Journal of Discrete Algorithms 34 (2015) 23–36

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Alphabet-independent algorithms for finding context-sensitive repeats in linear time

Enno Ohlebusch [*], Timo Beller

Institute of Theoretical Computer Science, University of Ulm, 89069 Ulm, Germany

A R T I C L E   I N F O

Article history:
Available online 27 May 2015

Keywords:
Maximal repeats
Supermaximal repeats
Near-supermaximal repeats
Context-diverse repeats
Enhanced suffix array

A B S T R A C T

The identification of repetitive sequences (repeats) is an essential component of genome sequence analysis, and there are dozens of algorithms that search for exact or approximate repeats. The notions of maximal and supermaximal (exact) repeats have received special attention, and it is possible to simultaneously compute them on index data structures like the suffix tree or the enhanced suffix array. Very recently, this research has been extended in two directions. Gallé and Tealdi [10] devised an alphabet-independent linear-time algorithm that finds all context-diverse repeats (which subsume maximal and supermaximal repeats as special cases), while Taillefer and Miller [31] gave a quadratic-time algorithm that simultaneously computes and classifies maximal, near-supermaximal, and supermaximal repeats. In this paper, we provide new alphabet-independent linear-time algorithms for both tasks.

1. Introduction

In the analysis of a genome, a basic task is to locate and characterize the repetitive sequences (repeats) that exceed a certain minimum length ℓ_{min}. While bacterial genomes usually do not contain large amounts of repetitive sequences, a considerable portion of the genomes of higher organisms is composed of repeats. For example, more than half of the 3 billion basepairs of the haploid human genome consist of repeats; see [13,19] for details.

In order to avoid redundant output, the search for exact repeats is usually restricted to so-called maximal and supermaximal repeats. Let us briefly recall these notions. A substring ω of a (long) string S is an exact repeat if it occurs at least twice in S. A repeat ω of S is a maximal repeat if any extension of ω occurs fewer times in S than ω. A supermaximal repeat is a maximal repeat that is not a proper substring of another repeat. Section 3 discusses articles that describe how maximal and supermaximal repeats can be computed efficiently.

This paper is inspired by two papers published recently. In the first paper, Gallé and Tealdi [10] introduced context-diverse repeats, which subsume maximal and supermaximal repeats as special cases. They provided three algorithms for finding all context-diverse repeats:

GT1. An $O(n\sigma)$ time algorithm based on the enhanced suffix array of S (using a variant of the bottom-up traversal of the lcp-interval tree [1], a method of simulating a bottom-up traversal of the suffix tree), where σ is the size of the underlying alphabet Σ.

[*] Corresponding author.
E-mail addresses: Enno.Ohlebusch@uni-ulm.de (E. Ohlebusch), Timo.Beller@uni-ulm.de (T. Beller).

http://dx.doi.org/10.1016/j.jda.2015.05.005
1570-8667/© 2015 Elsevier B.V. All rights reserved.

i	SA	LCP	BWT	$S_{\mathsf{SA}[i]}$
1	12	−1	i	$\$$
2	11	0	p	$i\$$
3	8	1	s	$ippi\$$
4	5	1	s	$issippi\$$
5	2	4	m	$ississippi\$$
6	1	0	$\$$	$mississippi\$$
7	10	0	p	$pi\$$
8	9	1	i	$ppi\$$
9	7	0	s	$sippi\$$
10	4	2	s	$sissippi\$$
11	6	1	i	$ssippi\$$
12	3	3	i	$ssissippi\$$

lcp-intervals

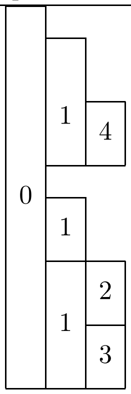

Fig. 1. Suffix array, LCP-array, BWT and lcp-intervals of the string $S = mississippi\$$. The entry $\mathsf{LCP}[13] = -1$ is not shown in the table.

GT2. An $O(n \log n)$ time algorithm based on Fenwick trees (a Fenwick tree permits to calculate a prefix sum of an array of values in $O(\log n)$ time; it is dynamic because the table can be modified in $O(\log n)$ time).
GT3. An $O(n)$ time algorithm that (a) computes right-diverse repeats based on the enhanced suffix array of S, (b) computes left-diverse repeats based on the enhanced suffix array of the reverse string of S, and (c) merges them.

Here, we provide a simpler $O(n)$ time algorithm. In essence, we replace the Fenwick tree in algorithm GT2 with the correction terms devised by Hui [15].

The second paper that inspired our work is [31]. In that paper, Taillefer and Miller study length distributions of repeats in genome sequences. To that end, they used a simple algorithm that simultaneously computes and classifies maximal, near-supermaximal, and supermaximal repeats. The worst-case time complexity of their simple algorithm is $O(n^2)$, but it can be improved to $O(n\sigma)$ if, in a bottom-up traversal of the lcp-interval tree, information at child intervals is propagated to their parent interval. The real challenge is to devise an $O(n)$ time algorithm for this task. Note that the result on context-diverse repeats does not apply because context-diverse repeats do not comprise near-supermaximal repeats. In this paper, we give the first alphabet-independent linear-time algorithm that simultaneously finds and classifies maximal, near-supermaximal, and supermaximal repeats.

Experimental results show that our algorithms are not only of theoretical interest: both have implementations that are faster in practice than their above-mentioned counterparts. A preliminary version of this paper appeared in [23].

2. Preliminaries

Let Σ be an ordered alphabet of size σ whose smallest element is the sentinel character $\$$. In the following, S is a string of length n on Σ having the sentinel character at the end (and nowhere else). For $1 \leq i \leq n$, $S[i]$ denotes the *character at position* i in S. For $i \leq j$, $S[i..j]$ denotes the *substring* of S starting with the character at position i and ending with the character at position j. Furthermore, S_i denotes the i-th suffix $S[i..n]$ of S. The *suffix array* SA of the string S is an array of integers in the range 1 to n specifying the lexicographic ordering of the n suffixes of S, that is, it satisfies $S_{\mathsf{SA}[1]} < S_{\mathsf{SA}[2]} < \cdots < S_{\mathsf{SA}[n]}$; see Fig. 1 for an example. We refer to the overview article [28] for suffix array construction algorithms (some of which have linear runtime).

The Burrows–Wheeler transform [6] converts S into the string $\mathsf{BWT}[1..n]$ defined by $\mathsf{BWT}[i] = S[\mathsf{SA}[i] - 1]$ for all i with $\mathsf{SA}[i] \neq 1$ and $\mathsf{BWT}[i] = \$$ otherwise; see Fig. 1.

The suffix array SA is often enhanced with the so-called LCP-array containing the lengths of longest common prefixes between consecutive suffixes in SA; see Fig. 1. Formally, the LCP-array is an array so that $\mathsf{LCP}[1] = -1 = \mathsf{LCP}[n+1]$ and $\mathsf{LCP}[i] = |\mathsf{lcp}(S_{\mathsf{SA}[i-1]}, S_{\mathsf{SA}[i]})|$ for $2 \leq i \leq n$, where $\mathsf{lcp}(u, v)$ denotes the longest common prefix between two strings u and v. Like the suffix array, the LCP-array can be computed in linear time; see [16,17].

Abouelhoda et al. [1] introduced the concept of lcp-intervals; see Fig. 1. An interval $[i..j]$, where $1 \leq i < j \leq n$, in the LCP-array is called an *lcp-interval of lcp-value* ℓ (denoted by ℓ-$[i..j]$) if

1. $\mathsf{LCP}[i] < \ell$,
2. $\mathsf{LCP}[k] \geq \ell$ for all k with $i + 1 \leq k \leq j$,
3. $\mathsf{LCP}[k] = \ell$ for at least one k with $i + 1 \leq k \leq j$,
4. $\mathsf{LCP}[j + 1] < \ell$.

Every index k, $i + 1 \leq k \leq j$, with $\mathsf{LCP}[k] = \ell$ is called *ℓ-index* or *lcp-index*. Note that each lcp-interval has at least one and at most $\sigma - 1$ many ℓ-indices. Abouelhoda et al. [1] showed that there is a one-to-one correspondence between the set