Information and Computation 232 (2013) 10–18

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Compressed property suffix trees ☆

Wing-Kai Hon [a], Manish Patil [b], Rahul Shah [b,*], Sharma V. Thankachan [b]

[a] Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
[b] Department of Computer Science, Louisiana State University, Baton Rouge, LA, USA

ARTICLE INFO

Article history:
Received 2 December 2011
Available online 18 September 2013

Keywords:
Property matching
Suffix trees
Property suffix trees

ABSTRACT

Property matching is a biologically motivated problem where the task is to find those occurrences of an online pattern P in a string text T (of size n), such that the matched text part satisfies some conceptual property. The property of a string is a set π of (possibly overlapping) intervals $\{(s_1, f_1), (s_2, f_2), \ldots\}$ corresponding to the part of text and an occurrence of a pattern $P = T[i, \ldots, (i + |P| - 1)]$ is a valid output only if $T[i, \ldots, (i + |P| - 1)]$ is completely contained in at least one interval $(s_j, f_j) \in \pi$. The indexing version of this problem was introduced by A. Amir (2008), where the text is preprocessed in $O(n \log \sigma + n \log\log n)$ time and an $O(n \log n)$ bits index, named Property Suffix Tree (PST) is maintained. PST can perform property matching in $O(|P| \log \sigma + occ_\pi)$ time, where occ_π is the number of occurrences of P in T satisfying the property. T. Kopelowitz (2010) considered the dynamic version of this problem where intervals can be added or deleted. However, all these indexes take space linear to the size of text ($O(n \log n)$ bits), which can be much more than the size of the text ($n \log \sigma$ bits). In this paper, we propose the first index for property matching occupying space close to the entropy compressed space requirement of the text. Our compressed index takes $|CSA| + n(2 + \epsilon + o(1))$ bits space and performs query answering in $O(t(|P|) + \frac{1}{\epsilon}(1 + occ_\pi) t_{SA})$ time, where $|CSA|$ is the size of compressed suffix array of T, $t(|P|)$ be the time for searching a pattern of length $|P|$ in CSA, t_{SA} is the time for computing the suffix array value and $\epsilon > 0$ is a constant. We also introduce a dynamic index, which takes $|CSA| + O(n + |\pi| \log n)$ bits space and performs query answering in $O(t(|P|) + (1 + occ_\pi) \log n (t_{SA} + \log n / \log\log n))$ time and can update (insert/delete) an interval (s, f) in $O((f - s)(\log n + t_{SA}))$ time.

1. Introduction

Given a text T of size n over an alphabet set Σ of size σ, the fundamental problem in text indexing is to preprocess this text and maintain an index such that whenever an online pattern P comes as query, all the occurrences of P in T can be reported efficiently. A classic data structure for solving this problem is suffix tree which can perform pattern matching in optimal $O(|P| + occ)$ time, where occ is the number of occurrences of P in T. Another classical data structure is suffix array with a query time of $O(|P| \log n + occ)$ [1]. This can be improved to $O(|P| + \log n + occ)$ using an additional data structure called LCP array [1]. However, these indexes take $O(n \log n)$ bits space, which can be much more than the optimal $n \log \sigma$ bits. For example in genome data ($\Sigma = \{A, G, C, T\}$), $\log \sigma$ is 2, where as $\log n$ is around 30. As the memory of computers is limited, this gives a clear motivation to have compressed data structures for handling large data. This long standing problem

☆ A preliminary version appears in *Proceedings of Data Compression Conference*, 2011. This work is supported in part by Taiwan NSC Grant 99-2221-E-007-123-MY3 (Wing-Kai Hon), and by US NSF Grant CCF-1017623, CCF-1218904 (Rahul Shah).

* Corresponding author at: 3122-A Patrick F. Taylor Hall, LSU School of Electrical Engineering and Computer Science, Baton Rouge, LA 70803, USA. Fax: +1 225 578 1465.

E-mail addresses: wkhon@cs.nthu.edu.tw (W.-K. Hon), mpatil@csc.lsu.edu (M. Patil), rahul@csc.lsu.edu (R. Shah), thanks@csc.lsu.edu (S.V. Thankachan).

0890-5401/$ – see front matter
http://dx.doi.org/10.1016/j.ic.2013.09.001

was positively answered by compressed suffix array proposed by Grossi and Vitter [2] and FM-index proposed by Ferragina and Manzini [3]. Different versions of these indexes are available which achieve different space–time trade-offs (see [4] for an excellent survey). Both these indexes can efficiently handle general pattern matching in compressed space. Focus of this paper is on a special kind of pattern matching called property matching.

The property of a text is a set π of (possibly overlapping) intervals $\{(s_1, f_1), (s_2, f_2), \ldots\}$, such that $T[s_j, \ldots, f_j]$ satisfies some conceptual property. Property matching is a variant of classic string matching with an additional constraint that an occurrence of a pattern $P = T[i, \ldots, (i + |P| - 1)]$ is completely contained in at least one interval $(s_j, f_j) \in \pi$. The main motivation of this problem comes from biological applications. In molecular biology, it has long been a practice to consider special genome areas by their structures [5]. For example, the following problem can be modeled as property matching: find all the occurrences of a give pattern in a genome, provided it appears in a repetitive genomic structure such as tandem repeats, SINEs (short Interspersed Nuclear Sequences), or LINEs (line Interspersed Nuclear Sequences) [6].

A new pattern matching paradigm has recently attracted a lot attention where the given text T is weighted [7–9]. At each position in a weighted text, a set of characters appears instead of a fixed single character occurring in a normal string. Such a set of characters appearing at a particular text position also includes probability of appearance of each of it's member character at that position. Weighted text is common in various applications of computational biology. The problem of "motif discovery" is one among these applications that can be benefited due to improvements in the area of property matching. It is essentially a problem of exact pattern matching on weighted text and is also known as "weighted matching". In this problem, task is to seek whether or not, and where a given motif (pattern) occurs in a weighted text. Amir et al. [7] present a reduction of this problem to property matching and show how off-the-shelf techniques for property matching can be used to solve it. The proposed reduction also yields solutions to other problems common in pattern matching community such as, scaled matching, swapped matching, and parameterized matching in weighted text [7].

It is easy to solve algorithmic version of property matching problem in time linear to the size of text. Any standard string searching algorithm can be used to retrieve all the occurrences and later filter out those occurrences which are within the intervals in π. When it comes to indexing problem, our task is to answer the query in time proportional to the size of pattern (not to the size of text) and the number of outputs. Normal string searching data structures like suffix trees or suffix arrays cannot be directly applied in this case as they report all the occurrences (occ) of P in T. Note that occ can be much more than occ_π, where occ_π is the number of occurrences of P satisfying the text property π. Hence the above strategy is not optimal.

Most of the indexing solutions augment suffix tree/array data structure with extra information, such that a query time proportional to $|P|$ and occ_π can be obtained. However, all the previously known indexes take $O(n \log n)$ bits and are not space efficient. In this paper, our objective is to design compressed space indexes for property matching problem. The suffix tree/array data structure can be replaced by compressed suffix tree/array and can be used as a black box. The challenging part is to come up with an encoding scheme for compressing the augmented information while maintaining efficient query capabilities, and is the key contribution of this paper.

1.1. Previous results

Property matching problem was first studied by Amir et al. [7]. They introduced a new data structure called property suffix tree (PST), which can solve the problem in $O(|P| \log \sigma + occ_\pi)$ time. Their structure is basically a suffix tree augmented with extra information so as to report only occ_π outputs. PST takes $O(n \log n)$ bits space and can be constructed in $O(n \log \sigma + n \log \log n)$ time. Later Iliopoulos and Rahman [10] proposed an alternative index called IDS-PIP, based on Range Maximum Queries (RMQ), which can be constructed in linear time. However their index could not correctly handle the case when the intervals are not disjoint. Juan et al. [11] modified IDS-PIP index for handling the general case, which takes $O(n \log n)$ bits space and can answer the query in optimal $O(|P| + occ_\pi)$ time. Another direction of research in this topic is to consider the dynamic version of the problem, where new intervals can be inserted or existing intervals can be deleted. Kopelowitz [12] proposed an index where insertion or deletion of an interval (s, f) can be performed in $O(f - s + \log \log n)$ time while maintaining linear index space and optimal query time. Though the open problem of designing a compressed index for property matching remains unanswered, Zhao and Lu [13] (parallelly and independently to our work) have proposed a space efficient index for property matching, which is truly succinct only when $\pi = O(n/\log n)$.

1.2. Our results

In this paper, we propose a space efficient index for property matching, namely Compressed Property Suffix Tree (C-PST). Theorem 1 summarizes the result for C-PST, when intervals in the text property π are fixed. Theorem 2 summarizes the result for a dynamic C-PST, where text property π is dynamic i.e. new intervals can be added to or existing intervals can be deleted from π.

Theorem 1. *Given a text T of length n and a property π, a C-PST can be maintained in $|CSA| + n(2 + \epsilon + o(1))$ bits such that property matching queries can be answered in $O(t(|P|) + \frac{1}{\epsilon}(1 + occ_\pi)t_{SA})$ time, where P is the online query pattern, $|CSA|$ is the size of compressed suffix array of T, $t(|P|)$ is the time for searching P in CSA, t_{SA} is the time for computing the suffix array (SA) value and $\epsilon > 0$ is a constant.*