Theoretical Computer Science 812 (2020) 137–148

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Fast string matching for DNA sequences

Cheol Ryu [a], Thierry Lecroq [b], Kunsoo Park [a,*]

[a] *Department of Computer Science and Engineering, Seoul National University, Seoul 08826, Republic of Korea*
[b] *Normandie University, UNIROUEN, LITIS, 76000 Rouen, France*

ARTICLE INFO

Article history:
Received 31 January 2019
Received in revised form 29 August 2019
Accepted 18 September 2019
Available online 24 September 2019

Keywords:
String matching
Pattern scan order
Scan speed

ABSTRACT

In this paper we propose the Maximal Average Shift (MAS) algorithm that finds a pattern scan order that maximizes the average shift length. We also present two extensions of MAS: one improves the scan speed of MAS by using the scan result of the previous window, and the other improves the running time of MAS by using q-grams. These algorithms show better average performances in scan speed than previous string matching algorithms for DNA sequences.

© 2019 Elsevier B.V. All rights reserved.

1. Introduction

String matching is one of fundamental problems in computer science [8]. The string matching problem is to find all occurrences of a pattern P of length m in a text T of length n. String matching is widely used in applications such as text editing and DNA sequence analysis.

Many string matching algorithms have been proposed over the years [5]. Knuth, Morris, and Pratt [18] proposed the KMP algorithm, which compares the pattern against the text from left to right. Boyer and Moore [4] proposed the Boyer–Moore algorithm (BM for short) which makes comparisons from right to left and uses the good-suffix heuristic and the bad-character heuristic. Horspool [16] gave a variant of the Boyer–Moore algorithm (BMH for short) that uses only the bad-character heuristic. A string matching algorithm typically works with a sliding window; it makes comparisons in a window and then moves the window to the right by some shift length. A *Boyer–Moore family algorithm* (BM-family algorithm for short) is an algorithm that 1) computes the shift length by using the scan result of the current window only, and 2) moves the window when the first mismatch is found in the current window [28]. In this paper we will divide string matching algorithms into two groups: BM-family algorithms and those that are not.

The performance evaluation of string matching algorithms is 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 [7,14,6], and the other is the average-case analysis. The average-case analysis finds the average number of text characters scanned for random patterns [2,3,17,20,15,23] or measures the average running time [17,20,12]. In this paper we use two measures of average-case analysis: 1) *scan speed*, which is the average ratio of the text length to the number of text characters scanned for random patterns, and 2) *running time*, which is the average running time for random patterns. The scan speed is an important measure in some special settings, e.g., when characters are encrypted [17].

To improve the average performance of string matching algorithms, many researchers have attempted to change the pattern scan order. Sunday [26] proposed the Optimal Mismatch algorithm (OM for short) that determines a pattern scan

* Corresponding author.
E-mail addresses: cryu@theory.snu.ac.kr (C. Ryu), Thierry.Lecroq@univ-rouen.fr (T. Lecroq), kpark@theory.snu.ac.kr (K. Park).

https://doi.org/10.1016/j.tcs.2019.09.031

order according to character frequencies, and Külekci [19] presented a variant of the OM algorithm for DNA sequences. In order to make the shift length longer, Sunday [26] designed the Maximal Shift algorithm (MS for short) that improves the pattern scan order of the BM algorithm. Lu, Lu, and Lee [22] gave an algorithm that optimizes the average shift function by using a branch and bound. Didier and Tichit presented an algorithm to find a pattern scan order that optimizes the asymptotic speed [10]. Ryu and Park [25] proposed the Greedy Shift algorithm (GS for short) and the High-order Maximal Shift algorithm (HS for short) that improves the shift length using the scan result of the current window.

In this paper we propose a BM-family algorithm called the Maximal Average Shift (MAS for short) algorithm that finds a pattern scan order that maximizes the average shift length. We also give two extensions of MAS: TMAS and QMAS$_q$. TMAS improves the scan speed of MAS by using the scan result of the previous window, and QMAS$_q$ improves the running time of MAS by using q-grams.

To evaluate the average performance of MAS, we compare it with other BM-family algorithms (BM, BMH, OM, MS, GS, HS) in scan speed and running time. In the experiments, the text is Human chromosomes 1, 10, and 20 downloaded from the 1000 Genomes Project website [27], and the pattern lengths are 4, 8, 16, 32, 64, and 128. MAS shows the best performances in both scan speed and running time for all pattern lengths.

We also compare TMAS and QMAS$_q$ with other state-of-the-art algorithms which are not BM-family algorithms. Some are non-BM-family algorithms because they use the scan result of the previous windows, e.g., the Shift Vector Matching algorithm (SVM for short) due to Peltola and Tarhio [24] remembers the scan results of all previous windows using the bit-parallel approach. Some others are non-BM-family algorithms because they do not make a shift as soon as they find the first mismatched character. The Backward Oracle Matching algorithm (BOM for short) by Allauzen, Crochemore, and Raffinot [1], an efficient variant of the Backward Oracle Matching algorithm (EBOM for short) by Faro and Lecroq [11], and the Turbo Reverse Factor algorithm (TRF for short) by Crochemore, Czumaj, Gasieniec, Jarominek, Lecroq, Plandowski, and Rytter [9] are such algorithms based on suffix automata. BOM, EBOM, and TRF make comparisons in a window until they find a text substring which is not a pattern substring (even though there is a mismatch). The HASH$_q$ algorithm by Lecroq [21] works with q-grams rather than characters, and applies a hash function to q-grams. In scan speed, TMAS shows the best performances among non-BM-family algorithms when pattern length m is 64 or less. SVM shows comparable performances in scan speed when m is 16 or less, but it cannot run when m is 32 or more due to its bit parallelism. When $m = 128$, TRF is the best performer in scan speed. In running time, EBOM is the fastest when m is 4. When m is 8 or more, HASH$_q$ is the best performer in running time, and QMAS$_q$ is the runner-up.

This paper is organized as follows. Section 2 describes basic notations and definitions of string matching algorithms. In Sections 3 and 4, we present the MAS algorithm and show the experimental results of MAS and BM-family algorithms. Section 5 presents two extensions of the MAS algorithm and shows the experimental results. We conclude in Section 6.

2. Preliminaries

For a string S, $S[i]$ denotes the ith character of S, and $S[i..j]$ denotes the substring of S starting from i and ending at j. The text T and the pattern P are strings of lengths n and m, respectively, over an alphabet Σ. Let $f_T(x)$ for $x \in \Sigma$ be the probability that a text character in T is x, assuming that the distributions for all text characters in T are independent and identical.

A string matching algorithm reads the text in a window whose size is m. When the window is on position w in T, the algorithm compares $T[w+1..w+m]$ against $P[1..m]$. A *pattern scan order* $(\text{scan}[1], \text{scan}[2], \dots, \text{scan}[m])$ is a permutation of $(1, 2, \dots, m)$. That is, $T[w + \text{scan}[1]]$ and $P[\text{scan}[1]]$ are compared first, and then $T[w + \text{scan}[2]]$ and $P[\text{scan}[2]]$ are compared, etc. Text positions $w + \text{scan}[1], \dots, w + \text{scan}[m]$ are called *scan positions* of window w. The text characters in scan positions which are accessed in a window are called the *scan result* of the window.

A string matching algorithm first places the window at position 0, and makes comparisons in the window by a pattern scan order. If there is a mismatch in the window or the whole pattern is matched, it determines the *shift length* and moves the window to the right by the shift length, and it repeats the same process. A *BM-family algorithm* is an algorithm that 1) computes the shift length by using the scan result of the current window only, and 2) moves the window when the first mismatch is found in the current window [28].

3. Maximal average shift algorithm

To reduce the number of text characters accessed (i.e., scanned) by a string matching algorithm, we first need to reduce the number of text characters scanned in one window, and second we need to increase the shift length. Therefore, Sunday proposed the OM algorithm to find a scan order to reduce the number of text characters scanned in one window and the MS algorithm to find a scan order to increase the shift length [26]. In general, it is difficult to find an optimal string matching algorithm that has the smallest number of text characters scanned for arbitrary patterns because the shift length increases as the number of text characters scanned in a window increases. It is difficult to consider all the elements of string matching simultaneously, such as the pattern scan order, the condition of moving the window, and whether the scan result of the previous window is used or not when calculating the shift length. Therefore, we will focus on the problem of finding a pattern scan order that maximizes the shift length in the BM-family algorithms.

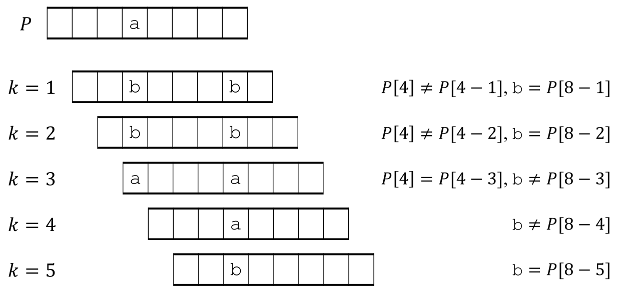
P
a
k = 1
b b
P[4] ≠ P[4 − 1], b = P[8 − 1]
k = 2
b b
P[4] ≠ P[4 − 2], b = P[8 − 2]
k = 3
a a
P[4] = P[4 − 3], b ≠ P[8 − 3]
k = 4
a
b ≠ P[8 − 4]
k = 5
b
b = P[8 − 5]

Fig. 1. $\text{shift}_2[8][\texttt{b}]$ is 5 when $P = \texttt{abbaabbb}$ and $\text{scan}[1] = 4$.

Table 1
First and second scan positions of MAS when $P = \texttt{abbaabbb}$ and $f_T(\texttt{a}) = f_T(\texttt{b}) = 0.5$.

l	1	2	3	4	5	6	7	8
$P[l]$	a	b	b	a	a	b	b	b
$\text{shift}_1[l][\texttt{a}]$	1	1	2	3	1	1	2	3
$\text{shift}_1[l][\texttt{b}]$	1	2	1	1	2	3	1	1
$\text{avr\_shift}_1[l]$	1	1.5	1.5	**2**	1.5	2	1.5	2
$\text{shift}_2[l][\texttt{a}]$	3	3	3	.	4	5	3	3
$\text{shift}_2[l][\texttt{b}]$	3	3	3	.	3	3	4	5
$\text{avr\_shift}_2[l]$	3	3	3	.	3.5	**4**	3.5	4

To find a pattern scan order that maximizes the shift length, we determine $\text{scan}[1], \text{scan}[2], \ldots, \text{scan}[m]$ in this order. To determine scan[1], we need to compute the shift length when $\text{scan}[1] = l$ for each value of $1 \le l \le m$, and select l that maximizes the shift length as scan[1]. Hence, we define $\text{shift}_i[l][s]$ as the shift length of a BM-family algorithm when $\text{scan}[i] = l$ and the text character in the ith scan position is s. When a BM-family algorithm accesses the ith scan position, it must have matches in all previous scan positions. Therefore, $\text{shift}_i[l][s]$ is computed as follows.

$$\text{shift}_i[l][s] = \min\{k \ge 1 \mid P[\text{scan}[j]] = P[\text{scan}[j]-k] \text{ for all } 1 \le j < i, \text{ and } s = P[l-k]\}, \tag{1}$$

where $P[l]$ for $l \le 0$ is a special character that matches any text character. The jth scan position for any $j < i$ must have a match (i.e., $P[\text{scan}[j]] = T[w + \text{scan}[j]]$), and in order for k to be a shift length, $T[w + \text{scan}[j]]$ should match $P[\text{scan}[j] - k]$. Thus, we have $P[\text{scan}[j]] = P[\text{scan}[j] - k]$ for all $1 \le j < i$ in Equation (1). Fig. 1 shows the calculation of $\text{shift}_2[8][\texttt{b}]$ when $P = \texttt{abbaabbb}$ and $\text{scan}[1] = 4$. In this case, the minimum number k that satisfies Equation (1) is 5, and thus $\text{shift}_2[8][\texttt{b}]$ is set to 5.

Ryu and Park proposed the GS algorithm that determines a pattern scan order to maximize the shift length when the ith scan position is a mismatch, and the HS algorithm that determines a pattern scan order to maximize the shift length when the ith scan position is a match [25]. However, if we assume that the text character s follows the distribution $f_T(s)$, the *average shift length* $\text{avr\_shift}_i[l]$ when the ith scan position is l can be defined as follows:

$$\text{avr\_shift}_i[l] = \sum_s (f_T(s) \times \text{shift}_i[l][s]).$$

The Maximal Average Shift (MAS) algorithm is a BM-family algorithm that finds a pattern scan order that maximizes the average shift length. That is, we select l such that $\text{avr\_shift}_i[l]$ is the largest as $\text{scan}[i]$ for $i = 1, 2, \ldots, m$. The case that multiple positions have the largest average shift length will be discussed later (for the moment we assume that we choose the leftmost position in such a case). Table 1 shows an example where $P = \texttt{abbaabbb}$ and $f_T(\texttt{a}) = f_T(\texttt{b}) = 0.5$. Since $\text{shift}_1[4][\texttt{a}] = 3$ and $\text{shift}_1[4][\texttt{b}] = 1$, we have $\text{avr\_shift}_1[4] = 2$. Since it is the largest (and the leftmost) among $\text{avr\_shift}_1[l]$, we set $\text{scan}[1] = 4$. Then $\text{shift}_2[6][\texttt{a}] = 5$ and $\text{shift}_2[6][\texttt{b}] = 3$ by Equation (1), and so $\text{avr\_shift}_2[6] = 4$. Hence, we set $\text{scan}[2] = 6$.

For string matching, there should be a shift length table as well as a scan order. The shift length table of MAS, $\text{mas\_shift}[l][s]$, is defined as the shift length when the scan position is l and the text character in the scan position is s. That is,

$$\text{mas\_shift}[l][s] = \min\{k \ge 1 \mid P[\text{scan}[j]] = P[\text{scan}[j]-k] \text{ for all } 1 \le j < \text{scan}^{-1}(l), \text{ and } s = P[l-k]\},$$

because a BM-family algorithm must have matches in all previous scan positions. By Equation (1), we get

$$\text{mas\_shift}[l][s] = \text{shift}_i[l][s] \text{ such that } \text{scan}[i] = l. \tag{2}$$

Algorithm 1 Preprocessing of MAS.

```
procedure MAS_PREPROCESSING($P, m, f_T$)
  $U \leftarrow \{1, 2, \ldots, m\}$
  for all $l \in U$ do
    for all $s \in \Sigma$ do
      shift[$l$][$s$] $\leftarrow 1$
  for $k \leftarrow 1$ to $m$ do
    safe[$k$] $\leftarrow 0$
  for $i \leftarrow 1$ to $m$ do
    for all $l \in U$ do
      avr_shift[$l$] $\leftarrow 0$
      for all $s \in \Sigma$ do
        for $k \leftarrow$ shift[$l$][$s$] to $m$ do
          if safe[$k$] $= 0$ and $s = P[l-k]$ then
            shift[$l$][$s$] $\leftarrow k$
            break
        avr_shift[$l$] $\leftarrow$ avr_shift[$l$] $+ f_T(s) \times$ shift[$l$][$s$]
    Pick the largest $l$ in $U$ such that avr_shift[$l$] is maximum
    Remove $l$ from $U$
    scan[$i$] $\leftarrow l$
    for $k \leftarrow 1$ to $l-1$ do
      if $P[l] \neq P[l-k]$ then
        safe[$k$] $\leftarrow 1$
  return (scan, shift)
```

Fig. 2. Preprocessing of MAS.

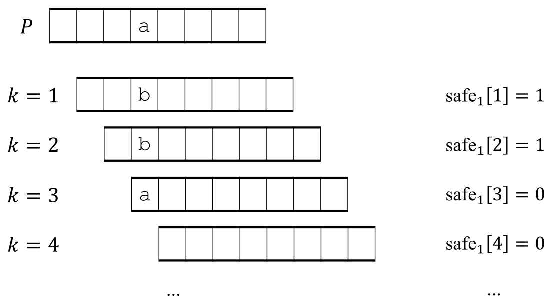

Fig. 3. $\text{safe}_1[k]$ when $P = \texttt{abbaabbb}$ and $\text{scan}[1] = 4$.

Fig. 2 shows the preprocessing of MAS for an arbitrary pattern P, which computes scan order scan[i] and shift length table mas_shift[l][s] by using $\text{shift}_i[l][s]$. Because it takes too much time to compute $\text{shift}_i[l][s]$ by Equation (1), we use the notion of *safe* shift length:

$$\text{safe}_i[k] = \begin{cases} 1 & \text{if } \exists j \text{ such that } P[\text{scan}[j]] \neq P[\text{scan}[j]-k] \text{ for } 1 \leq j \leq i \\ 0 & \text{otherwise.} \end{cases}$$

Fig. 3 shows the calculation of $\text{safe}_1[k]$ when $P = \texttt{abbaabbb}$ and $\text{scan}[1] = 4$. When $k = 1, 2$, $\text{safe}_1[k]$ is set to 1 because $P[\text{scan}[1]] \neq P[\text{scan}[1]-k]$. Then, Equation (1) can be expressed as

$$\text{shift}_i[l][s] = \min\{k \geq 1 \mid \text{safe}_{i-1}[k] = 0 \text{ and } s = P[l-k]\}.$$

If $\text{safe}_{i-1}[k] = 0$, then $\text{safe}_{i-2}[k]$ must be 0 by definition of safe shift length. Thus we have $\text{shift}_i[l][s] \geq \text{shift}_{i-1}[l][s]$, and the equation above can be changed to

$$\text{shift}_i[l][s] = \min\{k \geq \text{shift}_{i-1}[l][s] \mid \text{safe}_{i-1}[k] = 0 \text{ and } s = P[l-k]\}. \quad (3)$$

The preprocessing of MAS in Fig. 2 computes $\text{shift}_i[l][s]$ by Equation (3).

- U in Fig. 2 is the set of positions which are not yet determined as pattern scan positions, and it is implemented as an array of size m. Initially, U is $\{1, 2, \ldots, m\}$, shift[l][s] is set to 1 (which is the minimum shift length), and safe[k] is set to 0.

Table 2
Preprocessing of MAS when $P = \texttt{abbaabbb}$ and $f_T(\texttt{a}) = f_T(\texttt{b}) = 0.5$.

i	avr_shift$_i[l]$								scan[i]	safe$_i[k]$								shift$_i$[scan[i]]	
	1	2	3	4	5	6	7	8		1	2	3	4	5	6	7	8	[a]	[b]
1	1	1.5	1.5	2	1.5	2	1.5	2	4	1	1	0	0	0	0	0	0	3	1
2	3	3	3	.	3.5	4	3.5	4	6	1	1	0	0	1	0	0	0	5	3
3	3	3	3	.	3.5	.	3.5	4.5	8	1	1	1	1	1	0	1	0	3	6
4	6	6	6	.	6	.	7	.	7	1	1	1	1	1	1	1	0	6	8
5	8	8	8	.	8	.	.	.	1	1	1	1	1	1	1	1	0	8	8
6	.	8	8	.	8	.	.	.	2	1	1	1	1	1	1	1	0	8	8
7	.	.	8	.	8	.	.	.	3	1	1	1	1	1	1	1	0	8	8
8	.	.	.	.	8	.	.	.	5	1	1	1	1	1	1	1	0	8	8

Table 3
mas_shift$[l][s]$ when $P = \texttt{abbaabbb}$ and $f_T(\texttt{a}) = f_T(\texttt{b}) = 0.5$.

l	1	2	3	4	5	6	7	8
$P[l]$	a	b	b	a	a	b	b	b
mas_shift$[l]$[a]	8	8	8	3	8	5	6	3
mas_shift$[l]$[b]	8	8	8	1	8	3	8	6

Algorithm 2 String Matching Algorithm MAS(P, m, T, n, f_T).

```
1: (scan, mas_shift) ←MAS_Preprocessing($P, m, f_T$)
2: $w \leftarrow 0$
3: while $w \le n - m$ do
4:     $i \leftarrow 1$
5:     while $i \le m$ and $T[w + \text{scan}[i]] = P[\text{scan}[i]]$ do
6:         $i \leftarrow i + 1$
7:     if $i > m$ then
8:         Output an occurrence at position $w + 1$
9:         $w \leftarrow w + \text{mas\_shift}[\text{scan}[m]][T[w + \text{scan}[m]]]$
10:    else
11:        $w \leftarrow w + \text{mas\_shift}[\text{scan}[i]][T[w + \text{scan}[i]]]$
```

Fig. 4. String matching algorithm MAS.

- At the ith iteration of the **for** loop in line 8, we compute shift$_i[l][s]$, avr_shift$_i[l]$, scan$[i]$, and safe$_i[k]$. Thus, shift$[l][s]$ in line 14, avr_shift$[l]$ in line 16, and safe$[k]$ in line 22 mean shift$_i[l][s]$, avr_shift$_i[l]$, and safe$_i[k]$, respectively. Note that shift$[l][s]$ in line 12 and safe$[k]$ in line 13 mean shift$_{i-1}[l][s]$ and safe$_{i-1}[k]$, respectively.
- Since shift$_i[l][s]$ is minimum k that satisfies the condition in Equation (3), we get out of the loop in line 15 as soon as we find such a k.
- Picking the largest l in U such that avr_shift$[l]$ is maximum in line 17 is implemented as the usual procedure to find the maximum. Then scan$[i]$ is set to l. Since l is removed from U in line 18, the value of shift$[l][s]$ will not be changed any more. Since the value of shift$[l][s]$ is shift$_i[l][s]$ such that scan$[i] = l$, it is mas_shift$[l][s]$ by Equation (2). That is, the shift array which the preprocessing of MAS returns is in fact the mas_shift array.
- Finally, we compute safe$_i[k]$. Since we have determined scan$[i] = l$, safe$_i[k]$ becomes 1 if $P[l] \neq P[l-k]$ for $k < l$.

Table 2 shows the preprocessing of MAS for pattern $P = \texttt{abbaabbb}$ and $f_T(\texttt{a}) = f_T(\texttt{b}) = 0.5$. As we determine scan$[i]$ for $i = 1, \ldots, m$, we can see that the values of safe$_i[k]$ changes. For example, when scan[1] is set to 4, the values of safe$_1[1]$ and safe$_1[2]$ become 1. As the values of safe$_i[k]$ change (as i increases), the values of avr_shift$_i[l]$ for a fixed l increase as well. Therefore, we need to calculate avr_shift$_i[l]$ again whenever we set the ith scan position. Table 3 shows mas_shift$[l][s]$ after the preprocessing for pattern $P = \texttt{abbaabbb}$ and $f_T(\texttt{a}) = f_T(\texttt{b}) = 0.5$. It can be seen that shift$_i$[scan$[i]$]$[s]$ in Table 2 is equal to mas_shift$[l][s]$ such that scan$[i] = l$ in Table 3.

The time complexity of the preprocessing of MAS is $O(m^2|\Sigma|)$. For fixed l and s, the value of shift$_i[l][s]$ monotonically increases from 1 to m as i increases, and we spend the time proportional to shift$_i[l][s]$ − shift$_{i-1}[l][s]$ to compute shift$_i[l][s]$. For fixed l and s, therefore, computing shift$_i[l][s]$ for all i takes $O(m)$ time. Hence, the total time to compute shift$_i[l][s]$ for all i, l, and s is $O(m^2|\Sigma|)$. Since the computation of safe$_i[k]$ at the ith iteration takes $O(m)$, the total time to compute safe$_i[k]$ is $O(m^2)$. For fixed i and l, computing avr_shift$_i[l]$ requires $O(|\Sigma|)$ time, and so the total time to compute avr_shift$_i[l]$ is $O(m^2|\Sigma|)$. The space complexity of the preprocessing of MAS is $O(m|\Sigma|)$, because shift$[l][s]$ is of $O(m|\Sigma|)$, and scan$[i]$, avr_shift$[l]$, and safe$[k]$ are of $O(m)$.

Fig. 4 shows the string matching algorithm MAS. First, the preprocessing of MAS is performed on a pattern P to compute the pattern scan order scan$[i]$ and the shift length table mas_shift$[l][s]$. Then it places the window at the leftmost position of

Table 4
Pattern scan orders depend on character frequencies.

$f_T(\text{a})$	$f_T(\text{b})$	l	1	2	3	4	5	6	7	8
		$P[l]$	a	b	b	a	a	b	b	b
		$\text{shift}_1[l][\text{a}]$	1	1	2	3	1	1	2	3
		$\text{shift}_1[l][\text{b}]$	1	2	1	1	2	3	1	1
0.75	0.25	$\text{avr\_shift}_1[l]$	1	1.25	1.75	**2.5**	1.25	1.5	1.75	2.5
0.25	0.75	$\text{avr\_shift}_1[l]$	1	1.75	1.25	1.5	1.75	**2.5**	1.25	1.5

Table 5
Scan speed of BM-family algorithms on Human chromosomes 1, 10, and 20.

m	BM	BMH	OM	MS	GS	HS	MAS
4	1.21	1.07	1.37	1.37	1.45	1.24	**2.11**
8	1.78	1.52	1.90	1.94	1.96	1.68	**3.30**
16	2.31	1.79	2.44	2.62	2.62	2.49	**4.84**
32	2.73	1.88	2.93	3.25	3.30	3.56	**6.76**
64	3.26	1.96	3.54	3.94	4.03	4.92	**9.71**
128	3.62	1.94	3.96	4.56	4.82	6.89	**13.25**

Table 6
Running time of BM-family algorithms on Human chromosomes 1, 10, and 20 (in milliseconds per 1,000,000 text characters).

m	BM	BMH	OM	MS	GS	HS	MAS
4	4.91	3.31	3.40	3.41	3.68	4.04	**3.17**
8	3.46	2.55	2.59	2.49	2.80	2.98	**2.18**
16	2.83	2.37	2.15	1.92	2.23	2.12	**1.63**
32	2.48	2.33	1.87	1.60	1.84	1.57	**1.29**
64	2.18	2.29	1.65	1.38	1.60	1.23	**1.04**
128	2.02	2.34	1.54	1.26	1.43	1.01	**0.90**

the text and scans the text in the window according to the pattern scan order. If there is a mismatch in scan position $\text{scan}[i]$, it moves the window to the right by $\text{mas\_shift}[\text{scan}[i]][T[w+\text{scan}[i]]]$. If there are all matches in the current window, it outputs an occurrence of the pattern and it moves the window to the right by $\text{mas\_shift}[\text{scan}[m]][T[w+\text{scan}[m]]]$.

Since MAS selects position l with the largest $\text{avr\_shift}_i[l]$ as the ith scan position, the pattern scan order varies according to the distribution of text characters even for the same pattern. Table 4 shows the values of $\text{avr\_shift}_1[l]$ as the distribution $f_T(s)$ changes for the same pattern $P = \text{abbaabbb}$. When the text distribution is $f_T(\text{a}) = 0.75$ and $f_T(\text{b}) = 0.25$, the largest (and leftmost) entry in $\text{avr\_shift}_1[l]$ is $\text{avr\_shift}_1[4] = 2.5$ and so the first scan position is 4. When the text distribution is $f_T(\text{a}) = 0.25$ and $f_T(\text{b}) = 0.75$, the largest entry is $\text{avr\_shift}_1[6] = 2.5$ and the first scan position is 6.

4. Experimental results of MAS

We compare the performances of BM-family algorithms by experiments. The text used in the experiments is Human chromosomes 1, 10, and 20 downloaded from the 1000 Genomes Project website [27]. The lengths of Human chromosomes 1, 10, and 20 are 249250621, 135534747, and 63025520, but after removing several chunks of character N's in each chromosome, the lengths of the text are 225280621, 131314738, and 59505520, respectively. The probability distribution of characters in the human genome is $f_T(\text{A}) = 0.293$, $f_T(\text{C}) = 0.207$, $f_T(\text{G}) = 0.207$, and $f_T(\text{T}) = 0.293$ [29]. The pattern lengths are 4, 8, 16, 32, 64, and 128, and for each pattern length m, 100 patterns of length m are randomly extracted from the text. For each pattern length, we compute two values in the experiments:

- *Scan speed*: the ratio of the text length to the number of text scanned by an algorithm, averaged over 100 patterns.
- *Running time*: the running time of an algorithm / text length $\times$ 1,000,000 (i.e., running time per 1,000,000 text characters), averaged over 100 patterns.

Experiments are conducted on a machine with an Intel E5300 CPU with 2.60 GHz and 4 GB memory running CentOS. The other BM-family algorithms were obtained from the String Matching Algorithms Research Tool [13].

Table 5 shows the scan speed of BM, BMH, OM, MS, GS, HS, and MAS, averaged over Human chromosomes 1, 10, and 20. MAS shows the best performances in scan speed for all pattern lengths. Specifically, MAS is 45.5% faster than GS when the pattern length is 4, and 92.3% faster than HS when the pattern length is 128 (GS and HS, respectively in each case, show the best performances among previous BM-family algorithms). The general trend is that the longer the pattern length is, the larger is the performance improvement of MAS over other algorithms.

Table 6 shows the running time of BM, BMH, OM, MS, GS, HS, and MAS, averaged over Human chromosomes 1, 10, and 20. Again MAS shows the best performances in running time for all pattern lengths. Table 7 shows the preprocessing time

Table 7
Preprocessing time of HS and MAS on Human chromosomes 1, 10, and 20 (in milliseconds).

m	HS	MAS
4	0.013	0.014
8	0.014	0.020
16	0.017	0.033
32	0.029	0.071
64	0.073	0.209
128	0.237	0.703

Table 8
Scan speed with different criteria for pattern scan order.

		1	2	3	4
priority1		*avr_sh*	*avr_sh*	*avr_sh*	*avr_sh*
priority2		*left*	*right*	*freq*	*freq*
priority3				*left*	*right*
m	4	2.084	2.062	**2.091**	2.078
	8	3.254	3.211	**3.270**	3.241
	16	4.896	4.864	**4.901**	4.888
	32	6.775	6.758	**6.785**	6.768
	64	9.599	9.598	**9.603**	9.598
	128	13.044	13.040	**13.045**	13.044

of HS and MAS, averaged over 100 patterns and then averaged over Human chromosomes 1, 10, and 20. Since the time complexities of the preprocessing of HS and MAS are $O(m^2)$ and $O(m^2|\Sigma|)$, respectively, the preprocessing time of MAS is longer than that of HS. However, the preprocessing time of MAS is less than 1% of the total execution time (preprocessing time + running time) for all pattern lengths.

In addition to the average shift length, we can consider other criteria when we determine the pattern scan order, which are *freq* and *left/right* in the order of decreasing importance.

- *freq*: If the average shift lengths are the same for several positions, we choose the position where the frequency of the pattern character is smaller so that the probability of a mismatch with a text character gets higher. That is, if $\text{avr\_shift}_i[l] = \text{avr\_shift}_i[l']$ for two positions l and l', and $f_T(P[l]) < f_T(P[l'])$, then we choose l as scan$[i]$.
- *left/right*: If other criteria are the same, *left* and *right* mean that we choose the leftmost position and the rightmost position, respectively.

Table 8 shows the scan speed of MAS with combinations of the three criteria for Human chromosome 1, where case 3 shows the best results. Therefore, we used three criteria, *avr_shift*, *freq*, *left*, in this order in our algorithm MAS. Since added criteria, *freq* and *left*, can be checked in line 17 of Algorithm 1 (when finding maximum *avr_shift*), the preprocessing time of MAS does not change.

5. Extending the MAS algorithm

In this section we present two extensions of MAS: one improves the scan speed of MAS by using the scan result of the previous window, and the other improves the running time of MAS by using q-grams.

5.1. Tuned MAS algorithm

The MAS algorithm computes the shift length using the scan result of the current window only. If we use the scan result of the previous window, we can increase the shift length. However, there is a trade-off between the number of text characters scanned in the previous window which we remember and the preprocessing time (and space). Therefore, we will remember only one text character scanned in the previous window. Since an algorithm scans at least one text character in a window, we will remember the first text character scanned in the previous window.

When we shift a window, there are two cases regarding the first scanned text character.

1. The first text character scanned in the previous window is within the current window: Let f be the position in the current window corresponding to the first text character scanned in the previous window. Since the shift length from the previous window to the current window can be any value between 1 and m, f is not a fixed value, and it is one of positions 1 to $m-1$ of the current window (because the shift length is at least 1). Since we will shift the window so that the scanned text characters in the previous window have matches in the current window as in MAS, position f of the current window will have a match. In Fig. 5, the first scan position is 5 and it has a match in the previous window;

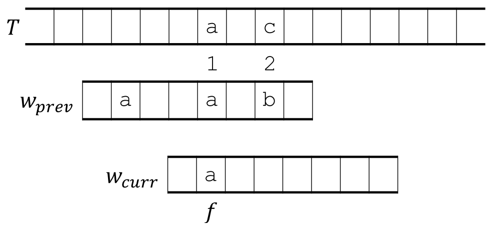

Fig. 5. The position f in the current window.

the second scan position is 7 and it has a mismatch. Then the window is shifted by 3 positions. In the current window, therefore, the value of f is 2.

2. The first text character scanned in the previous window is to the left of the current window: In this case, we set $f = 0$.

Now we describe the tuned MAS (TMAS) algorithm. TMAS will use different scan orders and different shift length tables, depending on the value of f, i.e., it will use scan$[f][i]$ and tmas_shift$[f][l][s]$ for $0 \le f < m$. When $f = 0$, scan$[0][i]$ and tmas_shift$[0][l][s]$ are exactly the same as scan$[i]$ and mas_shift$[l][s]$ of MAS, respectively. When $f \ge 1$, position f of the current window w already has a match (i.e., $P[f] = T[w + f]$). When $f \ge 1$ (i.e., the first text character scanned in the previous window is in position f of the current window), we define scan$[f][i]$, shift$_i[f][l][s]$, avr_shift$_i[f][l]$, and tmas_shift$[f][l][s]$ for each value of f as follows.

- scan$[f][i]$: ith scan position, where $i = 1, 2, \ldots, m - 1$ and scan$[f][i] \neq f$ for any i because position f already has a match.
- shift$_i[f][l][s]$: shift length when scan$[f][i] = l$ and the text character in the ith scan position is s.
- avr_shift$_i[f][l] = \sum_s (f_T(s) \times \text{shift}_i[f][l][s])$.
- tmas_shift$[f][l][s]$: shift length of TMAS when the scan position is l and the text character in the scan position is s.

Like Equation (1), shift$_i[f][l][s]$ can be formulated as follows.

$$\text{shift}_i[f][l][s] = \min\{k \ge 1 \mid P[\text{scan}[f][j]] = P[\text{scan}[f][j] - k] \text{ for all } 1 \le j < i,\ P[f] = P[f - k], \text{ and } s = P[l - k]\}.$$

If we define safe$_i[f][k]$ as follows:

$$\text{safe}_i[f][k] = \begin{cases} 1 & \text{if } \exists j \text{ such that } P[\text{scan}[f][j]] \neq P[\text{scan}[f][j] - k] \text{ for } 1 \le j \le i \text{ or } P[f] \neq P[f - k] \\ 0 & \text{otherwise,} \end{cases}$$

the equation above can be changed to:

$$\text{shift}_i[f][l][s] = \min\{k \ge \text{shift}_{i-1}[f][l][s] \mid \text{safe}_{i-1}[f][k] = 0 \text{ and } s = P[l - k]\}. \tag{4}$$

For $f = 1, 2, \ldots, m - 1$, TMAS does the following. Just as Algorithm 1 of MAS computes shift$_i[l][s]$ using safe$_i[k]$, TMAS computes shift$_i[f][l][s]$ efficiently using safe$_i[f][k]$ by Equation (4). Then, TMAS selects l such that avr_shift$_i[f][l]$ is the largest as scan$[f][i]$ for $i = 1, 2, \ldots, m - 1$. As in MAS, tmas_shift$[f][l][s]$ is computed as follows.

$$\text{tmas\_shift}[f][l][s] = \text{shift}_i[f][l][s] \text{ such that scan}[f][i] = l.$$

For each value of $1 \le f \le m-1$, scan$[f][i]$ is defined for $i = 1, 2, \ldots, m-1$. Hence, the entry scan$[f][m]$ is empty, and so we assign scan$[f][m] = f$ to simplify the text search of TMAS. That is, we make the mth comparison at position f, though it is not necessary because we know that it will be a match. Since TMAS makes a shift as soon as it finds a mismatch, the probability that it will make the mth comparison is very low, while the simplification in the text search will improve the running time of TMAS.

Since scan$[f][m]$ is set to f, we need to give the shift length when TMAS makes the mth comparison. Since the mth comparison will be a match, it is the shift length when the whole pattern matches, which is mas_shift$[\text{scan}[m]][P[\text{scan}[m]]]$ of MAS because it is the shift length when MAS makes the mth comparison and the text character in scan position scan$[m]$ is $P[\text{scan}[m]]$ (i.e., the mth comparison is a match). Since mas_shift$[\text{scan}[m]][P[\text{scan}[m]]]$ of MAS is stored in tmas_shift$[0][\text{scan}[0][m]][P[\text{scan}[0][m]]]$, we assign it to tmas_shift$[f][f][P[f]]$, which is the shift length when TMAS makes the mth comparison.

The preprocessing of TMAS computes the scan order and the shift length table for each value of $0 \le f < m$. Therefore, the space and time complexities of the preprocessing of TMAS are m times those of MAS. Consequently, the space complexity and the time complexity of the preprocessing of TMAS are $O(m^2|\Sigma|)$ and $O(m^3|\Sigma|)$, respectively.

Tables 9 and 10 show part of scan$[f][i]$ and tmas_shift$[f][l][s]$ for pattern $P = \texttt{abbaabbb}$ and $f_T(\texttt{a}) = f_T(\texttt{b}) = 0.5$. We can see that TMAS uses different scan orders and different shift lengths, depending on the value of f. Especially,

Table 9
scan$[f][i]$ when $P =$ abbaabbb and $f_T(\mathrm{a}) = f_T(\mathrm{b}) = 0.5$.

l	1	2	3	4	5	6	7	8
$P[l]$	a	b	b	a	a	b	b	b
scan$[0][i]$	4	6	8	7	1	2	3	5
...				...				
scan$[2][i]$	5	7	6	8	1	3	4	2
scan$[3][i]$	2	6	8	7	1	4	5	3
...				...				

Table 10
tmas_shift$[f][l][s]$ when $P =$ abbaabbb and $f_T(\mathrm{a}) = f_T(\mathrm{b}) = 0.5$.

l	1	2	3	4	5	6	7	8
$P[l]$	a	b	b	a	a	b	b	b
tmas_shift$[0][l]$[a]	8	8	8	3	8	5	6	3
tmas_shift$[0][l]$[b]	8	8	8	1	8	3	8	6
...				...				
tmas_shift$[2][l]$[a]	8	.	8	8	4	5	6	4
tmas_shift$[2][l]$[b]	8	8	8	8	2	4	4	8
tmas_shift$[3][l]$[a]	8	1	.	8	8	5	6	3
tmas_shift$[3][l]$[b]	8	3	8	8	8	3	8	6
...				...				

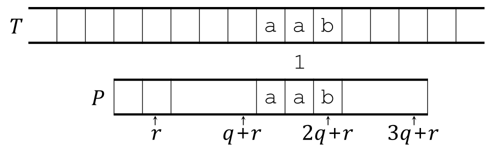

Fig. 6. Division of the pattern into q-grams.

scan$[0][i]$ and tmas_shift$[0][l][s]$ are equal to scan$[i]$ and mas_shift$[l][s]$ of MAS in Tables 2 and 3. When $f \geq 1$, we can see that scan$[f][m]$ is set to f (e.g., scan$[2][8] = 2$ in Table 9) and tmas_shift$[f][f][P[f]]$ is set to tmas_shift$[0][\text{scan}[0][m]]$ $[P[\text{scan}[0][m]]]$ (e.g., tmas_shift$[2][2][\mathrm{b}] =$ tmas_shift$[0][5][\mathrm{a}] = 8$ in Table 10).

The text search of TMAS is essentially the same as that of MAS except for the following differences.

- TMAS maintains variable f. Initially, $f = 0$. When TMAS makes a shift by shift length sh, the new value of f is set to scan$[f][1] - sh$; it is set to 0 if scan$[f][1] - sh < 1$, because scan$[f][1]$ is the first scan position in the previous window and so scan$[f][1] - sh$ is the position in the current window corresponding to scan$[f][1]$ of the previous window.
- scan$[i]$ and mas_shift$[\text{scan}[i]][T[w + \text{scan}[i]]]$ in Algorithm 2 of MAS are replaced by scan$[f][i]$ and tmas_shift$[f][\text{scan}[f][i]][T[w + \text{scan}[f][i]]]$, respectively.

5.2. *q-gram MAS algorithm*

We now extend MAS to the q-gram MAS (QMAS$_q$) algorithm using q-grams. A q-gram is a substring (of P or T) of q characters. Given a q-gram $s_0 s_1 \cdots s_{q-1}$ over Σ, we define its fingerprint $h(s_0 s_1 \cdots s_{q-1})$ as follows:

$$h(s_0 s_1 \cdots s_{q-1}) = \sum_{i=0}^{q-1} |\Sigma|^i \cdot s_i.$$

In this way, a fingerprint represents a character s^q of a larger alphabet Σ^q. Let $f_T^q(s^q)$ for $s^q \in \Sigma^q$ be the probability that a q-gram in T is s^q. When $h(s_0 s_1 \ldots s_{q-1}) = s^q$, $f_T^q(s^q)$ is equal to $f_T(s_0) \cdot f_T(s_1) \cdots f_T(s_{q-1})$ because we assumed that the distributions for all text characters in T are independent and identical. For a string S, let $G_S(l, q)$ denote the q-gram $S[l - q + 1..l]$.

Let $r = m \mod q$. In order to apply the q-gram to MAS, we divide the pattern (or a window) into $\lfloor \frac{m}{q} \rfloor$ q-grams and the rest: $G_P(q + r, q), G_P(2q + r, q), \ldots, G_P(\lfloor \frac{m}{q} \rfloor \cdot q + r, q)$, and the rest $P[1..r]$ as illustrated in Fig. 6. As in MAS, we define the following.

- The scan order is $(\text{scan}^q[1], \text{scan}^q[2], \ldots, \text{scan}^q[\lfloor \frac{m}{q} \rfloor])$, where $\text{scan}^q[i] = l$ means that the ith scan position is l (in terms of q-grams). When we consider $\text{scan}^q[i] = l$, we compare pattern q-gram $G_P(lq + r, q)$ against text q-gram $G_T(w + lq +$

Table 11
First scan position of QMAS$_3$ when $P = \texttt{abbaabbb}$ and $f_T(\texttt{a}) = f_T(\texttt{b}) = 0.5$.

l	1	2
$G_P(lq+r,q)$	baa	bbb
$\text{shift}_1^q[l][h(\texttt{aaa})]$	3	6
$\text{shift}_1^q[l][h(\texttt{aab})]$	3	2
$\text{shift}_1^q[l][h(\texttt{aba})]$	3	6
$\text{shift}_1^q[l][h(\texttt{abb})]$	2	1
$\text{shift}_1^q[l][h(\texttt{baa})]$	3	3
$\text{shift}_1^q[l][h(\texttt{bab})]$	3	6
$\text{shift}_1^q[l][h(\texttt{bba})]$	1	4
$\text{shift}_1^q[l][h(\texttt{bbb})]$	3	6
$\text{avr\_shift}_1^q[l]$	2.625	**4.250**

$r, q)$. If the two q-grams match, we go on to $\text{scan}^q[i+1]$; if there is any mismatch between the two q-grams, we shift the window.

- $\text{shift}_i^q[l][s^q]$: shift length when $\text{scan}^q[i] = l$ and the text q-gram in the ith scan position is s^q.
- $\text{avr\_shift}_i^q[l] = \sum_{s^q}(f_T^q(s^q) \times \text{shift}_i^q[l][s^q])$.
- $\text{qmas\_shift}[l][s^q]$: shift length of QMAS$_q$ when the scan position is l and the text q-gram in the scan position is s^q (i.e., $h(G_T(w + lq + r, q)) = s^q$).

Then, shift_i^q is formulated as follows.

$$\text{shift}_i^q[l][s^q] = \min\{k \geq 1 \mid G_P(\text{scan}^q[j] \cdot q + r, q) = G_P(\text{scan}^q[j] \cdot q + r - k, q) \text{ for all } 1 \leq j < i,$$
$$\text{and } s^q = h(G_P(lq + r - k, q))\},$$

where $G_P(l, q)$ for $l \leq q - 1$ is a special q-gram that matches any q-gram. Note that shift length k can be any integer between 1 and $m - q + 1$ rather than a multiple of q.

QMAS$_q$ selects l such that $\text{avr\_shift}_i^q[l]$ is the largest as $\text{scan}^q[i]$ for $i = 1, 2, \ldots, \lfloor \frac{m}{q} \rfloor$. As in MAS, QMAS$_q$ computes $\text{shift}_i^q[l][s^q]$ using $\text{safe}_i^q[k]$ defined as follows:

$$\text{safe}_i^q[k] = \begin{cases} 1 & \text{if } \exists j \text{ such that } G_P(\text{scan}^q[j] \cdot q + r, q) \neq G_P(\text{scan}^q[j] \cdot q + r - k, q) \text{ for } 1 \leq j \leq i \\ 0 & \text{otherwise.} \end{cases}$$

Table 11 shows an example of QMAS$_3$ when $P = \texttt{abbaabbb}$ and $f_T(\texttt{a}) = f_T(\texttt{b}) = 0.5$. Since $m = 8$ and $q = 3$, the pattern is divided into two q-grams $G_P(5, 3) = \texttt{baa}$ and $G_P(8, 3) = \texttt{bbb}$, and the rest $P[1..2] = \texttt{ab}$. Since $\text{avr\_shift}_1^q[1] = 2.625$ and $\text{avr\_shift}_1^q[2] = 4.250$, we set $\text{scan}^q[1] = 2$. The shift length table $\text{qmas\_shift}[l][s^q]$ of QMAS$_q$ is computed as follows:

$$\text{qmas\_shift}[l][s^q] = \text{shift}_i^q[l][s^q] \text{ such that } \text{scan}^q[i] = l.$$

The time complexity of the preprocessing of QMAS$_q$ is $O(m^2|\Sigma|^q)$. For fixed l and s^q, computing $\text{shift}_i^q[l][s^q]$ for all i takes $O(mq)$ time, because reading a q-gram takes $O(q)$ time. Since the number of l is $O(\frac{m}{q})$, the total time to compute $\text{shift}_i^q[l][s^q]$ for all i, l, and s^q is $O(m^2|\Sigma|^q)$. For fixed i and l, computing $\text{avr\_shift}_i[l]$ requires $O(|\Sigma|^q)$ time, and so the total time to compute $\text{avr\_shift}_i[l]$ is $O(\frac{m^2}{q^2}|\Sigma|^q)$. Therefore, the time complexity is $O(m^2|\Sigma|^q)$. The space complexity of the preprocessing of QMAS$_q$ is $O(m + \frac{m}{q}|\Sigma|^q)$, because $\text{shift}^q[l][s^q]$ is of $O(\frac{m}{q}|\Sigma|^q)$, $\text{scan}^q[i]$ and $\text{avr\_shift}^q[l]$ are of $O(\frac{m}{q})$, and $\text{safe}^q[k]$ is of $O(m)$.

The text search of QMAS$_q$ is essentially the same as that of MAS except that it works by q-grams rather than by characters. Initially, it computes $r = m \mod q$. If all of $\lfloor \frac{m}{q} \rfloor$ q-grams of the pattern match against the text, it compares the rest $P[1..r]$ against $T[w+1..w+r]$ character by character. When it moves the window, the window is shifted to the right by $\text{qmas\_shift}[\text{scan}^q[i]][h(G_T(w + \text{scan}^q[i] \cdot q + r, q))]$.

5.3. Experimental results

We compare the performances of state-of-the-art non-BM-family algorithms by experiments. We selected the algorithms that show good performances for DNA sequences in the review papers [20,12], and obtained the source codes from the String Matching Algorithms Research Tool [13]. All experimental settings are the same as in Section 4.

Table 12
Scan speed of non-BM-family algorithms on Human chromosomes 1, 10, and 20.

m	HASH_3	HASH_5	BOM	EBOM	TRF	SVM	MAS	TMAS	QMAS_2	QMAS_4
4	0.65	–	1.86	1.23	1.94	2.22	2.11	**2.29**	1.29	–
8	1.86	0.79	3.15	2.56	3.23	**3.84**	3.30	3.82	2.67	1.23
16	3.97	2.32	5.25	4.70	5.46	**6.64**	4.84	6.29	4.66	3.13
32	7.25	5.17	8.80	8.24	9.33	–	6.76	**10.11**	7.12	6.73
64	11.61	10.21	15.07	14.46	16.25	–	9.71	**16.26**	10.00	13.20
128	15.77	18.13	26.23	25.56	**28.76**	–	13.25	24.91	13.39	23.74

Table 13
Running time of non-BM-family algorithms on Human chromosomes 1, 10, and 20 (in milliseconds per 1,000,000 text characters).

m	HASH_3	HASH_5	BOM	EBOM	TRF	SVM	MAS	TMAS	QMAS_2	QMAS_4
4	3.32	–	5.88	**2.26**	5.48	3.60	3.17	4.44	2.66	–
8	**1.41**	1.74	3.80	1.71	3.19	2.25	2.18	2.83	1.48	1.71
16	0.87	**0.85**	2.50	1.23	1.97	1.50	1.63	1.98	1.03	0.90
32	0.68	**0.62**	1.69	0.89	1.34	–	1.29	1.48	0.85	0.64
64	0.62	**0.60**	1.19	0.69	0.96	–	1.04	1.21	0.75	**0.60**
128	**0.62**	0.65	1.01	0.66	0.92	–	0.90	1.10	0.70	0.66

Table 14
Preprocessing time of HASH_5 and QMAS_4 on Human chromosomes 1, 10, and 20 (in milliseconds).

m	HASH_5	QMAS_4
4	–	–
8	0.012	0.025
16	0.012	0.064
32	0.012	0.226
64	0.012	0.869
128	0.012	3.166

Table 12 shows the scan speed of the algorithms, averaged over Human chromosomes 1, 10, and 20. TMAS shows the best performances among the algorithms, except SVM which remembers the scan results of all previous windows by using the bit-parallel approach, when pattern length m is 64 or less. Since SVM uses the bit parallelism and it can use 31 bits in the 32-bit word, it cannot run when m is 32 or more. Although TMAS remembers only one character in the previous window, it shows performances comparable to SVM when m is 16 or less. TRF is very close to TMAS when $m = 64$, and TRF is the best performer in scan speed when $m = 128$.

Table 13 shows the running time of the algorithms, averaged over Human chromosomes 1, 10, and 20. When pattern length m is 4, EBOM is the fastest. When m is 8 or more, HASH_q is the best performer, and QMAS_q is the runner-up in general. Note that QMAS_4 matches HASH_5 when $m = 64$. While QMAS_4 has better scan speeds than HASH_5, HASH_5 is faster than QMAS_4 in running time due to its simplicity. Table 14 shows the preprocessing time of HASH_5 and QMAS_4. Since the time complexities of the preprocessing of HASH_5 and QMAS_4 are $O(mq)$ and $O(m^2|\Sigma|^q)$, respectively, the preprocessing time of QMAS_4 is much longer than that of HASH_5. When the pattern lengths are 64 and 128, the preprocessing times of QMAS_4 are 1.37% and 4.33%, respectively, of the total execution time (preprocessing time + running time).

6. Conclusion

We have proposed the Maximal Average Shift (MAS) algorithm that finds a pattern scan order that maximizes the average shift length. Experiments show that MAS outperforms other BM family algorithms in both scan speed and running time. Additionally, we presented two extensions of MAS: TMAS improves the scan speed of MAS by using the scan result of the previous window, and QMAS_q improves the running time of MAS by using q-grams. In particular, TMAS shows the best performances in general among non-BM-family algorithms in scan speed.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

Ryu and Park were supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2018-0-00551, Framework of Practical Algorithms for NP-hard Graph Problems).

References

[1] C. Allauzen, M. Crochemore, M. Raffinot, Factor oracle: a new structure for pattern matching, in: SOFSEM'99: Theory and Practice of Informatics, in: Lecture Notes in Computer Science, vol. 1725, Springer, Berlin, Heidelberg, 1999, pp. 295–310.
[2] R.A. Baeza-Yates, G.H. Gonnet, M. Régnier, Analysis of Boyer-Moore-type string searching algorithms, in: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 1990, pp. 328–343.
[3] R.A. Baeza-Yates, M. Régnier, Average running time of the Boyer-Moore-Horspool algorithm, Theor. Comput. Sci. 92 (1) (1992) 19–31.
[4] R.S. Boyer, J.S. Moore, A fast string searching algorithm, Commun. ACM 20 (10) (1977) 762–772.
[5] C. Charras, T. Lecroq, Handbook of Exact String Matching Algorithms, King's College Publications, 2004.
[6] R. Cole, R. Hariharan, Tighter upper bounds on the exact complexity of string matching, SIAM J. Comput. 26 (3) (1997) 803–856.
[7] L. Colussi, Correctness and efficiency of pattern matching algorithms, Inf. Comput. 95 (2) (1991) 225–251.
[8] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd edition, The MIT Press, 2009.
[9] M. Crochemore, A. Czumaj, L. Gasieniec, S. Jarominek, T. Lecroq, W. Plandowski, W. Rytter, Speeding up two string-matching algorithms, Algorithmica 12 (4) (1994) 247–267.
[10] G. Didier, L. Tichit, Designing optimal- and fast-on-average pattern matching algorithms, J. Discret. Algorithms 42 (Suppl. C) (2017) 45–60.
[11] S. Faro, T. Lecroq, Efficient variants of the backward-oracle-matching algorithm, Int. J. Found. Comput. Sci. 20 (6) (2009) 967–984.
[12] S. Faro, T. Lecroq, The exact online string matching problem: a review of the most recent results, ACM Comput. Surv. 45 (2) (2013) 13.
[13] S. Faro, T. Lecroq, S. Borzì, S.D. Mauro, A. Maggio, The string matching algorithms research tool, in: Proceedings of the Prague Stringology Conference 2016, Prague Stringology Club, 2016, pp. 99–113.
[14] Z. Galil, R. Giancarlo, On the exact complexity of string matching: upper bounds, SIAM J. Comput. 21 (3) (1992) 407–437.
[15] A. Gittleman, Predicting string search speed, J. Exp. Algorithmics 1 (1996) 1–11.
[16] R.N. Horspool, Practical fast searching in strings, Softw. Pract. Exp. 10 (6) (1980) 501–506.
[17] A. Hume, D. Sunday, Fast string searching, Softw. Pract. Exp. 21 (11) (1991) 1221–1248.
[18] D.E. Knuth, J.H. Morris Jr., V.R. Pratt, Fast pattern matching in strings, SIAM J. Comput. 6 (2) (1977) 323–350.
[19] M.O. Külekci, An empirical analysis of pattern scan order in pattern matching, in: Proceedings of the World Congress on Engineering, 2007, pp. 337–341.
[20] T. Lecroq, Experimental results on string matching algorithms, Softw. Pract. Exp. 25 (7) (1995) 727–765.
[21] T. Lecroq, Fast exact string matching algorithms, Inf. Process. Lett. 102 (6) (2007) 229–235.
[22] C.W. Lu, C.L. Lu, R.C.T. Lee, Improved exact string matching algorithms based upon selective matching order and branch and bound approach, in: Proceedings of the 31st Workshop on Combinatorial Mathematics and Computation Theory, 2014, pp. 19–28.
[23] T. Marschall, S. Rahmann, Exact analysis of Horspool's and Sunday's pattern matching algorithms with probabilistic arithmetic automata, in: Language and Automata Theory and Applications, in: Lecture Notes in Computer Science, vol. 6031, Springer, Berlin, Heidelberg, 2010, pp. 439–450.
[24] H. Peltola, J. Tarhio, Alternative algorithms for bit-parallel string matching, in: String Processing and Information Retrieval, in: Lecture Notes in Computer Science, vol. 2857, Springer, Berlin, Heidelberg, 2003, pp. 80–93.
[25] C. Ryu, K. Park, Improved pattern-scan-order algorithms for string matching, J. Discret. Algorithms 49 (2018) 27–36.
[26] D.M. Sunday, A very fast substring search algorithm, Commun. ACM 33 (8) (1990) 132–142.
[27] The 1000 Genomes Project Consortium, The 1000 genomes project, http://www.1000genomes.org, 2010.
[28] B. Watson, G. Zwaan, A Taxonomy of Keyword Pattern Matching Algorithms, Technical Report, No. 27, Faculty of Computing Science, Eindhoven University of Technology, 1992.
[29] M.E.B. Yamagishi, A.I. Shimabukuro, Nucleotide frequencies in human genome and Fibonacci numbers, Bull. Math. Biol. 70 (3) (2008) 643–653.