Journal of Discrete Algorithms 16 (2012) 129–150

Contents lists available at SciVerse ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Variations of the parameterized longest previous factor ☆

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 1 June 2012

Keywords:
Parameterized suffix array
Parameterized longest common prefix
p-string
p-match
p-border array
LPF
LCP
Permuted-LCP
Border array

ABSTRACT

The parameterized longest previous factor (*pLPF*) problem as defined for parameterized strings (p-strings) adds a level of parameterization to the longest previous factor (*LPF*) problem originally defined for traditional strings. In this work, we consider the construction of the *pLPF* data structure and identify the strong relationship between the *pLPF* linear time construction and several variations of the problem. Initially, we propose a taxonomy of classes for longest factor problems. Using this taxonomy, we show an interesting connection between the *pLPF* and popular data structures. It is shown that a subset of longest factor problems may be created with the *pLPF* construction. More specifically, the *pLPF* problem is used as a foundation to achieve the linear time construction of popular data structures such as the *LCP*, parameterized-*LCP* (*pLCP*), parameterized-*border* (*p-border*) array, and *border* array. We further generalize the *permuted-LCP* for p-strings and provide a linear time construction. A number of new variations of the *pLPF* problem are proposed and addressed in linear time for both p-strings and traditional strings, including the longest not-equal factor (*LneF*), longest reverse factor (*LrF*), and longest factor (*LF*). The framework of the *pLPF* construction is exploited to efficiently address a multitude of data structures with prospects in various applications. Finally, we implement our algorithms and perform various experiments to confirm theoretical results.

1. Introduction

The longest previous factor (*LPF*) problem finds for each suffix at i in an n-length traditional string $W = W[1]W[2]\dots W[n]$ a longest factor $W[h\dots n]$ with $1 \leqslant h < i \leqslant n$ preceding the suffix $W[i\dots n]$ in W. Crochemore and Ilie [15] studied this data structure for traditional strings. In order to construct the *LPF* array, it was shown in [15] that the suffix array *SA* is useful to quickly identify the most lexicographically similar suffixes that are candidate previous factors for the chosen suffix in question. The use of *SA* expedites the work to construct the *LPF* array in linear time. The linear time construction of the *LPF* data structure makes it a justified choice to use in various applications. The *LPF* array is naturally setup for applications in string compression [41] and detecting runs [32] within a string.

In [10], we introduce the parameterized longest previous factor (*pLPF*) to extend the traditional *LPF* problem to parameterized strings (p-strings). A p-string, introduced by Baker [5], is a generalized form of a string produced from the constant alphabet Σ and the parameter alphabet Π. The alphabet to which a symbol belongs determines exactly *how* the symbol is matched. The parameterized pattern matching (p-match) problem is to identify an equivalence between a pair of p-strings S and T when (1) the individual constant symbols match and (2) there exists a bijection between the parameter symbols of S and T. Prominent applications concerned with the p-match problem include detecting plagiarism in academia and industry,

☆ A shorter variation of this paper was presented at the International Workshop on Combinatorial Algorithms, Victoria, BC, 2011 (Beal and Adjeroh, 2011 [10]). 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
http://dx.doi.org/10.1016/j.jda.2012.05.004

reporting similarities in biological sequences [38], discovering cloned code segments in a program [6], and even answering critical legal questions regarding the unauthorized use of intellectual property [40]. Baker [5] identifies that the p-match bijection can be handled by using a previous (`prev`) encoding scheme where a p-match exists between S and T if and only if $\texttt{prev}(S) = \texttt{prev}(T)$. The `prev` encoding codes each symbol s with the same constant s if $s \in \Sigma$. Otherwise when $s \in \Pi$, `prev` encodes s with the integer distance to the previous s in T or 0 if it is the first instance of s in T. For example, the following p-strings that represent program statements $f/a*q-++q$ and $f/b*c-++c$ over the alphabets $\Sigma = \{*, /, -, +\}$ and $\Pi = \{a, b, c, f, q\}$ successfully p-match since $\texttt{prev}(\text{“}f/a*q-++q\text{”}) = \text{“}0/0*0-++4\text{”} = \texttt{prev}(\text{“}f/b*c-++c\text{”})$. By solving a problem with the p-match, we are also solving the same problem with an exact match when $|\Pi| = 0$ and a mapped match (m-match) when $|\Sigma| = 0$ [4]. We show in [10] that the *pLPF* problem is not a straightforward extension of the *LPF* problem because of the added challenges of the p-match and dynamic nature of the parameterized suffixes (p-suffixes) under the `prev` encoding. We provide an algorithm in [10,11] to construct the *pLPF* data structure in linear time. A significant contribution of [10] is identifying the connection between the *pLPF* data structure with other popular data structures such as the *LPF*, longest common prefix (*LCP*), and parameterized longest common prefix (*pLCP*). We are influenced by the variations of the traditional *LPF* problem studied in [18,17] to further define variations for the *pLPF* problem.

Main contributions. In this work, we extend the power of the *pLPF* construction by addressing variations of the data structure with the same algorithm. Initially, we consider the *pLPF* problem and prove its linear time construction. We are the first to introduce a taxonomy for longest factor problems, which identifies that problems satisfying certain properties can be solved with the same *pLPF* algorithm by simply altering the preprocessing and postprocessing. The *pLPF* framework is the basis for the data structures in this paper. First, it is proven that the *pLCP* and the newly introduced *permuted-pLCP* can be constructed with the *pLPF* framework in linear time. Next, we introduce three new variants of the *pLPF* array, namely the parameterized longest not-equal factor (*pLneF*), parameterized longest reverse factor (*pLrF*), and parameterized longest factor (*pLF*), and prove that we can use the *pLPF* framework to construct these variants in linear time. We identify that the *border* array [39], an important data structure in string pattern matching, is also a variant of the *pLPF*. It is then shown how to compute the parameterized-*border* (*p-border*) array in linear time using the *pLPF* framework. For simplicity, throughout this work, our construction algorithms consider the most common case of p-strings – where the ordering of p-suffixes behave like the ordering of regular suffixes. This corresponds to the case of Fig. 1(a) of [11]. A straightforward implementation of the details in [11] allow the algorithms to support all the other cases identified. In terms of traditional data structures such as the longest common prefix (*LCP*) and longest previous factor (*LPF*), we prove that our p-string oriented algorithms can be used to solve these standard problems. Finally, we implement our algorithms considering all of the details in [11] and confirm the linear time nature of the constructions. We also show how our newfound algorithms compare with standard algorithms in terms of the traditional *LCP* and *LPF* constructions. The following theorems formalize our core contributions, where pSA_S denotes the p-suffix array on p-string S.

Theorem 16. *Given an n-length p-string T, $prevT = \texttt{prev}(T)$, the `prev` encoding of T, and pSA_T, the parameterized suffix array for T, the algorithm `compute_pLPF` constructs the pLPF array in $O(n)$ time.*

Theorem 20. *Given an n-length p-string T, $prevT = \texttt{prev}(T)$, the `prev` encoding of T, and pSA_T, the parameterized suffix array for T, the `construct` algorithm can be used to construct the pLCP and permuted-pLCP arrays in $O(n)$ time.*

Theorem 27. *Given an n-length p-string T, $prevT = \texttt{prev}(T)$, pSA_T, $Q_1 = T[1\ldots n-1]\$_1$, $Q_2 = T[1\ldots n-1]^R\$_2$, $Q = Q_1 \circ Q_2$, $prevT_1 = \texttt{prev}(Q_1)$, $prevT_2 = \texttt{prev}(Q_2)$, and pSA_Q, the pLneF, pLrF, and pLF data structures are each constructed in $O(n)$ time.*

Theorem 28. *Given an n-length p-string T, $prevT = \texttt{prev}(T)$, the `prev` encoding of T, and pSA_T, the parameterized suffix array for T, the algorithm `compute_p-border` computes the p-border array in $O(n)$ expected time.*

2. Background/related work

Baker [6] identifies three types of pattern matching: (1) exact matching, (2) parameterized matching (p-match), and (3) matching with modifications. The p-match generalizes exact matching by using a parameterized string (p-string) composed of symbols from a constant symbol alphabet Σ and a parameter alphabet Π. A p-match exists between a pair of p-strings S and T of length n when (1) the constant symbols $\sigma \in \Sigma$ match and (2) there exists a bijection of parameter symbols $\pi \in \Pi$ between the p-strings. The first p-match breakthroughs, namely, the `prev` encoding and the parameterized suffix tree (p-suffix tree) were introduced by Baker [5]. The p-suffix tree construction time was improved by Baker in [7]. Other contributions in the area of parameterized suffix trees include the improved construction in [29] and the randomized algorithms in [14,30,31]. Like the traditional suffix tree [22,39,1], the p-suffix tree [5] 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 [4] and parameterized-BM [8], variants of traditional pattern matching approaches. Idury et al. [26] studied the multiple p-match problem using automata and a structure that is now referred to as the