Journal of Discrete Algorithms 49 (2018) 27–36

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Improved pattern-scan-order algorithms for string matching ☆

Cheol Ryu, Kunsoo Park *

Department of Computer Science and Engineering, Seoul National University, Seoul, Republic of Korea

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
String matching
Pattern scan order
Asymptotic speed

ABSTRACT

The pattern scan order is a major factor affecting the performance of string matching algorithms. Depending on the pattern scan order, one can reduce the number of comparisons in a window or increase the shift length. Classical algorithms for string matching determine the pattern scan order only using the characteristics of a text and a pattern. However, if we additionally use the scan results at the time we determine each scan position of the pattern, we can improve the performance of string matching. In this paper we propose new pattern-scan-order algorithms that maximize shift lengths using scan results. We present the theoretical analysis and experimental results that these algorithms run faster than previous algorithms on average.

1. Introduction

String matching is one of fundamental problems in computer science [9]. The string matching problem is to find all occurrences of a pattern P of length m in a text T of length n. A string matching algorithm typically works with a sliding window. It makes comparisons in the current window by some pattern scan order, and if there is a mismatch in the window or the whole pattern is matched, it shifts the window to the right and repeat the same process.

Extensive research has been done to improve the performance of string matching. Knuth et al. [17] proposed the Knuth–Morris–Pratt algorithm, which compares the pattern with the text from left to right using the prefix information of the pattern. Boyer and Moore [3] proposed the Boyer–Moore algorithm (BM for short) that changes the direction of scanning the pattern (i.e., right to left) using the good-suffix heuristic and the bad-character heuristic. Horspool [15] gave a variant of the Boyer–Moore algorithm (BMH for short) that considers only the bad-character heuristic. Sunday [21] presented the Quick Search algorithm which improves the bad-character heuristic of the BM algorithm. Moreover, Sunday gave the Optimal Mismatch algorithm (OM for short) and the Maximal Shift algorithm (MS for short), each of which scans the pattern in a specific order.

The performance evaluation of string matching algorithms is also an important issue in string matching. There are basically two analysis methods: one is the worst-case analysis that finds the maximum number of comparisons over all possible patterns and texts [7,13,5,6], and the other is the average-case analysis which finds the average number of comparisons over random patterns and texts [25,23,20]. Baeza-Yates et al. [1,2] presented an average-case analysis of the BMH algorithm and compared the theoretical result to experimental results. Didier [10] defined the *asymptotic speed* which is an average-case analysis when each text character is generated by an independent and identical distribution with given charac-

☆ This research was supported by Collaborative Genome Program for Fostering New Post-Genome industry through the National Research Foundation of Korea (NRF) funded by the Ministry of Science ICT and Future Planning (No. NRF-2014M3C9A3063541).

* Corresponding author.
E-mail addresses: cryu@theory.snu.ac.kr (C. Ryu), kpark@theory.snu.ac.kr (K. Park).

https://doi.org/10.1016/j.jda.2018.05.002
1570-8667/

ter probabilities. Many studies [16,14] counted the number of actual comparisons and compared the average performances of various string matching algorithms. In this paper we use the asymptotic speed and the number of actual comparisons as the measures of average performances.

To improve average performances of string matching algorithms, many researchers have attempted to change the *pattern scan order* [12,4]. Colussi [8] proposed the Reverse Colossi algorithm that changes the pattern scan order of the BM algorithm to reduce the number of comparisons. Sunday [21] gave the OM and MS algorithms which obtain better pattern scan orders with $O(m^2)$ preprocessing time, among which OM determines a pattern scan order according to character probabilities. Motivated by OM, Külekci [18] determined a pattern scan order according to consecutive character probabilities of the DNA sequence modeled by a Markov chain. Lu et al. [19] defined an *average shift function* and gave an algorithm to find a pattern scan order that optimizes the average shift function by using a branch and bound. Didier and Tichit [11] proposed an algorithm to find a pattern scan order that optimizes the asymptotic speed [10] by constructing a deterministic finite automaton (DFA for short) with $O(2^m)$ states. Then they gave k-heuristic algorithms for $k \geq 1$, where a k-heuristic algorithm constructs a DFA with $O(m^{k+1})$ states in $O(m^{k+2})$ time.

A *Boyer–Moore family algorithm* [24] is an algorithm that computes its shift length by using only the scan result of the current window (i.e., not using the scan results of the previous windows). The DFA associated with a Boyer–Moore family algorithm has m states. In this paper we consider Boyer–Moore family algorithms and propose new pattern-scan-order algorithms that use the scan results at each time they determine the ith scan position of the pattern, in order to maximize shift lengths: the Greedy Shift algorithm (GS for short) and the Higher-Order Maximal Shift algorithm (HS for short). Each of GS and HS builds a DFA with m states in $O(m^2)$ preprocessing time. We show that GS and HS are faster than OM and MS with respect to the asymptotic speed and the average number of actual comparisons. GS and HS are faster than the 1-heuristic algorithm when the pattern length is large, even though they use less space and preprocessing time than the 1-heuristic algorithm.

This paper is organized as follows. Section 2 describes notations and a framework of string matching using a DFA. In Section 3 we derive a simple formula of the asymptotic speed for Boyer–Moore family algorithms. In Section 4 we present new pattern-scan-order algorithms GS and HS. Section 5 shows the experimental results of comparing the average performances of the proposed algorithms and existing algorithms, and we conclude in Section 6.

2. Preliminaries

Text T and pattern P are strings of lengths n and m, respectively, over an alphabet Σ. Let $t[i]$, $1 \leq i \leq n$, denote the ith character of T, and $p[i]$, $1 \leq i \leq m$, denote the ith character of P. Let $f_T(x)$ for $x \in \Sigma$ be the probability that a text character in T is x, assuming that each text character in T has an independent and identical distribution.

A string matching algorithm reads the text with a window whose size is m. When the position of the window on T is w, the string matching algorithm compares $t[w+1], \ldots, t[w+m]$ and $p[1], \ldots, p[m]$. A *pattern scan order* $(\alpha(1), \alpha(2), \ldots, \alpha(m))$ is a permutation of $(1, 2, \ldots, m)$. That is, $t[w+\alpha(1)]$ and $p[\alpha(1)]$ are compared first, and then $t[w+\alpha(2)]$ and $p[\alpha(2)]$ are compared, etc. The result of a comparison between $t[w+\alpha(i)]$ and $p[\alpha(i)]$ is either a match (which is denoted by $p[\alpha(i)]$) or a mismatch (which is denoted by $p[\alpha(i)]$ with a dash). A *scan result up to i* is defined as the results of the first i comparisons in a window, where an uncompared position is denoted by $*$. For example, when P is abab, $\alpha(1) = 4$, $\alpha(2) = 3$, and the results of the first two comparisons are a match and a mismatch, the scan result up to 2 is $**\bar{a}b$.

A string matching algorithm with a sliding window works as follows. First, it places the window at position 0, and makes comparisons in the window by the pattern scan order. If there is a mismatch in the window or the whole pattern is matched, it shifts the window to the right and repeat the same process. Let w be the current window position. We say that a shift length k is *safe* if it is guaranteed that the pattern cannot occur at window position $w+k$ by the scan results so far; it is *non-safe* if the pattern may occur. When some scan results are given, we define the *maximal shift length* as the smallest non-safe shift length computed by the scan results. A *Boyer–Moore family algorithm* (BM family algorithm for short) [24] is an algorithm that computes the maximal shift length by using only the scan result of the current window. In this paper we will deal with BM family algorithms only.

A BM family can be represented by a deterministic finite automaton $(Q, o, F, \alpha, \delta, \gamma)$ (DFA for short) as follows.

- $Q = \{q_1, q_2, \ldots, q_m\}$, where q_i is the state that makes the ith comparison in the current window.
- o is the initial state, which is q_1.
- F is the set of *pre-match* states (i.e., states just before the whole pattern matches). In BM family algorithms, $F = \{q_m\}$.
- α is a pattern scan order, i.e., a permutation of $(1, 2, \ldots, m)$, and $\alpha(i)$ is the position where state q_i make the ith comparison. For notational convenience, $\alpha(q_i)$ will mean $\alpha(i)$.
- $\delta : Q \times \Sigma \to Q$ is a state transition function. In BM family algorithms, if the ith comparison at state q_i for $i < m$ is a match, the next state is q_{i+1}. If it is a mismatch, the next state is q_1. At state q_m the next state is q_1, whether the mth comparison is a match or not. Hence, for a text character $x \in \Sigma$

$$\delta(q_i, x) = \begin{cases} q_{i+1} & \text{if } x = p[\alpha(i)], i < m \\ q_1 & \text{otherwise.} \end{cases}$$