Journal of Discrete Algorithms 42 (2017) 45–60

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

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

Gilles Didier *, Laurent Tichit

Aix-Marseille Université, CNRS, Centrale Marseille, I2M, UMR7373, Marseille, France

ARTICLE INFO

Article history:
Received 3 May 2016
Received in revised form 29 August 2016
Accepted 12 November 2016
Available online 22 November 2016

Keywords:
Pattern matching
Algorithms
Average complexity
Markov chains

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 makes 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

* Corresponding author.
E-mail addresses: gilles.didier@univ-amu.fr (G. Didier), laurent.tichit@univ-amu.fr (L. Tichit).

http://dx.doi.org/10.1016/j.jda.2016.11.001
1570-8667/

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 [18–20,17] 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 $O_\Gamma = \max_{q \in Q} \{\boldsymbol{\alpha}(q)\}$.

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

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