Interdisciplinary Information Sciences Vol. 21, No. 1 (2015) 37–47
©Graduate School of Information Sciences, Tohoku University
ISSN 1340-9050 print/1347-6157 online
DOI 10.4036/iis.2015.37

Filtering Multi-set Tree: Data Structure for Flexible Matching Using Multi-track Data

Kazuyuki NARISAWA*, Takashi KATSURA, Hiroyuki OTA and Ayumi SHINOHARA

Graduate School of Information Sciences, Tohoku University, Sendai 980-8579, Japan

Received May 17, 2014; final version accepted October 1, 2014

Multi-track data are multi-set sequences that are suitable for representing time series data, such as multi-sensor data, polyphonic music data and traffic data. The permuted pattern matching problem aims to determine the occurrences of multi-track patterns in multi-track text by allowing the order of the pattern tracks to be permuted. In this study, we address permuted pattern matching by proposing a new data structure called a filtering multi-set tree (FILM tree). The FILM tree is a complete binary tree based on a spectral Bloom filter (SBF) with hash functions. This data structure is very simple but powerful, and it can be applied to both exact and approximate matching problems. We present experimental results that demonstrate the efficiency of our FILM tree-based approach.

KEYWORDS: data structure, multi-track, permuted pattern matching

1. Introduction

The rapid development of sensor devices, means that various types of traffic information can be obtained easily from multiple places at the same time and stored as time series data. The analysis of these traffic data provides many benefits, which are of great importance. For example, the detection of traffic jam facilitates the selection of alternative routes to a destination that avoid traffic jams. In addition, classification of normal and abnormal behaviors of cars and people based on previous data may allow the development of evacuation maps for emergency situations such as earthquakes. Analysis of traffic information can determine important relationships among data. For example, if a traffic accident occurs, it will cause traffic jams in the surrounding area. To discover the relationships among various places, the associations among information from several places should be regarded as a pattern. The traffic information for specific place is generally represented as time series data. Therefore, we need to consider methods that allow multiple datasets to be treated as single data types.

A suitable method is permuted pattern matching, which was proposed recently by Katsura *et al.* [7]. Permuted matching allows strings, called tracks, to be swapped in multi-track strings. For example, let us consider a multi-track text $\mathbb{T} = \begin{pmatrix} t_1, \\ t_2, \\ t_3 \end{pmatrix} = \begin{pmatrix} \texttt{ababa}, \\ \texttt{aabbb}, \\ \texttt{bbaab} \end{pmatrix}$ and a multi-track pattern $\mathbb{P} = \begin{pmatrix} p_1, \\ p_2 \end{pmatrix} = \begin{pmatrix} \texttt{ab}, \\ \texttt{bb} \end{pmatrix}$. In standard two-dimensional pattern matching, the pattern $\mathbb{P}$ only matches $(t_1[3:4], t_2[3:4])$. In permuted pattern matching, however, $\mathbb{P}$ matches positions 1, 3, and 4, i.e., $(t_1[1:2], t_3[1:2])$, $(t_1[3:4], t_2[3:4])$, and $(t_2[4:5], t_1[4:5])$. If the number M of the tracks of the pattern is equal to that of the text N, this is called a *full-permuted matching problem*, whereas if $M < N$, it is a *sub-permuted matching problem*. Various algorithms and data structures have been proposed for permuted pattern matching problems, which depend on the specific situations (see Table 1). The paper [7] proposed algorithms based on three data structures: (1) the multi-track suffix tree (MTST) that is an indexing structure for full-permuted pattern matching with a fixed multi-track text, (2) the Aho-Corasick automaton (AC automaton) that is suitable for sub-permuted pattern matching with a fixed multi-track pattern, and (3) the generalized suffix array (GSA) that is suitable for sub- and full-permuted pattern matching problems if both the multi-track text and multi-track pattern are fixed. Moreover, we can consider easily an algorithm employing the generalized suffix tree (GST) that is applicable to the sub- and full-permuted pattern matching problems with fixed multi-track text. These algorithms are efficient for *exact* permuted pattern matching problems. However, they are not suitable for handling numerical sequence data, for which *approximate* matching is indispensable in practical applications.

In the present study, we propose a simple and powerful data structure that we refer to as a *filtering multi-set tree (FILM tree)*. The FILM tree is a complete binary tree based on spectral Bloom filter (SBF) [3], which utilizes hash functions. Using this data structure, we can solve permuted pattern matching problems for versions of all combinations

2010 Mathematics Subject Classification: Primary 62F12, Secondary 62F05.
This work is supported by JSPS KAKENHI Grant Numbers 23300051 and 25240003.
*Corresponding author. E-mail: narisawa@ecei.tohoku.ac.jp

Table 1. Comparisons of algorithms using data structures for permuted-pattern matchings.

Data structure	Permutation type				Constructed for
	String		Numerical		
	sub	full	sub	full	
MTST [7]		✓			text
AC automaton [7]	✓	✓			pattern
GSA [7]	✓	✓			text and pattern
GST	✓	✓			text
FILM tree (this present study)	✓	✓	✓	✓	text

exact/approximate and sub-/full-permutation. We demonstrate the performance of our approach experimentally for approximate permuted pattern matching problems with some hash functions.

2. Notations

Let Σ be a finite set of symbols, which is called an *alphabet*. An element of Σ^* is called a *string*. For a string w, $|w|$ denotes the length of w, and $w[i]$ denotes the i-th symbol of w for $1 \leq i \leq |w|$. The empty string is denoted by ε, that is, $|\varepsilon| = 0$. Let $x \cdot y$, briefly denote by xy, be the *concatenation* of strings x and y. Then, $w = w[1]w[2] \dots w[|w|]$. Let Σ^n be the set of strings of length n. For a string $w = xyz$, strings x, y, and z denote the *prefix*, *substring*, and *suffix* of w, respectively. The substring of w that begins at position i and ends at position j is denoted by $w[i:j]$ for $1 \leq i \leq j \leq |w|$, i.e., $w[i:j] = w[i]\ w[i+1] \cdots w[j]$.

Let $\mathbf{\Sigma}_N$ be the set of all N-tuples $(a_1, a_2, \dots, a_N)$ with $a_i \in \Sigma$ for $1 \leq i \leq N$, which is called a *multi-track alphabet*. An element of $\mathbf{\Sigma}_N$ is called a *multi-track character* (mt-character), and an element of $\mathbf{\Sigma}_N^*$ is called a *multi-track string* (or simply *multi-track*), where the concatenation of two multi-track strings is defined as $(a_1, a_2, \dots, a_N) \cdot (b_1, b_2, \dots, b_N) = (a_1b_1, a_2b_2, \dots, a_Nb_N)$. For a multi-track $\mathbb{T} = (t_1, t_2, \dots, t_N) \in \mathbf{\Sigma}_N^n$, the i-th element t_i of $\mathbb{T}$ is called the *i-th track*, the length of multi-track $\mathbb{T}$ is denoted by $|\mathbb{T}|_{len} = |t_1| = |t_2| = \cdots = |t_N| = n$, and the number of tracks in multi-track $\mathbb{T}$ or the *track count* of $\mathbb{T}$, is denoted by $|\mathbb{T}|_{num} = N$. Let $\mathbf{\Sigma}_N^+$ be the set of all multi-tracks of length at least 1, and let $\mathbb{E}_N = (\varepsilon, \varepsilon, \dots, \varepsilon)$ denote the *empty multi-track of track count N*. Then, $\mathbf{\Sigma}_N^* = \mathbf{\Sigma}_N^+ \cup \{\mathbb{E}_N\}$. For a multi-track $\mathbb{T} = \mathbb{XYZ}$, multi-track $\mathbb{X}$, $\mathbb{Y}$, and $\mathbb{Z}$ represent the *prefix*, *substring*, and *suffix* of $\mathbb{T}$, respectively. $\mathbb{T}[i]$ denotes the i-th mt-character of $\mathbb{T}$ for $1 \leq i \leq |\mathbb{T}|_{len}$, i.e., $\mathbb{T} = \mathbb{T}[1]\mathbb{T}[2] \dots \mathbb{T}[|\mathbb{T}|_{len}]$. The substring of $\mathbb{T}$ that begins at position i and ends at position j is denoted by $\mathbb{T}[i:j] = (t_1[i:j], t_2[i:j], \dots, t_N[i:j])$ for $1 \leq i \leq j \leq |\mathbb{T}|_{len}$.

Definition 2.1 (Permuted multi-track)**.** Let $\mathbb{X} = (x_1, x_2, \dots, x_N)$ be a multi-track of track count N. Let $\mathbf{r} = (r_1, r_2, \dots, r_K)$ be a sub-permutation of $(1, \dots, N)$, where $1 \leq K \leq N$. A *permuted multi-track of* $\mathbb{X}$ *specified by* $\mathbf{r}$ is a multi-track $(x_{r_1}, x_{r_2}, \dots, x_{r_K})$, which is denoted by either $\mathbb{X}\langle r_1, r_2, \dots, r_K\rangle$ or $\mathbb{X}\langle \mathbf{r}\rangle$. If $K = N$, $\mathbf{r}$ is called a *full-permutation* and $\mathbb{X}\langle \mathbf{r}\rangle$ is called a *full-permuted multi-track* of $\mathbb{X}$. If $K < N$, $\mathbf{r}$ is called a *sub-permutation* and $\mathbb{X}\langle \mathbf{r}\rangle$ is called a *sub-permuted multi-track* of $\mathbb{X}$.

Definition 2.2 (Permuted-match)**.** For any multi-tracks $\mathbb{X} = (x_1, x_2, \dots, x_{|\mathbb{X}|_{num}})$ and $\mathbb{Y} = (y_1, y_2, \dots, y_{|\mathbb{Y}|_{num}})$ with $|\mathbb{X}|_{len} = |\mathbb{Y}|_{len}$ and $|\mathbb{X}|_{num} \leq |\mathbb{Y}|_{num}$, we say that $\mathbb{X}$ *permuted-matches* $\mathbb{Y}$, which is denoted by $\mathbb{X} \overset{\bowtie}{\sqsubseteq} \mathbb{Y}$, if $\mathbb{X} = \mathbb{Y}'$ for some permuted multi-track $\mathbb{Y}'$ of $\mathbb{Y}$. In particular, if $|\mathbb{X}|_{num} = |\mathbb{Y}|_{num}$, then we say that $\mathbb{X}$ *full-permuted-matches* $\mathbb{Y}$, which we denote by $\mathbb{X} \overset{\bowtie}{=} \mathbb{Y}$. Otherwise, i.e., if $|\mathbb{X}|_{num} < |\mathbb{Y}|_{num}$, then we say that $\mathbb{X}$ *sub-permuted-matches* $\mathbb{Y}$.

Figure 1 shows examples of numerical multi-tracks and a permuted multi-track. Consider a multi-track $\mathbb{T}$ and a multi-track pattern $\mathbb{P}$ in (a) and (d) of Fig. 1. Since $t_2[2:4] = p_2$ and $t_3[2:4] = p_1$, we can say that multi-track pattern $\mathbb{P}$ sub-permuted-matches $\mathbb{T}[2:4]$.

The problem we consider is formally defined as follows.

Problem 2.3 (Permuted pattern matching problem)**.** Given multi-tracks $\mathbb{T}$ and $\mathbb{P}$, output all positions i that satisfy $\mathbb{P} \overset{\bowtie}{\sqsubseteq} \mathbb{T}[i : i + |\mathbb{P}|_{len} - 1]$.

When $|\mathbb{T}|_{num} = |\mathbb{P}|_{num}$, the problem is called the *full*-permuted pattern matching problem, and when $|\mathbb{T}|_{num} < |\mathbb{P}|_{num}$, it is called the *sub*-permuted pattern matching problem.

3. Approximate Permuted Pattern Matching

We extend the concept of multi-track strings to numerical data, by extending the domain Σ of the finite set of symbols to the set $\mathcal{R}$ of real numbers. We say that a multi-track $\mathbb{T} = (t_1, \dots, t_N)$ is a *multi-track numerical sequence* or *numerical multi-track* if each entry $t_i[j]$ is a numerical value in $\mathcal{R}$. In this study, we address the approximate matching

Algorithm 1: Naive algorithm for permuted pattern matching problems

```
    Input: multi-track text $\mathbb{T}$, multi-track pattern $\mathbb{P}$ (and criterion $\delta$)
    Output: matching positions of $\mathbb{P}$ in $\mathbb{T}$
1   $n = |\mathbb{T}|_{len}$; $N = |\mathbb{T}|_{num}$; $m = |\mathbb{P}|_{len}$; $M = |\mathbb{P}|_{num}$;
2   for $i = 1$ to $n - m + 1$ do
3       foreach $\mathbf{r} \in \Pi_{N,M}$ do
4           if $\mathbb{T}[i : i + m - 1]\langle\mathbf{r}\rangle$ and $\mathbb{P}$ (approximate) permuted-match then
5               output position $i$;
6               break
7           end
8       end
9   end
```

$$\mathbb{T} = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} = \begin{pmatrix} \mathbf{5} & \mathbf{6} & \mathbf{8} & \mathbf{4} \\ \mathbf{3} & \mathbf{5} & \mathbf{6} & \mathbf{8} \\ \mathbf{2} & \mathbf{4} & \mathbf{5} & \mathbf{7} \end{pmatrix}$$

(a) Multi-track $\mathbb{T}$

$$\mathbb{T}[2:3] = \begin{pmatrix} t_1[2:3] \\ t_2[2:3] \\ t_3[2:3] \end{pmatrix} = \begin{pmatrix} \mathbf{6} & \mathbf{8} \\ \mathbf{5} & \mathbf{6} \\ \mathbf{4} & \mathbf{5} \end{pmatrix}$$

(b) Substring $\mathbb{T}[2:3]$

$$\mathbb{T}\langle\mathbf{r}\rangle = \begin{pmatrix} t_2 \\ t_1 \\ t_3 \end{pmatrix} = \begin{pmatrix} \mathbf{3} & \mathbf{5} & \mathbf{6} & \mathbf{8} \\ \mathbf{5} & \mathbf{6} & \mathbf{8} & \mathbf{4} \\ \mathbf{2} & \mathbf{4} & \mathbf{5} & \mathbf{7} \end{pmatrix}$$

(c) Permuted multi-track $\mathbb{T}\langle\mathbf{r}\rangle$ by $\mathbf{r} = (2, 1, 3)$

$$\mathbb{P} = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \begin{pmatrix} \mathbf{4} & \mathbf{5} & \mathbf{7} \\ \mathbf{5} & \mathbf{6} & \mathbf{8} \end{pmatrix}$$

(d) Multi-track pattern $\mathbb{P}$

Fig. 1. Examples of multi-tracks.

problem for multi-track numerical sequences, which is defined in a metric space with a distance function D for multi-tracks as follows.

Problem 3.1 (Approximate permuted pattern matching problem)**.** Given a numerical multi-track text $\mathbb{T}$, numerical multi-track pattern $\mathbb{P}$, and a criterion $\delta \geq 0$, output all positions i that satisfy $D(\mathbb{T}[i : i + m - 1]\langle\mathbf{r}\rangle, \mathbb{P}) \leq \delta$, where $\mathbf{r} = (r_1, r_2, \ldots, r_K)$ is a sub-permutation of $(1, \ldots, N)$.

In this problem, we must specify the distance function $D(\mathbb{X}, \mathbb{Y})$ for multi-tracks in order to measure the similarity of $\mathbb{X}$ and $\mathbb{Y}$. Several distance functions have been proposed for numerical data in previous studies. For example, the dynamic time warping (DTW) [8] is often used for time series data. In the present study, we focus on a metric based on the Euclidean distance.

Definition 3.2 (Euclidean distance)**.** For two numerical tracks t_1 and t_2 of the same length n, the *Euclidean distance* $d(t_1, t_2)$ between t_1 and t_2 is defined by

$$d(t_1, t_2) = \sqrt{\sum_{i=1}^{n} (t_1[i] - t_2[i])^2}.$$

Next, we define the Euclidean distance for multi-track numerical sequences.

Definition 3.3 (Multi-track Euclidean distance)**.** For two numerical multi-tracks $\mathbb{X} = (x_1, x_2, \cdots, x_N)$ and $\mathbb{Y} = (y_1, y_2, \cdots, y_N)$ where the length of x_i and y_i is n, the *multi-track Euclidean distance* is defined by

$$D(\mathbb{X}, \mathbb{Y}) = \sum_{j=1}^{N} d(x_j, y_j).$$

If $M < N$, then Problem 3.1 is called the *approximate sub-permuted pattern matching problem*, whereas if $M = N$, it is called the *approximate full-permuted pattern matching problem*. For example, consider the approximate permuted pattern matching problem in Fig. 1. For the multi-track text $\mathbb{T}$ in Fig. 1(a) and the multi-track pattern $\mathbb{P}$ in Fig. 1(d), and given the criterion $\delta = 2$, the solution is positions 1 and 2 because $D(\mathbb{T}[1:3]\langle 2, 1\rangle, \mathbb{P}) = \sqrt{2}$ and $D(\mathbb{T}[2:4]\langle 3, 2\rangle, \mathbb{P}) = 0$.

We present a naive algorithm for permuted pattern matching problems in Algorithm 1. This algorithm computes all of the positions that $\mathbb{P}$ permuted- or approximate permuted-match with $\mathbb{T}[i : i + m - 1]\langle\mathbf{r}\rangle$. The computation in line 4 of Algorithm 1 requires $O(mM)$ time. This algorithm has two loops, where the outside loop requires $O(n)$ and the inside loop requires $O(\frac{N!}{(N-M)!})$. Thus, Algorithm 1 runs in $O(nmM\frac{N!}{(N-M)!})$ time. For Problem 2.3 and 3.1, the output

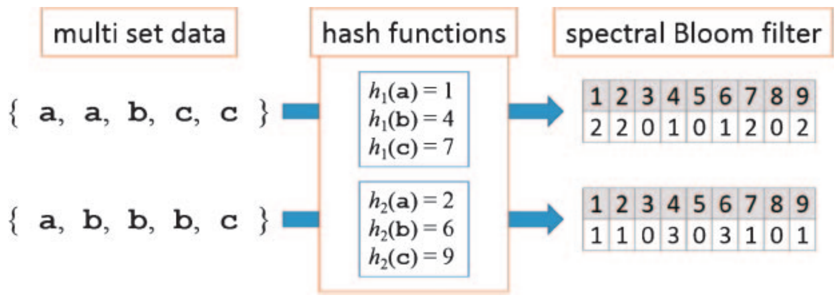


Fig. 2. Spectral Bloom filters $SBF_H(Q_1)$ and $SBF_H(Q_2)$ using $H = \{h_1, h_2\}$ for $Q_1 = \{\mathsf{a}, \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{c}\}$ and $Q_2 = \{\mathsf{a}, \mathsf{b}, \mathsf{b}, \mathsf{b}, \mathsf{c}\}$, respectively.

permutations $\mathbf{r}$ require $O(n\frac{N!}{(N-M)!})$ time, because the number of candidates is $(n-m)\frac{N!}{(N-M)!}$.

As a slightly more efficient approach, we note an algorithm that reduces the permuted pattern matching problem to the minimum weight bipartite matching problem. For each position i on multi-track text $\mathbb{T}$, we consider a weighted bipartite graph $G_i = (A \cup B, A \times B)$, where $A = \{1, \ldots, N\}$, $B = \{1, \ldots, M\}$, and the weight of an edge $(j,k) \in A \times B$ is the distance $d(t_j[i : i+m-1], p_k[1 : m])$. The minimum weight perfect bipartite matching on G_i corresponds to the distance $D(\mathbb{T}[i : i+m-1]\langle\mathbf{r}\rangle, \mathbb{P})$ for the best choice of the permutation $\mathbf{r}$. Because the construction of G_i for each position i requires $O(mNM)$ time, and the minimum weighted bipartite matching for G_i can be found in $O(M(N+M)^2)$ time by the minimum cost flow algorithm [9, 10], the total time of this method is $O(nM(mN + (N+M)^2))$. In the next section, we propose a more efficient method based on SBF and hash functions.

4. FILM Tree

In this section, we propose a new data structure *FILM tree* for solving permuted pattern matching problems in an efficient manner. The matching using the FILM tree is based on the following proposition.

Proposition 4.1. *For multi-tracks* $\mathbb{X} = (x_1, x_2, \ldots, x_{|\mathbb{X}|_{num}})$ *and* $\mathbb{Y} = (y_1, y_2, \ldots, y_{|\mathbb{Y}|_{num}})$, *let* X *and* Y *be multi-sets* $X = \{x_1, x_2, \ldots, x_{|\mathbb{X}|_{num}}\}$ *and* $Y = \{y_1, y_2, \ldots, y_{|\mathbb{Y}|_{num}}\}$. $\mathbb{X} \overset{\bowtie}{\sqsubseteq} \mathbb{Y}$ *if and only if* $X \subseteq Y$.

Proposition 4.1 implies that we can determine whether $\mathbb{P} \overset{\bowtie}{\sqsubseteq} \mathbb{T}[i : i+m-1]$ at position i or not by checking $\{p_1, \ldots, p_M\} \subseteq \{t_1[i : i+m-1], \ldots, t_N[i : i+m-1]\}$. For convenience, we treat a multi-track as a multi-set in the following. In order to verify the multi-set inclusion relation efficiently, we use the spectral Bloom Filter (SBF) [3] that is a data structure constructed for a given multi-set by using hash functions. Since the FILM tree is based on the SBF, we describe the SBF at first in Section 4.1. Then, we provide a definition of the FILM tree in Section 4.2. The choice of hash functions depends on the types of the permuted matching problem, thus we consider them in detail in Section 5.

4.1 Spectral Bloom filter

A spectral Bloom filter (SBF) for a multi-set Q is a data structure that answers the query of whether $R \subseteq Q$ for any given multi-set R.

Definition 4.2 (SBF [3]). Let $H = \{h_1, h_2, \cdots, h_k\}$ be a set of k hash functions, $h_i : U \to \{0, \cdots, u-1\}$, where the domain U is either Σ or $\mathscr{R}$. Let Q be a multi-set over U. A *spectral Bloom filter* $SBF_H(Q)$ *for* Q *using* H is an integer array of length ω, defined by $SBF_H(Q) = (C_Q[1], C_Q[2], \cdots, C_Q[\omega])$ such that

$$C_Q[i] = \sum_{h \in H} \sum_{q \in Q} [\![h(q)(\bmod \omega) + 1 = i]\!],$$

where $[\![P]\!]$ is 1 if the predicate P is true and 0 otherwise.

For example, Fig. 2 shows two SBFs using two hash functions.

Definition 4.3. For two SBFs, $SBF_H(Q_1)$ and $SBF_H(Q_2)$ of the same size ω, we define their *addition* and *subtraction* by

$$SBF_H(Q_1) \oplus SBF_H(Q_2) = (C_{Q_1}[1] + C_{Q_2}[1], C_{Q_1}[2] + C_{Q_2}[2], \cdots, C_{Q_1}[\omega] + C_{Q_2}[\omega]),$$
$$SBF_H(Q_1) \ominus SBF_H(Q_2) = (C_{Q_1}[1] - C_{Q_2}[1], C_{Q_1}[2] - C_{Q_2}[2], \cdots, C_{Q_1}[\omega] - C_{Q_2}[\omega]).$$

We can easily verify the following properties.

Property 4.4. *For any two multi-sets* Q *and* R,
(1) $SBF_H(Q) \oplus SBF_H(R) = SBF_H(Q \cup R)$,
(2) *if* $R \subseteq Q$, *then* $\min(SBF_H(Q) \ominus SBF_H(R)) \geq 0$.

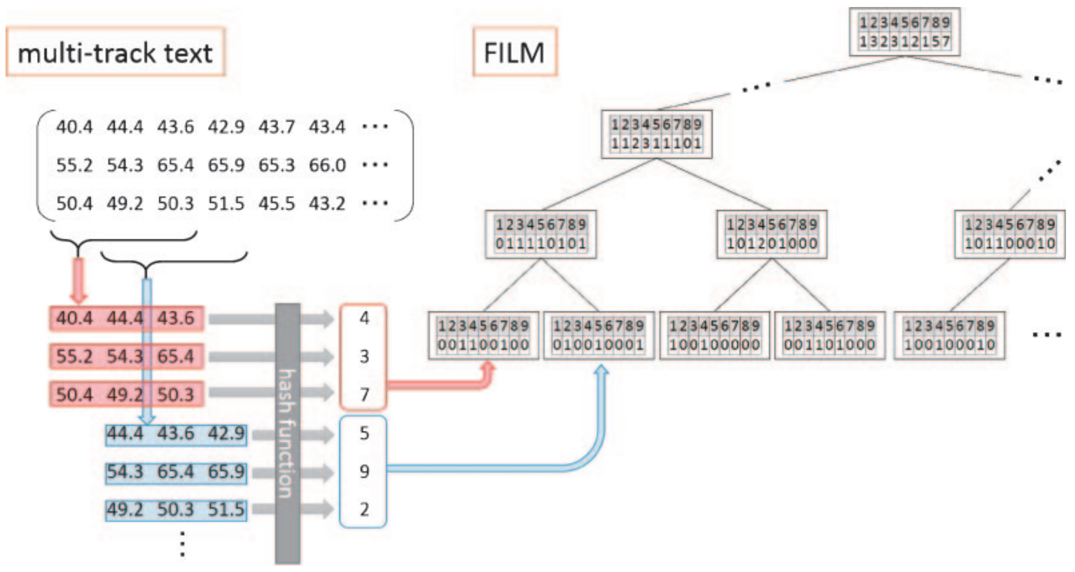


Fig. 3. Example of a FILM tree, using only one hash function ($k = 1$).

We note that the converse of (2) in Property 4.4 does not always hold. For instance, let us consider $Q = \{\mathtt{a}, \mathtt{b}\}$, $R = \{\mathtt{c}\}$ and $H = \{h_1, h_2\}$ such that $h_1(\mathtt{a}) = 1$, $h_2(\mathtt{a}) = 2$, $h_1(\mathtt{b}) = 3$, $h_2(\mathtt{b}) = 4$, $h_1(\mathtt{c}) = 1$, $h_2(\mathtt{c}) = 3$. Then we have $SBF_H(Q) = (1, 1, 1, 1)$ and $SBF_H(R) = (1, 0, 1, 0)$, which yield that $\min(SBF_H(Q) \ominus SBF_H(R)) = 0$, although $R \not\subseteq Q$. Therefore, if we use the condition $\min(SBF_H(Q) \ominus SBF_H(Q')) \geq 0$ to reply a subset query $Q' \subseteq Q$, we may encounter a *false positive*.

We now estimate the probability E_{sbf} that a false positive occurs, assuming that every hash function in H outputs each value with equal probability, and Q contains s elements. Let f_a be the number of occurrences of $a \in Q$, and $\min_a = \min\{C_Q[h_1(a)], \ldots, C_Q[h_k(a)]\}$. Then, it has been shown in [3] that if $f_a \leq \min_a$, then

$$E_{sbf} \simeq (1 - e^{-ks/\omega})^k. \tag{4.1}$$

For $k = \frac{\omega}{s} \ln 2$, the formula (4.1) is minimized as

$$E_{sbf_{\min}} = 2^{-k} \simeq 0.6185^{\frac{\omega}{s}}.$$

For example, if we take $\omega = 8s$, the probability that SBF replies a correct answer is 98%.

4.2 FILM tree

Definition 4.5 (FILM tree). Let $\mathbb{T}$ be a multi-track of length $|\mathbb{T}|_{len} = n$, and m be a positive integer. Let $H = \{h_1, h_2, \cdots, h_k\}$ be a set of k hash functions $h_i : U \to \{0, \ldots, u-1\}$, where the domain U is either Σ^m or $\mathcal{R}^m$. A *FILM tree* $FILM_m(\mathbb{T})$ *of size* m *for* $\mathbb{T}$ is a complete binary tree containing $2^{\lceil \log_2 n \rceil}$ leaves, where each node v represents an SBF of size ω defined as follows. If v is an i-th leaf node with $1 \leq i \leq n - m + 1$, then v represents $SBF_H(\mathbb{T}[i : i + m - 1])$. If $i > n - m + 1$, it represents an empty SBF, that is $(0, 0, \cdots, 0)$. If v is an inner node, v represents the SBF $s_L \oplus s_R$, where s_L (reps. s_R) is the SBF represented by the left (resp. right) child of v. In the sequel, we identify a node v with the SBF that v represents.

Figure 3 shows a simple example of a FILM tree. The number of nodes of a $FILM_m(\mathbb{T})$ is $2^{\lceil \log_2 n \rceil + 1} - 1 = O(n)$ because the number of leaves is $2^{\lceil \log_2 n \rceil}$ and each inner node has exactly two children. Each node represents an SBF so that it requires $O(\omega)$ space. Thus, $FILM_m(\mathbb{T})$ needs $O(n\omega)$ space. Algorithm 2 shows a construction algorithm for the FILM tree, in which the tree structure is implemented as an array A in a standard way; the left (resp. right) child of a node $A[i]$ is stored in $A[2i]$ (resp. $A[2i + 1]$).

By using $FILM_m(\mathbb{T})$, we can efficiently solve both the full- and sub-permuted pattern matching problems for a multi-track pattern $\mathbb{P}$ of length $|\mathbb{P}|_{len} = m$. Algorithm 3 shows the matching algorithm. At line 2, it computes a *query filter* $QF = SBF_H(\mathbb{P})$ for $\mathbb{P}$ using the same set H of hash functions. All positions i satisfying $\mathbb{P} \overset{\bowtie}{\sqsubseteq} \mathbb{T}[i : i + m - 1]$ can be found by a depth first search of the FILM tree, defined as the recursive function *DFS* in line 4. When the algorithm visits c-th node v in the search, if $\min(FILM_m(\mathbb{T}[c]) \ominus QF) \geq 0$ and v is a leaf (line 6 and 7), then the algorithm outputs $c - \mathtt{leafNum} + 1$ as a candidate of permuted-matching position.

The output of Algorithm 3 contains at least all positions i satisfying $\mathbb{P} \overset{\bowtie}{\sqsubseteq} \mathbb{T}[i : i + m - 1]$ for $1 \leq i \leq n - m + 1$. The correctness is shown as follows. Let $c_l = SBF_H(\mathbb{C}_l)$ and $c_r = SBF_H(\mathbb{C}_r)$ be sibling leaves of the FILM tree computed from substrings $\mathbb{C}_l$ and $\mathbb{C}_r$ of multi-track $\mathbb{T}$. Assume that $\mathbb{P}$ permuted-matches $\mathbb{C}_l$, that is $\mathbb{P} \subseteq \mathbb{C}_l$. Because

Algorithm 2: FILM tree construction algorithm

```
    Input: multi-track text $\mathbb{T}$ and an integer $m$
    Output: $FILM_m(\mathbb{T})$
 1  $n = |\mathbb{T}|_{len}$; $N = |\mathbb{T}|_{num}$;
 2  height = $\lceil \log_2 n \rceil$;
 3  leafNum = $2^{\text{height}}$;
 4  for $i = 1$ to $n - m + 1$ do
 5      $FILM_m(\mathbb{T})[\text{leafNum} + i - 1] = SBF_H(\mathbb{T}[i : i + m - 1])$
 6  end;
 7  for $j = 1$ to height do
 8      beginNode = $2^{\text{height}-j}$;
 9      endNode = $2 \cdot$beginNode $- 1$;
10      for $i$ = beginNode to endNode do
11          $FILM_m(\mathbb{T})[i] = FILM_m(\mathbb{T})[2i] \oplus FILM_m(\mathbb{T})[2i + 1]$
12      end
13  end;
14  output $FILM_m(\mathbb{T})$ as a FILM tree
```

Algorithm 3: Permuted pattern matching algorithm using $FILM_m(\mathbb{T})$ $(|\mathbb{T}|_{len} = n)$

```
    Input: pattern $\mathbb{P}$ satisfying $|\mathbb{P}|_{len} = m$
    Output: matching positions of $\mathbb{P}$ in $\mathbb{T}$    /* it may contains some false positives */
 1  leafNum = $2^{\lceil \log_2 n \rceil}$;
 2  $QF = SBF_H(\mathbb{P})$;
 3  $DFS(1)$;  /* start depth-first-search from the root node */
 4  Function $DFS(c)$
 5      sub$_{\min}$ = $\min(FILM_m(\mathbb{T})[c] \ominus QF)$;
 6      if sub$_{\min} \geq 0$ then
 7          if $c \geq$ leafNum then    /* $FILM_m(\mathbb{T})[c]$ is a leaf node */
 8              output $c -$ leafNum $+ 1$
 9          else    /* $FILM_m(\mathbb{T})[c]$ is an inner node */
10              $DFS(2c)$;
11              $DFS(2c + 1)$
12          end
13      end
14  end
```

$QF = SBF_H(\mathbb{P})$, we have $\min(c_l \ominus QF) \geq 0$ by Property 4.4 (2). Let v be the parent node of c_l and c_r. Then, $v = SBF_H(\mathbb{C}_l \cup \mathbb{C}_r)$ from Property 4.4 (1) and the definition of the FILM tree. Now, $\mathbb{P} \subseteq (\mathbb{C}_l \cup \mathbb{C}_r)$ holds, thus $\min(v \ominus QF) \geq 0$. Recursively, $\min(u \ominus QF) \geq 0$ holds for all ancestor nodes u of c_l. Therefore, all positions i satisfying $\mathbb{P} \overset{\bowtie}{\sqsubseteq} \mathbb{T}[i : i + m - 1]$ can be found correctly by the depth first search of the FILM tree. Remark that the output of this algorithm may contain some false positives with small probability, as we have already mentioned in Section 4.1. Thus, to obtain the correct matching positions, we have to verify whether $\mathbb{P} \overset{\bowtie}{\sqsubseteq} \mathbb{T}[i : i + m - 1]$ holds or not for each candidate position i, in a naive way.

5. Hash Functions

By selecting suitable hash functions for the SBF, FILM trees can be adopted to solve various permuted pattern matching problems. In this section, we introduce two types of hash functions; one for exact matching of string data, and the other for approximate matching of numerical data.

5.1 Rolling hash for multi-track strings

In permuted pattern matching for multi-track string, we need a hash function for strings in order to consider a FILM tree for strings. We adopt a simple rolling hash found in Karp–Rabin string matching algorithm [6].

Definition 5.1. We define a *rolling hash* $h_{rh}(t)$ for a string t of length l by

$$h_{rh}(t) = t[1]a^{l-1} + t[2]a^{l-2} + \cdots + t[l]a^0 \pmod{q},$$

where a is a prime number and q is a positive integer.

An advantage of the rolling hash is its efficiency. We can calculate the hash value $h_{rh}(t_i[j : j+m-1])$ in $O(1)$ time after calculating the value $h_{rh}(t_i[j-1 : j+m-2])$, because the following equation holds for any position j in a text t_i;

$$h_{rh}(t_i[j : j+m-1]) = a \cdot h_{rh}(t_i[j-1 : j+m-2]) + t_i[j+m-1] \pmod{q}.$$

5.2 Locality sensitive hashing for numerical multi-tracks

In the approximate permuted pattern matching, if each Euclidean distance between tracks of the text and the pattern is small, the multi-track Euclidean distance is small. Based on this idea, we select hash functions for the Euclidean distance. We use the locality sensitive hashing (LSH), that is a hashing algorithm for the nearest neighbor search problem [1, 2, 4, 5]. The hash values of a hash function in a LSH family are in a collision with high probability if input data are similar. Different LSH families can be used for different distance functions. In this paper, we use the following hash function, that is one of the standard LSH families for Euclidean metric space.

Definition 5.2. We define a *locality sensitive hashing function* $h_{lsh} : \mathcal{R}^m \to \mathcal{N}$ by

$$h_{lsh}(\mathbf{v}) = \left\lfloor \frac{\mathbf{a} \cdot \mathbf{v} + b}{r} \right\rfloor,$$

where each entry of $\mathbf{a} \in \mathcal{R}^m$ is selected independently from a stable distribution, r is a positive real number, and b is selected uniformly from the range $(0, r]$.

By using LSH functions h_{lsh}, we can directly construct a FILM tree for numerical data, without converting them into strings. During the construction, we need to prepare some hash functions. The number of hash functions is determined by the expected collision probability. The construction time of *FILM tree* depends on the number of hash functions.

6. Experiments

We performed three sets of experiments. In the first two of them, we assessed the construction time and search time of our algorithm on random data. In the third one, we performed experiments on real-world data. Throughout the experiments, we used a Linux machine with a 2.4GHz Intel© Xeon CPU EE5-2609 and 256GB RAM, running Debian 7.0. In the experiments on random data, we used the following basic parameter values: the length of a text $|\mathbb{T}|_{len} = 100000$, the number of tracks of a text $|\mathbb{T}|_{num} = 1000$, the length of a pattern $|\mathbb{P}|_{len} = 300$, the alphabet size of a multi-track string text $|\Sigma| = 26$, the number of hash functions $k = 1$ and the size of a SBF used in a FILM tree $\omega = 10000$.

In the experiments, we compared our algorithms using the FILM tree with a multi-track suffix tree (MTST). However, MTST can be applied only to multi-track strings. Thus, we prepared numerical data for the FILM tree and string data for MTST converted from the numerical data in the following manner; the value range of numerical data is divided into equal $|\Sigma|$ parts, and each divided part is assigned to a distinct character. Note that the construction time of MTST excludes the time of this conversion.

A naive implementation of SBF in Algorithm 2 and Algorithm 3 would be an integer array. Instead of it, we used an *associative array* in the experiments, because SBFs are very sparse, that means most elements are 0's, especially if they are near to the leaves. By storing only non-zero elements in the associative array, we can greatly reduce the memory requirement of SBFs. Moreover, the operations $u \oplus v$ and $u \ominus v$ can be computed efficiently, that depends only on the number of non-zero elements, but not the size ω.

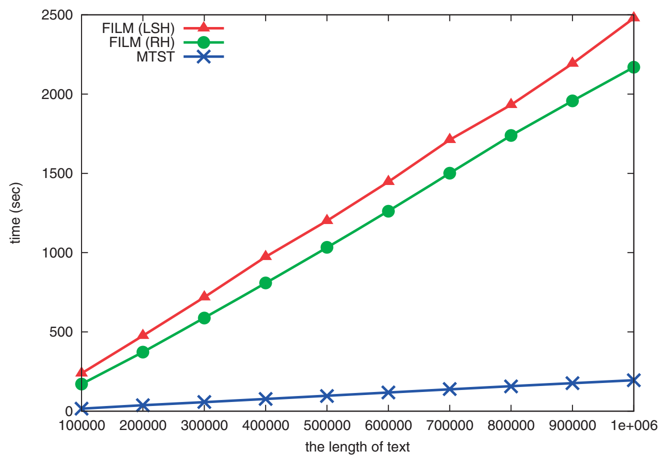
6.1 Construction time on random data

The first set of our experiments assesses the construction time of FILM trees on random data. We varied the values of $|\mathbb{T}|_{len}$, $|\mathbb{T}|_{num}$ and ω, and compared the construction time of FILM trees using rolling hash (RH) and LSH with that of multi-track suffix trees (MTSTs) [7]. In this experiment, we used random numerical data for FILM_{LSH} and the string data converted from the numerical data for FILM_{RH} and MTST.

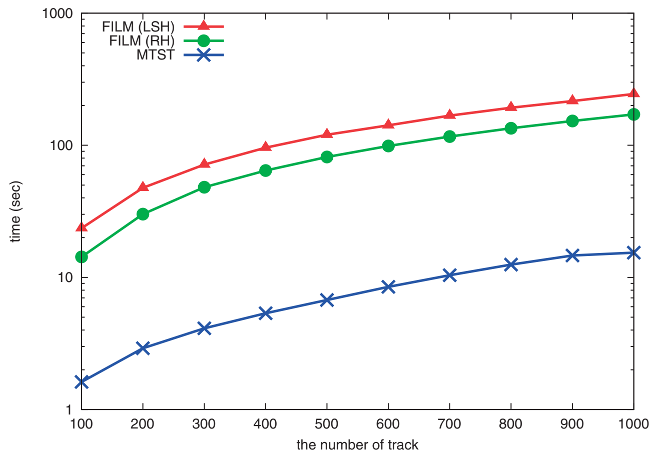
Figure 4 shows the results. In this figure, the y-axes represent the construction times (seconds), and they are on logarithmic scales in Figs. 4(b) and 4(c). The x-axes represent the length $|\mathbb{T}|_{len}$ of the multi-track text, the number $|\mathbb{T}|_{num}$ of tracks, and the size ω of SBF in FILM tree in Figs. 4(a), 4(b), and 4(c), respectively. In all experiments, MTST was the fastest among them, and FILM trees were much slower than MTST. The construction time of FILM tree depended on $|\mathbb{T}|_{len}$ and ω.

6.2 Search time on random data

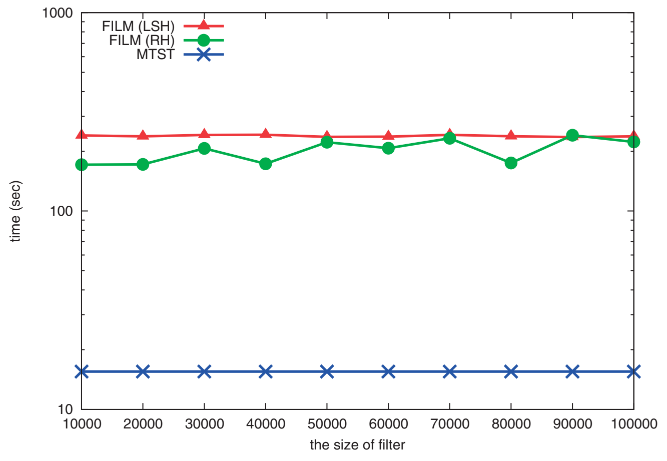
The experiments in the second set concern with the search time on random data. The search using the FILM tree depends on the size ω of SBF, and the height of the FILM tree that reflects the length $|\mathbb{T}|_{len}$ of text. On the other hand, the search using MTST depends on the length $|\mathbb{P}|_{len}$ of pattern. Thus, we compared the search time of FILM_{LSH}, FILM_{RH} and MTST for various values of $|\mathbb{T}|_{len}$, $|\mathbb{P}|_{len}$ and ω.



(a) The length $|\mathbb{T}|_{len}$ of text varied from 100000 to 1000000, while the other parameters were fixed as $|\mathbb{T}|_{num}$ = 1000, $m = |\mathbb{P}|_{len}$ = 300, and ω = 10000.



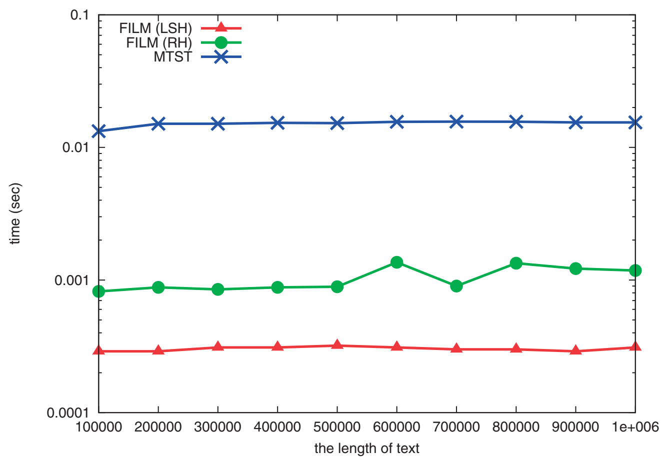
(b) The number $|\mathbb{T}|_{num}$ of tracks varied from 100 to 1000, while the other parameters were fixed as $|\mathbb{T}|_{len}$ = 100000, $m = |\mathbb{P}|_{len}$ = 300, and ω = 10000.



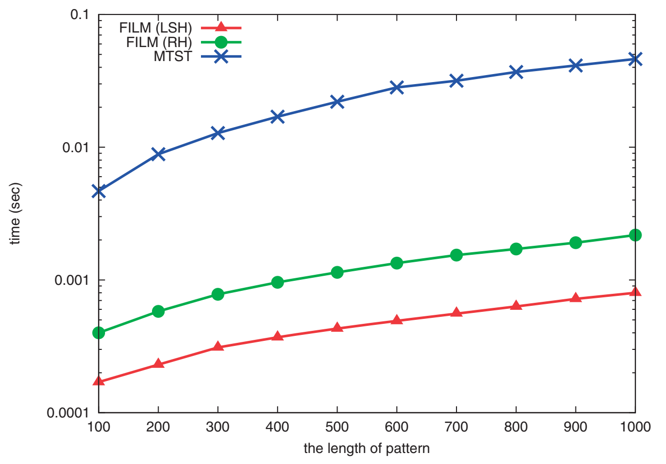
(c) The size ω of SBF varied from 10000 to 100000, while the other parameters were fixed as $|\mathbb{T}|_{len}$ = 100000, $|\mathbb{T}|_{num}$ = 1000 and $m = |\mathbb{P}|_{len}$ = 300. Because MTST does not use SBF, the construction time of it is constant.

Fig. 4. Comparison of the construction time on random data.

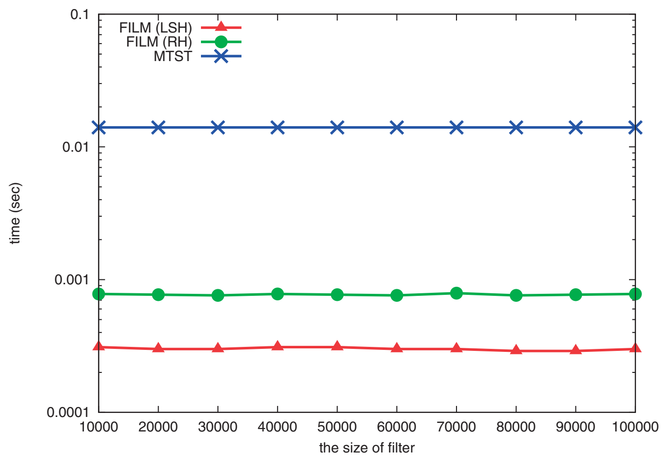
Figure 5 shows the results. The y-axes represent the search times (seconds) on logarithmic scales. The x-axis in Fig. 5(a) represents the length $|\mathbb{T}|_{len}$ of the multi-track text. The matching algorithm using the FILM tree needs to search from the root node to leaf nodes in order to identify the matching positions. The search time depends on the height of the FILM tree, which is $O(\log_2 n)$ with respect to the text length $n = |\mathbb{T}|_{len}$. The result shows that we can ignore this influence from a practical viewpoint. The x-axis in Fig. 5(b) represents the length $|\mathbb{P}|_{len}$ of the pattern.



(a) The length $|\mathbb{T}|_{len}$ of the text varied from 100000 to 1000000, while the other parameters were fixed as $|\mathbb{T}|_{num} = |\mathbb{P}|_{num} = 1000$, $|\mathbb{P}|_{len} = 300$ and $\omega = 10000$.



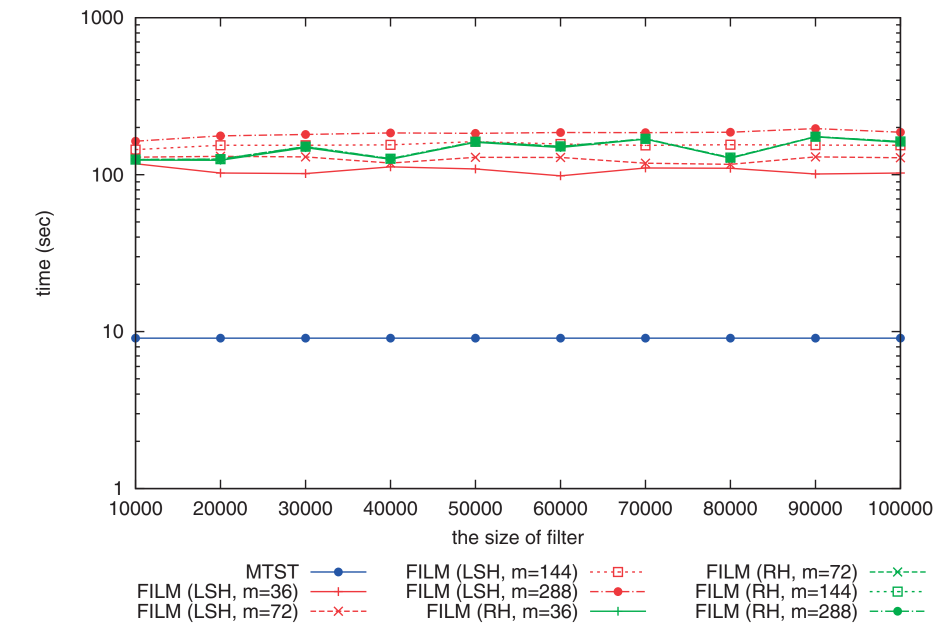
(b) The number $|\mathbb{T}|_{num}$ of tracks varied from 100 to 1000, while the other parameters were fixed as $|\mathbb{T}|_{len} = 100000$, $|\mathbb{T}|_{num} = |\mathbb{P}|_{num} = 1000$ and $\omega = 10000$.



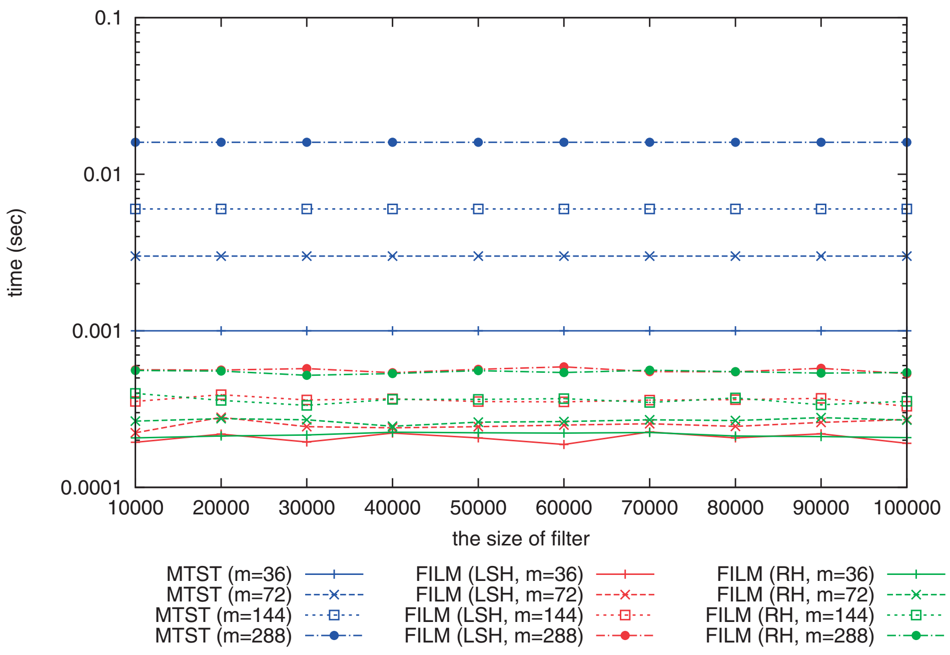
(c) The size ω of SBF varied from 10000 to 100000, while the other parameters were fixed as $|\mathbb{T}|_{len} = 100000$, $|\mathbb{T}|_{num} = |\mathbb{P}|_{num} = 1000$ and $|\mathbb{P}|_{len} = 300$. Because MTST does not use SBF, the search time of it is constant.

Fig. 5. Comparison of the search time on random data.

Concerning with MTST, as we expected, the search time increases as the pattern $\mathbb{P}$ becomes longer, because we have to traverse a path of length $|\mathbb{P}|_{len}$ in MTST. If we implement SBF in the FILM tree with a normal array, the search time does not depend on the length of the pattern in principle. However, we had confirmed in a preliminary experiment that



(a) Construction time.



(b) Search time.

Fig. 6. Comparisons of construction time and search time on the real traffic data. The size ω of SBF varied from 10000 to 100000, and the pattern length $m = |\mathbb{P}|_{len}$ was changed to 36, 72, 144, and 288.

the search time of this implementation was very slow and required large memory, compared to MTST. Thus, we decided to use associative arrays for SBFs. In consequence, the search time of FILM tree is faster than that of MTST, although it mildly depends on the pattern length $|\mathbb{P}|_{len}$. The x-axis in Fig. 5(c) represents the size ω of SBF. If we use normal arrays for SBFs, the search using FILM tree would slow down as ω increases. However, thanks to the associative array implementation, we obtained much faster search, that is practically independent of ω.

All results in these experiments show that our proposed method using FILM tree outperforms MTST on search time in any cases.

6.3 Construction time and search time on traffic data

The third set is the experiments for real-world data. We used some traffic data comprised of car speed measurements at 702 monitoring points on Tokyo Metropolitan Expressway in Japan. We regarded a time series of them as a multi-track numerical text $\mathbb{T}$ with $|\mathbb{T}|_{num} = 702$. At each monitoring point, the average speed of the cars were recorded at every 5 minutes for one year, so that the length $|\mathbb{T}|_{len}$ was 105120.

As typical applications on the traffic data, we are interested in various subjects; for instance, detecting traffic jams, finding some common patterns in them, and extracting some relations among them, with respect to the time, week, and month, and so on. In the most of all these processing, pattern matching is indispensable as a fundamental operation. Therefore, we evaluated the real performance of our proposed method on these data for pattern matching. We examined four lengths of patterns, $|\mathbb{P}|_{len} = m = 288$ (one day record), 144 (12 hours), 72 (6 hours), and 36 (3 hours), and each pattern was randomly cut out from the text $|\mathbb{T}|_{len}$. As is the previous subsection, we fixed $|\Sigma| = 26$ and $k = 1$, and varied the size ω of SBF.

Figure 6(a) shows the construction time, and Fig. 6(b) shows the search time, both in logarithmic scales. We observe that the tendency of the performance is similar to the one for the random data; although construction of FILM tree is slower than that of MTST, searching using FILM tree is much faster in most situations, and the choice of the size ω does not affect the running time very much. We conclude that our proposed method provides an efficient way to support pattern matching on multi-tracks of this amount of numerical data.

7. Conclusion

In this study, we proposed a new data structure FILM tree to solve the permuted pattern matching problem for multi-tracks. FILM tree can be applied to the various types of permuted pattern matching problems, depending on the hash functions employed. We considered some examples to demonstrate the effectiveness of this approach, such as full/sub-permuted pattern matching problems on string multi-tracks and full/sub-permuted approximate pattern matching problems on numerical multi-tracks, as well as providing their algorithms. FILM tree requires $O(n\omega)$ space, where n and ω are the lengths of the multi-track text and the size of SBF, respectively. We performed a comparison with MTST for the full-permuted pattern matching problem and demonstrated that FILM tree can search patterns faster than MTST.

FILM tree is a simple and powerful data structure for permuted pattern matching problems, but it is only suitable for fixed length patterns. Thus, we need to consider the development of a version for variable length patterns in our future research.

REFERENCES

[1] Broder, A. Z., Charikar, M., Frieze, A. M., and Mitzenmacher, M., "Min-wise independent permutations," In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 327–336 (1998).

[2] Charikar, M., "Similarity estimation techniques from rounding algorithms," In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 380–388 (2002).

[3] Cohen, S., and Matias, Y., "Spectral Bloom filters," In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 241–252 (2003).

[4] Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S., "Locality-sensitive hashing scheme based on p-stable distributions," In *Proceedings of 20th Annual Symposium on Computational Geometry*, pages 253–262 (2004).

[5] Indyk, P., and Motwani, R., "Approximate nearest neighbors: Towards removing the curse of dimensionality," In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 604–613 (1998).

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

[7] Katsura, T., Narisawa, K., Shinohara, A., Bannai, H., and Inenaga, S., "Permuted pattern matching on multi-track strings," In *Proceedings of SOFSEM 2013: Theory and Practice of Computer Science*, pages 280–291 (2013).

[8] Sakoe, H., and Chiba, S., "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech and Signal Processing*, **26**: 43–49 (1978).

[9] Busacker, R. G., and Gowen, P. J., "A procedure for determining a family of minimum cost flow networks," *Operations Research Office Technical Report*, vol. 15 (1961).

[10] Dijikstra, E. W., "A note on two problems in connexion with graphs," *Numerische Mathematik*, **1**(1): 269–271 (1959).