Available online at www.sciencedirect.com

ScienceDirect

Journal of Discrete Algorithms 5 (2007) 682–695

JOURNAL OF DISCRETE ALGORITHMS

www.elsevier.com/locate/jda

A simple fast hybrid pattern-matching algorithm [☆]

Frantisek Franek [a], Christopher G. Jennings [b], W.F. Smyth [a,c,*]

[a] Algorithms Research Group, Department of Computing & Software, McMaster University, Hamilton ON L8S 4K1, Canada
[b] School of Computing Science, Simon Fraser University, 8888 University Drive, Burnaby BC V5A 1S6, Canada
[c] School of Computing, Curtin University, GPO Box U1987, Perth WA 6845, Australia

Received 5 December 2005; received in revised form 31 October 2006; accepted 23 November 2006

Available online 16 January 2007

Abstract

The Knuth–Morris–Pratt (KMP) pattern-matching algorithm guarantees both independence from alphabet size and worst-case execution time linear in the pattern length; on the other hand, the Boyer–Moore (BM) algorithm provides near-optimal average-case and best-case behaviour, as well as executing very fast in practice. We describe a simple algorithm that employs the main ideas of KMP and BM (with a little help from Sunday) in an effort to combine these desirable features. Experiments indicate that in practice the new algorithm is among the fastest exact pattern-matching algorithms discovered to date, apparently dominant for alphabet size above 15–20.
© 2006 Elsevier B.V. All rights reserved.

MSC: 68U15; 68R15

Keywords: String; Word; Pattern-matching; Algorithm

1. Introduction

Since 1977, with the publication of both the Boyer–Moore [2] and Knuth–Morris–Pratt [24] pattern-matching algorithms, there have certainly been hundreds, if not thousands, of papers published that deal with exact pattern-matching, and in particular discuss and/or introduce variants of either BM or KMP. The pattern-matching literature has had two main foci:

1. reducing the number of letter comparisons required in the worst/average case (for example, [4–6,8,13–15], see also [29, pp. 219–225]);
2. reducing the time requirement in the worst/average case (for example, [7,9,20,21,30]).

[☆] A preliminary version of this paper appeared in Proc. 16th Annual Symp. Combinatorial Pattern Matching, LNCS 3537, Springer-Verlag, 2005. The authors were supported in part by grants from the Natural Sciences & Engineering Research Council of Canada.
[*] Corresponding author.
E-mail addresses: franek@mcmaster.ca (F. Franek), cjennings@acm.org (C.G. Jennings), smyth@computing.edu.au (W.F. Smyth).

1570-8667/$ – see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.jda.2006.11.004

This contribution resides in the second of these categories: in an effort to reduce processing time, we propose a mixture of Sunday's variant [30] of BM [2] with KMP [24,27]. Our goal is to combine the best/average case advantages of Sunday's algorithm (BMS) with the worst case guarantees of KMP. According to the experiments we have conducted, our new algorithm (FJS) is among the fastest in practice for the computation of all occurrences of a pattern $\boldsymbol{p} = \boldsymbol{p}[1..m]$ in a text string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ on an alphabet Σ of size k. Moreover, based on $\Theta(m+k)$-time preprocessing, it guarantees that matching requires at most $3n - 2m$ letter comparisons and $O(n)$ matching time.

Several comparative surveys of pattern-matching algorithms have been published over the years [10,21,25,26,29] and a useful website [3] is maintained that gives executable C code for the main algorithms (including preprocessing) and describes their important features.

In this paper we first introduce, in Section 2, the new algorithm, establish its asymptotic time and space requirements in the worst case, and demonstrate its correctness. Next, extensions to the algorithm for large alphabets and patterns with don't-care letters are discussed in Section 3. Then in Section 4 we describe experiments that compare Algorithm FJS with algorithms that, from the available literature, appear to provide the best competition in practice; specifically,

- Horspool's algorithm (BMH) [20];
- Sunday's algorithm (BMS) [30];
- Reverse Colussi (RC) [7];
- Turbo-BM (TBM) [9].

Finally, in Section 5, we draw conclusions from our experiments.

2. The new algorithm

Algorithm FJS searches for matches to $\boldsymbol{p} = \boldsymbol{p}[1..m]$ in $\boldsymbol{x} = \boldsymbol{x}[1..n]$ by shifting $\boldsymbol{p}$ from left to right along $\boldsymbol{x}$. At the start of each step, position $j = 1$ of $\boldsymbol{p}$ is aligned with a position $i \in 1..n-m+1$ in $\boldsymbol{x}$, so that position m of $\boldsymbol{p}$ is therefore aligned with position $i' = i + m - j$ in $\boldsymbol{x}$.

In KMP matching, $\boldsymbol{p}$ is matched left-to-right with $\boldsymbol{x}[i], \boldsymbol{x}[i+1]$, and so on until either a match with all of $\boldsymbol{p}$ is found or else a mismatch is located at some position $j \in 1..m$. In the case of a mismatch at $j > 1$, we say that a *partial match* has been determined with $\boldsymbol{p}[1..j-1]$. As matching proceeds, both i and j are incremented, thus maintaining the relationship $i' = i + m - j$, the basic invariant of Algorithm FJS.

The strategy of FJS is as follows:

1. Whenever no partial match of $\boldsymbol{p}$ with $\boldsymbol{x}[i..i+m-1]$ has been found, Sunday shifts (described below) are performed to determine the next position i' at which $\boldsymbol{x}[i'] = \boldsymbol{p}[m]$. When such an i' has been found, KMP matching is then performed on $\boldsymbol{p}[1..m-1]$ and $\boldsymbol{x}[i'-m+1..i'-1]$.
2. If a partial match of $\boldsymbol{p}$ with $\boldsymbol{x}$ has been found, KMP matching is continued on $\boldsymbol{p}[1..m]$.

The pseudocode shown below illustrates this overall strategy.

Algorithm 1 *(Hybrid matching).*
Find all occurrences of $\boldsymbol{p} = \boldsymbol{p}[1..m]$ in $\boldsymbol{x} = \boldsymbol{x}[1..n]$

```
**if** $m < 1$ **then return**
$j \leftarrow 1$; $i \leftarrow 1$; $i' \leftarrow m$; $m' \leftarrow m - 1$
**while** $i' \leqslant n$ **do**
  **if** $j \leqslant 1$ **then**

  – If no partial match with $p$, perform Sunday shifts,
  – returning next position $i'$ such that $x[i'] = p[m]$
  SUNDAY-SHIFT($i'$)
```