.62	0.73	0.53	1.47	1.59	1.64	**1.69**
baab	0.53	0.70	0.73	0.55	0.94	0.56	0.39	0.67	0.53	1.27	1.31	**1.34**	**1.34**
baba	0.53	0.70	0.80	0.50	0.73	0.56	0.50	0.73	0.54	1.22	1.33	1.36	**1.43**
babb	0.53	0.73	0.80	0.63	0.73	0.66	0.61	0.62	0.54	1.23	1.34	**1.38**	**1.38**
bbaa	0.53	0.76	0.84	0.69	0.73	0.71	0.50	0.69	0.53	1.30	1.43	1.54	**1.56**
bbab	0.53	0.76	0.89	0.51	0.73	0.59	0.54	0.62	0.53	1.19	1.33	1.35	**1.37**
bbba	0.53	0.76	0.94	0.51	1.18	0.69	0.39	0.73	0.52	1.37	1.52	**1.60**	**1.60**
bbbb	0.53	0.70	1.00	0.98	1.18	0.78	0.72	0.93	0.52	1.50	1.69	1.80	**1.83**

Table 1: Asymptotic speeds for the patterns of length 4 on $\{\mathtt{a}, \mathtt{b}\}$ under the uniform model.

	Horspool	FJS	TVSBS	EBOM	Hashq	Unif. 1-Heuristic	Unif. 2-Heuristic	Unif. 3-Heuristic	Unif. Fastest	1-Heuristic	2-Heuristic	3-Heuristic	Fastest
aaaa	3.30	1.97	1.68	1.49	0.67	3.02	3.46	**3.50**	**3.50**	3.02	3.47	**3.50**	**3.50**
aaab	1.77	0.53	0.27	1.37	0.66	2.43	2.55	2.55	2.55	2.43	2.60	2.60	**2.61**
aaba	0.91	0.87	1.55	1.24	0.66	1.77	2.14	2.14	2.14	1.77	**2.19**	**2.19**	**2.19**
aabb	0.38	0.53	0.29	0.70	0.63	1.34	1.71	1.77	1.77	1.74	1.79	**1.80**	**1.80**
abaa	1.67	1.24	1.63	1.24	0.66	1.80	2.14	2.17	2.16	1.80	2.15	**2.18**	**2.18**
abab	0.85	0.54	0.28	1.17	0.63	1.29	1.78	1.68	1.64	1.42	1.80	1.80	**1.81**
abba	0.93	0.87	0.82	0.63	0.61	1.06	1.31	1.51	1.54	1.30	1.73	**1.80**	**1.80**
abbb	0.33	0.57	0.31	0.31	0.46	1.08	1.12	1.14	**1.15**	1.08	1.10	1.14	**1.15**
baaa	2.50	1.49	1.33	1.37	0.66	2.44	2.60	**2.61**	**2.61**	2.44	2.60	**2.61**	**2.61**
baab	1.34	0.37	0.22	1.18	0.65	**1.75**	**1.75**	**1.75**	**1.75**	**1.75**	**1.75**	**1.75**	**1.75**
baba	0.91	0.80	1.16	1.17	0.63	1.09	1.35	1.44	1.74	1.09	1.78	**1.84**	**1.84**
babb	0.38	0.39	0.21	0.55	0.54	1.03	0.91	1.04	**1.05**	1.04	1.04	1.04	**1.05**
bbaa	1.67	1.10	1.06	0.70	0.63	1.09	1.72	**1.84**	1.83	1.09	1.72	**1.84**	**1.84**
bbab	0.85	0.31	0.23	0.55	0.54	1.00	0.83	1.06	**1.08**	1.01	**1.08**	**1.08**	**1.08**
bbba	1.00	0.90	0.73	0.31	0.36	1.02	1.09	1.17	1.17	1.08	1.16	**1.24**	**1.24**
bbbb	0.35	0.27	0.24	0.27	0.19	1.03	1.03	**1.05**	**1.05**	1.03	1.03	**1.05**	**1.05**

Table 2: Asymptotic speeds for the patterns of length 4 on $\{\mathtt{a}, \mathtt{b}\}$ under the iid model $(\pi_{\mathtt{a}}, \pi_{\mathtt{b}}) = (0.1, 0.9)$.

	Horspool	FJS	TVSBS	EBOM	Hashq	Unif. 1-Heuristic	Unif. 2-Heuristic	Unif. 3-Heuristic	1-Heuristic	2-Heuristic	3-Heuristic
babbbaabab	0.85	0.42	0.22	1.42	1.57	1.09	1.98	1.54	1.22	2.38	**2.71**
ababbbbbab	0.70	0.55	0.29	0.77	0.85	1.19	1.34	1.47	1.43	1.87	**2.34**
aaabaaaaba	0.91	0.87	3.01	3.93	2.61	3.04	4.78	4.92	3.04	4.79	**5.27**
bbbabbabab	0.84	0.27	0.24	1.45	1.77	1.02	1.05	1.72	1.02	1.29	**2.30**
bbabaabbab	0.80	0.31	0.22	2.21	1.92	1.15	1.41	1.69	1.15	2.14	**3.02**
baabbaaaaa	4.10	2.17	1.86	2.40	2.45	1.99	4.03	4.72	1.99	4.06	**4.78**
abbbababbb	0.31	0.58	0.32	1.32	1.20	1.08	1.33	1.52	1.35	1.90	**2.29**
baabbbabba	0.90	0.81	0.68	1.39	1.14	1.61	1.12	1.63	1.73	2.44	**2.78**
baabbaabab	0.85	0.37	0.22	2.22	2.29	1.77	2.21	2.26	1.77	3.15	**3.54**
bbbbababbb	0.31	0.26	0.20	0.95	0.84	1.03	1.05	1.10	1.03	1.20	**1.50**

Table 3: Asymptotic speeds for some patterns of length 10 on $\{\mathtt{a}, \mathtt{b}\}$ (drawn from the uniform distribution) under the iid model $(\pi_{\mathtt{a}}, \pi_{\mathtt{b}}) = (0.1, 0.9)$.

Table 2 displays the asymptotic speeds with regard to the same patterns as Table 1, but under the iid model $(\pi_{\mathtt{a}}, \pi_{\mathtt{b}}) = (0.1, 0.9)$. This table shows the asymptotic speeds of the K-Heuristics and the Fastest strategies computed with regard to an uniform iid model (the columns starting with "Unif."). The strategies such obtained are not optimized according to the letter probabilities of the model. They may be used as general purpose approaches, while the strategies obtained from the model probabilities will be called *adapted* below. Overall, our methods are faster than the pre-existing algorithms, with a few exceptions: Horspool is faster than the 1-Heuristic for two patterns ending with the rare letter a: aaaa and baaa. And EBOM is faster than the 1-Heuristic for searching baba. The K-Heuristics and the Fastest strategies computed with regard to an uniform iid model have asymptotic speeds smaller than their counterparts obtained from the actual probabilities of the text model (here highly unbalanced). Nevertheless, the uniform approaches still perform quite well, notably better than the pre-existing algorithms, except for the uniform 1-Heuristic and the same patterns as above.

Considering longer patterns leads to similar observations. Table 3 shows the asymptotic speeds obtained for random patterns of length 10. The 3-Heuristic outperforms all the others approaches (the Fastest strategy cannot be computed for this length). The (uniform) 1-Heuristic is slower than algorithms such EBOM or Hashq. But both the uniform 2- and 3-Heuristic overall perform better than the pre-existing algorithms, though they are slightly slower for a few patterns.

7.3 Average speed

Our data benchmark consists in the *Wigglesworthia glossinidia* genome, known for its bias in nucleotide composition (78% of $\{\mathtt{a}, \mathtt{t}\}$), and the Bible in English

	Horspool	FJS	TVSBS	EBOM	Hashq	Unif. 1-Heuristic	Unif. 2-Heuristic	Unif. 3-Heuristic	Unif. Fastest	1-Heuristic	2-Heuristic	3-Heuristic	Fastest
`atat`	1.17	0.77	0.74	1.04	0.60	1.46	1.70	1.74	1.80	1.46	1.73	1.73	**1.81**
`tatg`	1.69	1.11	1.28	1.10	0.63	1.64	2.16	2.16	2.16	1.64	2.16	2.16	**2.17**
`aaat`	1.58	0.85	0.66	0.92	0.58	1.57	1.76	1.84	1.84	1.57	1.80	**1.90**	**1.90**
`tccc`	2.82	1.70	1.50	1.43	0.63	2.72	3.03	3.08	**3.09**	2.72	3.03	3.08	**3.09**
`caat`	1.53	0.77	0.71	1.05	0.63	1.74	1.87	1.83	1.87	1.74	1.94	**1.97**	**1.97**
`aacc`	2.47	1.56	1.45	1.23	0.63	2.12	2.67	**2.77**	**2.77**	2.12	2.69	**2.77**	**2.77**
`acta`	1.36	0.70	0.76	1.26	0.65	1.67	1.77	1.81	1.85	1.67	1.85	**1.88**	**1.88**
`tatc`	1.67	1.12	1.27	1.09	0.63	1.64	**2.15**	2.14	2.14	1.64	**2.15**	2.14	**2.15**
`gtga`	1.92	0.82	0.93	1.34	0.65	1.97	2.11	2.09	2.11	1.97	2.17	2.18	**2.19**
`gatt`	1.07	0.87	0.89	1.06	0.63	1.75	2.04	1.87	2.04	1.75	2.04	2.04	**2.05**
`he m`	3.16	2.02	1.85	1.39	0.66	2.88	3.23	3.23	**3.24**	2.88	**3.24**	**3.24**	**3.24**
`, to`	3.20	1.91	1.78	1.42	0.66	3.01	3.28	3.25	3.28	3.01	**3.29**	**3.29**	**3.29**
`usal`	3.53	2.18	1.87	1.48	0.66	3.50	3.60	3.60	**3.62**	3.50	**3.62**	**3.62**	**3.62**
`le t`	2.84	1.78	1.70	1.42	0.66	2.89	2.99	3.03	3.06	2.89	**3.07**	**3.07**	**3.07**
`hem`	3.04	1.59	1.48	1.45	0.66	2.89	3.05	**3.06**	3.01	2.89	3.05	**3.06**	**3.06**
` are`	3.13	1.76	1.60	1.45	0.67	3.01	**3.16**	**3.16**	3.11	3.01	**3.16**	**3.16**	**3.16**
`at d`	3.15	2.02	1.85	1.46	0.66	2.96	**3.22**	**3.22**	**3.22**	2.96	**3.22**	**3.22**	**3.22**
`f th`	2.72	1.81	1.69	1.35	0.65	2.94	3.03	2.87	3.05	2.94	**3.10**	**3.10**	**3.10**
`r th`	2.73	1.80	1.69	1.36	0.66	2.94	3.04	2.87	3.06	2.94	**3.12**	**3.12**	**3.12**
`fede`	3.35	1.97	1.73	1.49	0.66	3.23	**3.53**	**3.53**	**3.53**	3.23	**3.53**	**3.53**	**3.53**

Table 4: Average speeds for some patterns of length 4 picked from the benchmark data (the *Wigglesworthia glossinidia* complete genome and the Bible in English).

from [10].

Table 4 displays the average speeds of patterns randomly picked from the data. Let us remark that we are now dealing with real texts, which are not iid. In particular, the Fastest strategy could possibly be outperformed (this is not observed on the benchmark data). The 2- and 3-Heuristics, uniform and adapted, are faster than the pre-existing algorithms for all the patterns, whereas the 1-Heuristic is sometimes slightly outperformed by Horspool. Horspool is almost as fast as our approaches on the Bible while being sometimes significantly outperformed on the *Wigglesworthia glossinidia* genome. The average speeds are overall greater on the Bible than on the DNA sequence. In both cases, we do not observe a wide performance gap between the uniform and the adapted approaches, though our benchmark data are far from following an uniform iid model. Let us remark that the 2- and 3-Heuristics have almost the same performances both in the uniform and the adapted cases.

Table 5 shows the averages speeds with regard to patterns of length 30. The average speeds on the Bible are about twice those on the *Wigglesworthia glossinidia* genome. One actually expects the speed to be greater in average on texts with large alphabets, since the less likely the match between two symbols, the greater the shift expectation per iteration. Again the 3-Heuristic, uniform or

	Horspool	FJS	TVSBS	EBOM	Hashq	Unif. 1-Heuristic	Unif. 2-Heuristic	Unif. 3-Heuristic	1-Heuristic	2-Heuristic	3-Heuristic
tggataaaaatttgttattaccatatctat	1.8	1.0	2.2	6.1	4.7	2.3	4.6	7.5	2.3	5.5	**8.9**
cttctttaattatgttttctatttcttttt	2.8	1.8	3.3	7.2	5.2	2.5	5.0	7.9	2.6	5.3	**8.5**
gttctatttgttggagatttaaaataatta	3.0	1.7	2.0	6.2	4.4	2.6	5.3	8.0	2.7	6.1	**9.1**
tcctactttaacctctaaatgtcccttatt	1.4	1.0	2.1	7.0	4.4	2.5	5.5	8.4	2.6	6.5	**9.9**
tccttatgtaaaatataaatgtagcaattt	1.4	1.0	1.7	6.1	5.0	2.2	4.7	7.5	2.3	5.5	**8.5**
aaaagaaccccggcgaggggagtgaaatag	2.7	1.6	3.7	8.3	5.4	3.4	6.9	11.3	3.5	9.3	**12.5**
aattttcaactaatattaaaccacgttctg	3.0	1.7	2.8	6.3	4.7	2.0	4.8	8.1	2.2	5.6	**9.7**
aaaggtccattaagtattactatcacagca	2.7	1.1	2.3	7.4	5.5	2.3	5.1	8.1	2.5	6.6	**10.2**
agatttgcgtgatttaaaataatcatctaa	1.9	1.1	1.9	6.4	4.9	2.4	4.9	7.5	2.5	5.8	**9.1**
ataggaaaagattggattaaactagatatg	2.1	1.3	2.9	6.8	4.7	2.2	5.1	8.3	2.3	5.9	**9.7**
at the mount called the mount	11.9	5.9	8.0	11.7	8.6	8.0	17.2	18.0	8.0	17.6	**18.4**
ith Israel, to wit, with all t	13.3	7.2	8.4	12.6	8.7	8.9	17.1	18.0	9.1	18.0	**18.8**
esus going up to Jerusalem too	11.9	6.0	8.9	13.2	8.8	9.5	18.0	18.5	9.4	18.1	**18.7**
them, as they were able to hea	12.4	6.1	8.1	11.7	8.6	7.8	16.5	17.9	7.8	16.6	**18.0**
o in Osee, I will call them my	14.8	7.5	9.0	12.0	8.5	9.0	18.5	**19.3**	8.9	18.8	**19.3**
things are come upon thee, the	10.1	5.8	7.6	11.7	8.6	8.5	16.3	17.7	8.5	16.8	**18.1**
Syria, that dwelt at Damascus,	16.2	8.4	9.6	12.8	8.8	10.4	19.3	20.0	10.4	19.6	**20.1**
full of darkness. If therefor	12.8	6.8	8.6	11.9	8.5	9.4	18.5	18.9	9.4	18.7	**19.1**
e it: for there is no other sa	12.4	5.5	8.0	11.7	8.8	7.7	16.4	17.6	7.7	16.9	**17.9**
g, Syria is confederate with E	12.1	6.3	9.4	12.8	8.6	9.8	18.6	**18.9**	9.8	18.6	**18.9**

Table 5: Average speeds for some patterns of length 30 picked from the benchmark data (the *Wigglesworthia glossinidia* complete genome and the Bible in English).

adapted, outperforms the pre-existing algorithms. The speeds of the 3-Heuristic and of the 2-Heuristic differ in a greater amount than with patterns of length 4 for the *Wigglesworthia glossinidia* genome, and, to a smaller extent, for the Bible.

8 Discussion

In practical situations and though they don't take into account the letter frequencies, the uniform K-Heuristics and the uniform Fastest strategy perform generally almost as well as their adapted counterparts. The greatest difference observed is for the patterns of length 30 on the *Wigglesworthia glossinidia* genome (Table 5) and is relatively small. We do observe a notable amount of difference for the quite extreme case of the asymptotic speed under the iid model $(\pi_{\mathtt{a}}, \pi_{\mathtt{b}}) = (0.1, 0.9)$. But even for these frequencies, the uniform approaches show greater asymptotic speeds than any of the selected pre-existing algorithms.

The 3-Heuristic has very good results whatever the pattern or the text. There is no situation for which the performances of the 2-heuristic are far from the best. On the contrary, the performance ranking of the pre-existing algorithms depends heavily on the patterns and on the texts or the model. For instance, Horspool may perform very well, even almost optimally, for some patterns and texts or models while its speed may completely plummet in other situations.

The question of selecting the most efficient order of K-Heuristic still deserves further investigations. A basic answer could be "the greater, the better" but we should take into consideration that an higher order of heuristic comes with an increased computational cost. After some experiments, we observed that the asymptotic speed the K-Heuristic tends to stop improving beyond a certain rank. For instance, the difference in average speed between the 2- and 3-Heuristics for patterns of length 4, both on the genome and on the Bible, probably does not justify the computational cost of the 3-Heuristic, while it is worth to use the 3-Heuristic rather than the 2-Heuristic for searching patterns of length 30 in the Bible (not that much for the *Wigglesworthia glossinidia* genome). The best trade-off for the order of the K-Heuristic depends on the pattern (notably its length) and on the text features (in particular the alphabet size and the letter frequencies).

It is certainly possible to obtain efficient heuristic with a lower computational cost than for the K-Heuristic. Since in standard situation, the length of the text is much greater thna that of the pattern, there is no real reason for considering only pattern matching algorithms with linear pre-processings of the pattern. In the extreme case where the texts are arbitrary long with regard to the patterns, any pre-processing, i.e. whatever its computation time, would be beneficial as soon as it improves the overall speed.

Authors' contributions

Gilles Didier provided the initial idea, led the software development and wrote all the manuscript but the section *Evaluation*. Laurent Tichit collaborated on the software development, ran the tests and wrote the section *Evaluation*. Both authors read, edited and approved the final manuscript.

References

[1] C. Allauzen, M. Crochemore, and M. Raffinot. Efficient experimental string matching by weak factor recognition. In *Combinatorial Pattern Matching*, pages 51–72. Springer, 2001.

[2] R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. *Communications of the ACM*, 35(10):74–82, 1992.

[3] R. A. Baeza-Yates and M. Régnier. Average running time of the Boyer-Moore-Horspool algorithm. *Theoretical Computer Science*, 92(1):19 – 31, 1992.

[4] G. Barth. An analytical comparison of two string searching algorithms. *Information Processing Letters*, 18(5):249 – 256, 1984.

[5] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

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

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

[8] G. Didier. Optimal pattern matching algorithms. `http://arxiv.org/abs/1604.08437`, 2016.

[9] S. Faro and T. Lecroq. Efficient variants of the backward-oracle-matching algorithm. *International Journal of Foundations of Computer Science*, 20(06):967–984, 2009.

[10] S. Faro and T. Lecroq. The Exact Online String Matching Problem: A Review of the Most Recent Results. *ACM Comput. Surv.*, 45(2):13:1–13:42, Mar. 2013.

[11] F. Franek, C. G. Jennings, and W. F. Smyth. A simple fast hybrid pattern-matching algorithm. In *Combinatorial Pattern Matching*, pages 288–297. Springer, 2005.

[12] L. Guibas and A. Odlyzko. String overlaps, pattern matching, and non-transitive games. *Journal of Combinatorial Theory, Series A*, 30(2):183 – 208, 1981.

[13] R. N. Horspool. Practical fast searching in strings. *Software: Practice and Experience*, 10(6):501–506, 1980.

[14] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

[15] D. E. Knuth, J. H. Morris, Jr, and V. R. Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977.

[16] H. M. Mahmoud, R. T. Smythe, and M. Régnier. Analysis of Boyer-Moore-Horspool string-matching heuristic. *Random Struct. Algorithms*, 10(1-2):169–186, 1997.

[17] T. Marschall, I. Herms, H. Kaltenbach, and S. Rahmann. Probabilistic Arithmetic Automata and Their Applications. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 9(6):1737–1750, Nov. 2012.

[18] T. Marschall and S. Rahmann. Probabilistic Arithmetic Automata and Their Application to Pattern Matching Statistics. In P. Ferragina and G. M. Landau, editors, *Combinatorial Pattern Matching*, volume 5029 of *Lecture Notes in Computer Science*, pages 95–106. Springer Berlin Heidelberg, 2008.

[19] T. Marschall and S. Rahmann. Exact Analysis of Horspools and Sundays Pattern Matching Algorithms with Probabilistic Arithmetic Automata. In A.-H. Dediu, H. Fernau, and C. Martín-Vide, editors, *Language and Automata Theory and Applications*, volume 6031 of *Lecture Notes in Computer Science*, pages 439–450. Springer Berlin Heidelberg, 2010.

[20] T. Marschall and S. Rahmann. An Algorithm to Compute the Character Access Count Distribution for Pattern Matching Algorithms. *Algorithms*, 4(4):285, 2011.

[21] M. Régnier and W. Szpankowski. Complexity of Sequential Pattern Matching Algorithms. In M. Luby, J. D. Rolim, and M. Serna, editors, *Randomization and Approximation Techniques in Computer Science*, volume 1518 of *Lecture Notes in Computer Science*, pages 187–199. Springer Berlin Heidelberg, 1998.

[22] R. T. Smythe. The Boyer-Moore-Horspool heuristic with Markovian input. *Random Struct. Algorithms*, 18(2):153–163, 2001.

[23] D. M. Sunday. A very fast substring search algorithm. *Communications of the ACM*, 33(8):132–142, 1990.

[24] R. Thathoo, A. Virmani, S. Sai Lakshmi, N. Balakrishnan, and K. Sekar. TVSBS: A fast exact pattern matching algorithm for biological sequences. *Current Science*, 91(1):47–53, 2006.

[25] T.-H. Tsai. Average Case Analysis of the Boyer-Moore Algorithm. *Random Struct. Algorithms*, 28(4):481–498, July 2006.

[26] S. Wu, U. Manber, et al. A fast algorithm for multi-pattern searching. *Tech. Report TR-94-17, CS Dept., University of Arizona*, 1994.

[27] A. C.-C. Yao. The complexity of pattern matching for a random string. *SIAM Journal on Computing*, 8(3):368–387, 1979.