Theoretical Computer Science 638 (2016) 159–170

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

FM-index of alignment: A compressed index for similar strings

Joong Chae Na[a], Hyunjoon Kim[b], Heejin Park[c], Thierry Lecroq[d,e], Martine Léonard[d], Laurent Mouchard[d,e], Kunsoo Park[b,*]

[a] Department of Computer Science and Engineering, Sejong University, Seoul 143-747, South Korea
[b] School of Computer Science and Engineering, Seoul National University, Seoul 151-742, South Korea
[c] Department of Computer Science and Engineering, Hanyang University, Seoul 133-791, South Korea
[d] Department of Computer Science, University of Rouen, 76821 Mont-Saint-Aignan, France
[e] Centre for Combinatorics on Words & Applications, School of Engineering & Information Technology, Murdoch University, Murdoch, WA 6150, Australia

ARTICLE INFO

Article history:
Received 15 April 2015
Received in revised form 25 July 2015
Accepted 9 August 2015
Available online 13 August 2015

Keywords:
Indexes for similar strings
FM-indexes
Suffix arrays
Alignments
Backward search

ABSTRACT

In this paper we propose the *FM-index of alignment*, a compressed index for similar strings with the functionalities of pattern search and random access. For this, we first design a new and improved version of the *suffix array of alignment*. The FM-index of alignment is an FM-index of this suffix array of alignment. The FM-index of alignment supports the LF-mapping and backward search, the key functionalities of the FM-index, but the LF-mapping and backward search of our index is significantly more involved than the original FM-index. We implemented the FM-index of alignment and did experiments on 100 genome sequences from the 1000 Genomes Project. The index size of the FM-index of alignment is about a half of that of RLCSA due to Mäkinen et al.

1. Introduction

The 1000 Genomes Project is aiming at building a database of a thousand individual human genome sequences using a cheap and fast sequencing, called Next Generation Sequencing (NGS), and the sequencing of 1092 genomes was announced in 2012 [5]. To sequence an individual genome using the NGS, the individual genome is divided into short segments (called reads) and they are aligned to the Human Reference Genome. This is possible because an individual genome is more than 99% identical to the reference genome. This similarity also enables us to store individual genome sequences efficiently. Instead of storing a thousand genome sequences, only 1% different regions of each individual sequence can be stored [4,22].

Not only efficient storing techniques but also efficient indexing techniques for similar strings have been developed. These indexing techniques provide efficient pattern search as well as random access to the strings. The first such index was proposed by Mäkinen et al. [16,17,23]. Their index uses run-length encoding, a suffix array, and BWT [3]. Huang et al. [13] indexed similar strings by building separate data structures for common regions and non-common regions. In addition, indexes based on Lempel-Ziv compression schemes [15,24] have been developed [6,7,14] and also compressed suffix trees for similar strings have been proposed [1,21]. Some of these indexes are surveyed in [20]. The space reductions of these indexes are achieved mostly by using classical compression schemes, which exploit the similarity of the given strings.

* Corresponding author.
E-mail addresses: jcna@sejong.ac.kr (J.C. Na), hjkim@theory.snu.ac.kr (H. Kim), hjpark@hanyang.ac.kr (H. Park), Thierry.Lecroq@univ-rouen.fr (T. Lecroq), Martine.Leonard@univ-rouen.fr (M. Léonard), Laurent.Mouchard@univ-rouen.fr (L. Mouchard), kpark@theory.snu.ac.kr (K. Park).

http://dx.doi.org/10.1016/j.tcs.2015.08.008
0304-3975/© 2015 Elsevier B.V. All rights reserved.

Na et al. [18,19] took a different approach, which starts with an alignment of similar strings, and they proposed the *suffix tree of alignment* [18] and the *suffix array of alignment* [19]. Finding an alignment of multiple strings is in general an NP-hard problem. In the Next-Generation Sequencing, however, an individual sequence is obtained by aligning its reads against the reference sequence, and thus an alignment between the individual sequence and the reference sequence is readily available. Hence, a multiple alignment of 100 genome sequences, for instance, based on the reference sequence can be easily created, which we did in our experiments of this paper. An advantage of this approach is that we can exploit the similarities (i.e., common regions and non-common regions) of the given strings more precisely because an alignment of the strings is given.

In this paper we propose the *FM-index of alignment*, a compressed index for similar strings with the functionalities of pattern search and random access. For this, we first design a new and improved version of the suffix array of alignment [19], which combines many suffixes of the given strings into one suffix (called an *alignment-suffix*). The FM-index of alignment is an FM-index of this suffix array of alignment. The FM-index of alignment supports the LF-mapping and backward search, the key functionalities of the FM-index [8–10], but the LF-mapping and backward search of our index is significantly more involved than the original FM-index because one alignment-suffix represents many suffixes of the given strings.

We implemented the FM-index of alignment and did experiments on 100 genome sequences from the 1000 Genomes Project. We compared the FM-index of alignment with the original FM-index [10] and RLCSA due to Mäkinen et al. [17]. The index size of the FM-index of alignment is about a half of that of RLCSA, and both of them take much smaller space than the FM-index (which is obvious because the FM-index does not take advantage of the similarities of the strings). The FM-index of alignment is the fastest in pattern search for 100 genome sequences with sparse samplings, and RLCSA is the fastest in random access.

This paper is organized as follows. We first introduce preliminaries in Section 2. Then, we describe our FM-index of alignment including the LF-mapping in Section 3 and we present our backward search algorithm in Section 4. In Section 5, the experimental results are given. Finally, we conclude in Section 6.

2. Preliminaries

Let S be a string over an alphabet Σ whose last character is a special symbol $\# \in \Sigma$ occurring nowhere else in S. We denote the length of S by $|S|$. The ith character of S is denoted by $S[i]$ $(1 \le i \le |S|)$, and a substring $S[i]S[i+1]\dots S[j]$ by $S[i..j]$. The suffix array [12] is a lexicographically sorted list of all the suffixes of a string and the *generalized suffix array (GSA)* is a suffix array for a set of strings. Fig. 1 shows the GSA for the following four strings $S^1 = \mathtt{at\underline{cca}actc\underline{c}c\#}$, $S^2 = \mathtt{at\underline{tca}actc\underline{a}c\#}$, $S^3 = \mathtt{at\underline{ctt}actc\underline{a}c\#}$, and $S^4 = \mathtt{at\underline{aca}actc\underline{c}c\#}$. Note that cyclic shifts rather than suffixes are presented in Fig. 1. The exact suffixes are obtained by deleting characters following # from the cyclic shifts. Moreover, the special character # in a string S^i is considered to be lexicographically smaller than # in S^j for $i < j$. We denote the kth entry of the GSA by $GSA[k]$ for $k \ge 1$. We denote the suffix of S^j starting at position q by suffix (j, q), e.g., $GSA[5] = (2, 5)$ in Fig. 1. Since the lengths of the four strings are all 12, the GSA has 48 entries.

The *FM-index* [10] is a compressed form of the suffix array based on the Burrows–Wheeler Transform (BWT) [3]. Consider suffixes in the suffix array as cyclic shifts. Let F and L be the arrays of the first characters and the last characters of the cyclic shifts, respectively (See Fig. 1). The key notion in the FM-index is the *LF-mapping* defined as follows: Let $L[i] = S^j[q]$. Then, the LF-mapping $LF(i)$ maps the ith entry in L to the kth entry in F such that $F[k] = S^j[q]$. For example, $LF(12) = 39$ in Fig. 1 since $L[12]$ and $F[39]$ store the same character $S^3[5]$ $(= \mathtt{t})$. The LF-mapping can be computed using an array C and a function *occ* defined as follows.

- Let C be the array of size $|\Sigma| + 1$ such that for $\sigma \in \Sigma$, $C[\sigma]$ is the total number of characters in the given strings which are alphabetically smaller than σ. $C[|\Sigma| + 1]$ is the total number of characters in the given strings. In Fig. 1, $C[\mathtt{a}] = 4$ and $C[\mathtt{t}] = 37$.
- Let $occ(\sigma, i)$ be the number of occurrences of character σ in $L[1..i]$. See Fig. 1.

Then, $LF(i) = C[\sigma] + occ(\sigma, i)$, where $\sigma = L[i]$. In Fig. 1, $LF(12) = C[\mathtt{t}] + occ(\mathtt{t}, 12) = 37 + 2 = 39$.

Pattern search is to find all occurrences of a given pattern $P[1..p]$ in the given strings $S^1, \dots, S^m$. Pattern search in the FM-index proceeds backward using the LF-mapping with C and *occ*. Algorithm 1 shows the pseudo-code of the search algorithm in the FM-index [10]. It consists of at most p steps from Step p to Step 1. In Step $\ell = p, \dots, 1$, the algorithm finds the range $(\mathsf{First}_\ell, \mathsf{Last}_\ell)$ in the GSA defined as follows:

i) First_p (resp. Last_p) is the smallest (resp. largest) index i such that $F[i] = P[p]$.
ii) For $\ell = p - 1, \dots, 1$, $\mathsf{First}_\ell = LF(i_s)$ and $\mathsf{Last}_\ell = LF(i_l)$ where i_s (resp. i_l) is the smallest (resp. largest) index in $(\mathsf{First}_{\ell+1}, \mathsf{Last}_{\ell+1})$ such that $L[i] = P[\ell]$. If there exists no such index i, then we set $\mathsf{First}_\ell = \mathsf{Last}_\ell + 1$.

Then, all the suffixes in $GSA[\mathsf{First}_\ell..\mathsf{Last}_\ell]$ are prefixed by $P[\ell..p]$ and no other suffix is prefixed by $P[\ell..p]$. The size of the range decreases monotonically when ℓ decreases. For $\ell = p - 1, \dots, 1$, the range $(\mathsf{First}_\ell, \mathsf{Last}_\ell)$ is computed as follows (line 4): $\mathsf{First}_\ell = C[\sigma] + occ(\sigma, \mathsf{First}_{\ell+1} - 1) + 1$ and $\mathsf{Last}_\ell = C[\sigma] + occ(\sigma, \mathsf{Last}_{\ell+1})$, where $\sigma = P[\ell]$. For example, supposing $P = \mathtt{atc}$ where the range $(\mathsf{First}_3, \mathsf{Last}_3) = (19, 37)$ for $\mathtt{c}$. Then, the range $(\mathsf{First}_2, \mathsf{Last}_2) = (40, 46)$ for $\mathtt{tc}$ is computed as $\mathsf{First}_2 = C[\mathtt{t}] + occ(\mathtt{t}, 19 - 1) + 1 = 37 + 2 + 1 = 40$ and $\mathsf{Last}_2 = C[\mathtt{t}] + occ(\mathtt{t}, 37) = 37 + 9 = 46$. Similarly, the range