Journal of Discrete Algorithms 16 (2012) 151–169

Contents lists available at SciVerse ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

p-Suffix sorting as arithmetic coding [☆]

Richard Beal [*], Donald Adjeroh

West Virginia University, Lane Department of Computer Science and Electrical Engineering, Morgantown, WV 26506, United States

ARTICLE INFO

Article history:
Available online 23 May 2012

Keywords:
Parameterized suffix array
Parameterized suffix sorting
Arithmetic coding
Fingerprints
p-string
p-match

ABSTRACT

The challenge of direct parameterized suffix sorting (p-suffix sorting) for a parameterized string (p-string), say T of length-n, is the dynamic nature of the n parameterized suffixes (p-suffixes) of T. In this work, we propose transformative approaches to direct p-suffix sorting by generating and sorting lexicographically numeric fingerprints and arithmetic codes that correspond to individual p-suffixes. Our algorithm to p-suffix sort via fingerprints is the first theoretical linear time algorithm for p-suffix sorting for non-binary parameter alphabets, which assumes that, in practice, all codes are within the range of an integral data type. We eliminate the key problems of fingerprints by introducing an algorithm that exploits the ordering of arithmetic codes to sort p-suffixes in linear time on average. The arithmetic coding approach is further extended to handle p-strings in the worst case. This algorithm is the first direct p-suffix sorting approach in theory to execute in $o(n^2)$ time in the worst case, which improves on the best known theoretical result on this problem that sorts p-suffixes based on p-suffix classifications in $O(n^2)$ time. We show that, based on the algorithmic parameters and the input data, our algorithm does indeed execute in linear time in various cases, which is confirmed with experimental results.

1. Introduction

Conventional pattern matching involves the matching of standard strings over an alphabet Σ. Parameterized pattern matching using parameterized strings, introduced by Baker [4], attempts to answer pattern matching questions beyond its classical counterpart. A parameterized string (p-string) is a production of symbols from the alphabets Σ and Π, which represent the constant symbols and parameter symbols respectively. Given a pair of p-strings S and T, the parameterized pattern matching (p-match) problem is to verify whether the individual constant symbols match and whether there exists a bijection between the parameter symbols of S and T. If the two conditions are met, S is said to be a p-match of T. For example, there exists a p-match between the p-strings $z = y * f / + + y$; and $a = b * f / + + b$; that represent program statements over the alphabets $\Sigma = \{*, /, +, =, ;\}$ and $\Pi = \{a, b, f, y, z\}$. Applications inherent to the p-matching problem include detecting plagiarism in academia and industry, reporting similarities in biological sequences [27], discovering cloned code segments in a program to assist with software maintenance [4], and answering critical legal questions regarding the unauthorized use of intellectual property [29].

Initial solutions to the p-match problem were based on the parameterized suffix tree (p-suffix tree) [4]. Idury et al. [17] studied the multiple p-match problem using automata. The physical space requirements of the p-suffix tree led to algorithms such as parameterized-KMP [3], parameterized-BM [6], and the parameterized suffix array (p-suffix array) [16,12]. Analogous

[☆] A shorter version of this paper was presented at the International Workshop on Combinatorial Algorithms, Victoria, BC, 2011 (Beal and Adjeroh, 2011 [9]). This work was partly supported by grants from the National Historical Publications & Records Commission, and from WV-EPSCoR RCG.
[*] Corresponding author.
E-mail addresses: r.beal@computer.org (R. Beal), don@csee.wvu.edu (D. Adjeroh).

1570-8667/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.jda.2012.05.001

to standard suffix sorting, the problem of parameterized suffix sorting (p-suffix sorting) is to sort all the n parameterized suffixes (p-suffixes) of an n-length p-string into a lexicographic order. The major difficulty is that unlike traditional suffixes of a string, the p-suffixes are dynamic, varying with the starting position of the p-suffix. Thus, standard suffix sorting approaches cannot be directly applied to the p-suffix sorting problem. Current approaches to directly construct the p-suffix array *without* a p-suffix tree for an n-length p-string from an arbitrary alphabet require $O(n^2)$ and $O(n^3)$ time in the worst case [16]. In [16], a standard quicksort was used to sort p-suffixes naïvely in $O(n^3)$ time. An improved algorithm was given in [16] to sort p-suffixes via classification and bucket or radix sorting in $O(n^2)$ time. Prior to our work, the group of I, Deguchi et al. [16,12] was the only known group with results on the direct p-suffix sorting problem. The existence of a better theoretical p-suffix sorting algorithm is proposed as an open problem in [16].

The *direct* p-suffix sorting problem is significant because of two major reasons. First, the p-suffix array, like the traditional suffix array, is a lightweight data structure in terms of space. If we traverse the p-suffix tree to *indirectly* obtain the p-suffix array, we must allocate the heavyweight memory footprint required for the tree, voiding any space advantage of the array representation. In a case where the tree is readily accessible, it is possible to obtain the n indices at the cost of a factor of the alphabet in either the time or space depending on the representation of outgoing edges, i.e. vectors, linked lists, or balanced binary trees. Second, traditional suffix sorting approaches benefit from successive doubling used in [24] or induction [1,2,18,25]. These techniques work with traditional strings because a suffix of a regular string shares information common to other suffixes in the string. This is not necessarily the case with p-suffixes because of their dynamic nature or more formally, the fact that the p-suffix at some position, say k of the p-string T, is not necessarily a suffix or even a substring of the p-suffix at position $(k-1)$, $(k-2)$, or even 1. These obstacles introduce many challenges to the p-suffix sorting problem. In this paper, we improve the running time of p-suffix sorting and address the open problem proposed in [16].

Main Contribution. We construct p-suffix arrays by generating and sorting m-length codes that represent the individual p-suffixes of a p-string. We propose the first theoretical linear time claims to directly p-suffix sort p-strings on average from non-binary parameter alphabets. Further, we propose the first direct p-suffix sorting solution to execute in $o(n^2)$ time in the worst case. We state our main result in the following, where m is the size of the block used in sorting and h is a measure of the number of m-blocks needed to clearly differentiate the p-suffixes:

Theorem 21. *Given a p-string T of length n, p-suffix-sorting of T can be accomplished in $O(n)$ time and $O(n)$ space on average via parameterized arithmetic coding.*

Theorem 26. *Given a p-string T of length n and an integer m with $1 \leqslant m \leqslant n$, the p-suffix-sorting of T is accomplished in $O(n)$ time on average and precisely $O(\sum_{i=1}^{h-1}[(n-i\times m)\log(n-i\times m)]+n\log n)$ time in the worst case via parameterized arithmetic coding. Worst case space is $O(n)$.*

Corollary 27. *Let $\epsilon > 0$ be a very small number $\epsilon = 0.00\ldots1$. For a p-string T of length n and a chosen $m = O(\log^{1+\epsilon} n)$, then in the worst case when $h = O(\frac{n}{m})$ the running time of Algorithm 6 is $o(n^2)$.*

2. Background/related work

Baker [4] defines pattern matching as either: (1) exact matching, (2) parameterized-matching, or (3) matching with modifications. A parameterized match (p-match) is a sophisticated matching scheme based on the composition of a parameterized string (p-string). A p-string is composed of symbols from a constant symbol alphabet Σ and a parameter alphabet Π. A pair of p-strings S and T of length n are said to p-match when the constant symbols $\sigma \in \Sigma$ match and there exists a bijection of parameter symbols $\pi \in \Pi$ between the pair of p-strings. Baker [4] offered the first p-match breakthroughs, namely, the `prev` encoding to detect a p-match and the parameterized suffix tree (p-suffix tree) analogous to the suffix tree for traditional strings [1,15,28]. The p-suffix tree is built on the `prev` encodings of the suffixes of the p-string, demanding $O(n(|\Pi| + \log(|\Pi| + |\Sigma|)))$ construction time in the worst case [5]. Improvements to the p-suffix tree construction were introduced by Kosaraju [21]. Other contributions in the area of parameterized suffix trees include construction via randomized algorithms [10,22,23]. Like the traditional suffix tree [1,15,28], the p-suffix tree [4] implementation suffers from a large memory footprint. Other solutions that address the p-match problem without the space limitations of the p-suffix tree include the parameterized-KMP [3] and parameterized-BM [6], variants of traditional pattern matching approaches. Such p-matching approaches always match a pattern across the text. As a result, these approaches are best suited for infrequent use: to return the location, maximum length, or existence of a p-match. Alternatively, suffix structures (suffix arrays and suffix trees) return a data structure with information about the suffixes of a string to assist in efficient pattern matching and are best suited for applications with a heavy demand on matching.

The native time and space efficiency of the suffix array led to the origination of the parameterized suffix array (p-suffix array). The p-suffix array is analogous to the suffix array for traditional strings introduced in [24]. Manber and Myers [24] show how to combine the suffix array and the *LCP* (longest common prefix) array to competitively search for pattern $P = P[1\ldots m]$ in a text $T = T[1\ldots n]$ in $O(m+\log n)$ time. Direct p-suffix array construction was first introduced by Deguchi