Journal of Discrete Algorithms 42 (2017) 23–34

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

A prefix array for parameterized strings

Richard Beal [a,*], Donald A. Adjeroh [a], W.F. Smyth [b,c]

[a] Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV, United States
[b] Algorithms Research Group, Department of Computing & Software, McMaster University, Hamilton, ON, Canada
[c] School of Engineering & Information Technology, Murdoch University, Perth, WA, Australia

ARTICLE INFO

Article history:
Available online 21 November 2016

Keywords:
Parameterized prefix array
Compact parameterized prefix array
Parameterized string with holes
Parameterized longest previous factor
p-string
hp-string
p-match
hp-match
pPA
cpPA
chpPA
pLPF

ABSTRACT

A parameterized string (p-string) is a generalization of the traditional string over two alphabets: a constant alphabet and a parameter alphabet. A parameterized match (p-match) exists between two p-strings if the constants match exactly and if there exists a bijection between the parameter symbols. Historically, p-strings have been leveraged for source code cloning, plagiarism detection, and biological sequence structural similarity. In this work, we identify the connection between the p-match and music, one of several applications to motivate our study of holes in p-strings, and prefix array-based data structures for p-strings. First, we introduce the parameterized prefix array (*pPA*) for p-strings and its succinct representation, the compact parameterized prefix array (*cpPA*). We show an interesting construction of the *cpPA* via the parameterized longest previous factor (*pLPF*), a more recently proposed array with connections to various pattern matching data structures and LZ factorization. Next, we introduce the parameterized string with holes (hp-string), needed to address a special form of indeterminate pattern matching with p-strings. Then, we show how to construct the compact prefix array for hp-strings. Finally, we discuss applications for our data structures.

1. Introduction

A parameterized string (p-string) is composed of symbols from a constant alphabet Σ and a parameter alphabet Π. Two p-strings S and T are parameterized matches (p-matches) if (1) all of the constant symbols match exactly, i.e. if $S[i], T[i] \in \Sigma$ then $S[i] = T[i]$, and (2) the parameter symbols form a bijection between S and T, i.e. for $S[i], T[i] \in \Pi$ and for every j where $S[j] \in \Pi$ and $S[j] = S[i]$, we have $T[j] \in \Pi$ and $T[j] = T[i]$. In other words, the p-match is a special form of inexact pattern matching, where the constants match exactly and the parameters form a bijection. While the p-match framework has been taken to *further* inexact forms such as approximate matching [35], function matching [5] (relaxing the parameter bijection), and matching with mismatches [7,6], the research in this area is very limited. Currently, the problem of indeterminate pattern matching on p-strings has not been studied.

We are motivated to study this area because of the generalization of the p-match and the intricate applications that the p-match naturally addresses. By using only the Σ alphabet, any p-match algorithm/result also applies to traditional string theory [46,34,1], the foundation of solutions to various applications. Further, the p-match naturally addresses problems in biology [45], software engineering [10], academia [8], and music. Various music applications have been approached from

* Corresponding author.
E-mail addresses: richard.a.beal@gmail.com (R. Beal), don@csee.wvu.edu (D.A. Adjeroh), smyth@mcmaster.ca (W.F. Smyth).

http://dx.doi.org/10.1016/j.jda.2016.11.002
1570-8667/© 2016 Elsevier B.V. All rights reserved.

the perspective of string theory and pattern matching [23,3]; we will later make the connection between music and the p-match. By allowing another level of inexactness to the p-match, we will add to the capability of the p-match framework and support more sophisticated applications.

Here, we consider the prefix array (*PA*), which was used to find repetitions in [44], used within the Z-algorithm for pattern matching [34], used in algorithms by [26], and formally defined as a data structure in [47,22]. For an n-length T, $PA[i]$ is the length of the longest prefix common between T and $T[i...n]$. The *PA* computation is powerful for traditional pattern matching applications and, unlike the related-*border* array [46], where each $border[i]$ is the length of the longest prefix of $T[1...i]$ that matches a suffix of $T[1...i]$, the *PA* is able to also support indeterminate pattern matching [47,21,24,2]. The *PA* has even been represented more compactly [47]. Currently, there is no *PA* or compact *PA* for p-strings.

In this work, we introduce the parameterized prefix array (*pPA*) and its succinct form, the compact parameterized prefix array (*cpPA*), whose length η is the number of nonzero entries in *pPA*. This extension is challenging because prefixes and suffixes of p-strings do not necessarily behave like traditional strings. Our constructions make use of the longest previous factor (*LPF*) data structure [32], which was primarily used for LZ factorization [28] and has recently motivated much research [27,29,30,12,13,17,18]. In [12], we proposed the parameterized longest previous factor (*pLPF*) to extend the *LPF* to p-strings. We show in [13] the power of the *pLPF*/*LPF* data structures, as they can be used to construct many data structures such as the longest common prefix (*LCP*), parameterized longest common prefix (*pLCP*), *border*, parameterized-*border*, etc.

We introduce the hp-string, a p-string with holes (don't care symbols), which is a generalization of partial words [20]. Then, we define the hp-matching problem, and construct *chpPA*, the compact parameterized prefix array for hp-strings. The (compact) prefix array is a staple for indeterminate pattern matching because the linear space data structure can be used to obtain all borders of each prefix with indeterminate symbols, which is a problem that is solved in quadratic time in the worst-case (for strings with wildcards) [41]. Lastly, we identify applications for our data structures in music and software engineering. Our main theoretical results are formalized below.

Theorem 2. *Given the pLPF and $\mathcal{L}$ on the n-length p-string T, the pPA on T can be constructed in $O(n)$ time and $O(1)$ extra space.*

Theorem 4. *Given the pLPF and $\mathcal{L}$ for the n-length p-string T, and the symbol alphabet encoding $\alpha(T)$, the η-length cpPA on T can be constructed in $O(n \log \eta)$ time and $O(1)$ extra space.*

Theorem 5. *Given the parameter alphabet Π and the symbol alphabet encoding $h\alpha(T)$ for the n-length hp-string T, the η-length chpPA on T can be constructed in $O(n\eta)$ time and $O(|\Pi|)$ extra space.*

2. Background

Baker [10] identifies three types of pattern matching: (1) exact matching, (2) parameterized matching (p-match), and (3) matching with modifications. In this work, we focus on (2) and a new fusion between (2) and (3). The first p-match breakthroughs revolved around suffix structures on the previous (`prev`) encoding. The parameterized suffix tree (p-suffix tree) was proposed and constructed in [8]. Additional improvements to the p-suffix tree are given in [42,25,43]. Like the traditional suffix tree [46,34,1], the p-suffix tree [8] implementation suffers from a large memory footprint in practice. One p-matching solution to address the space problem is the parameterized suffix array (p-suffix array) in [37,31]. The work in [11] gives sub-quadratic and near-linear time worst-case p-suffix array constructions. The structural string (s-string) was introduced by Shibuya [45] as an extension to the p-match that incorporates complementary symbols to support structural matching (s-matching) of RNA secondary structures. Originally, the structural suffix tree was constructed in [45] to perform the s-match. The structural suffix array was proposed in [16] as an alternative suffix structure for the s-match.

Other solutions that address the p-match without the space limitations of the p-suffix tree include the parameterized-KMP [4] and parameterized-BM [9], variants of traditional pattern matching schemes. Further, the p-match problem is addressed via the Shift-OR mechanism in [33]. Idury et al. [40] studied a heuristic known as the `pfail` function to address the multiple p-match problem using the traditional Aho–Corasick automaton. This `pfail` function is viewed as the parameterized border array (*p-border*), analogous to the traditional *border* array [46]. This has been used for p-matching and studied in a variety of combinatorial problems in [38,39]. In [15], we construct the structural border (*s-border*) array, a border array with respect to the s-string, and provide the first s-match algorithm without suffix structures.

The aforementioned p-matching is specifically for uncompressed texts. In [7,6], fully compressed p-matching was proposed on run-length encodings. In [15], we showed how to p-match on run-length encoded strings as an application of the *s-border* array. Compressed p-matching between a compressed text and an uncompressed pattern was addressed in [14]. The problem of p-matching with mismatches was studied in [35,7,6]. In [5], generalized function matching was defined to relax the p-match and address problems where the parameter bijection is too restrictive.

Indeterminate strings allow a position in the string to be any subset of symbols in the alphabet, and a hole (or don't care symbol) can match any symbol in the alphabet. Even though the *border* array supports standard pattern matching [46] and extensions support p-matching/s-matching [15], the *border* definition fails to support indeterminate symbols [47]. However, the prefix array (*PA*) [44,34,26,22], though closely related to the *border* array, does support indeterminate pattern matching [47,21]. Indeterminate pattern matching has also been addressed without the prefix array [34,36]. In this work, we