Theoretical Computer Science 638 (2016) 44–51

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

The property suffix tree with dynamic properties ☆

Tsvi Kopelowitz

University of Michigan, Ann Arbor, MI, USA

ARTICLE INFO

Article history:
Received 15 April 2015
Received in revised form 24 February 2016
Accepted 24 February 2016
Available online 21 March 2016

Keywords:
Pattern matching
Property matching
Suffix tree
Dynamic indexing

ABSTRACT

In the Property Indexing Problem the goal is to preprocess a text T of size n over a constant sized alphabet Σ and a set of intervals π over the text positions, such that given a query pattern P of size m we can report all of the occurrences of P in T which are completely contained within some interval from π. This type of matching is extremely helpful for scenarios in molecular biology where it has long been a practice to consider special areas in the genome by their structure. It is also helpful for solving pattern matching problems over weighted sequences.

So far the focus has been on the static version of this problem where the intervals are given a priori and never changed. This paper focuses on several dynamic settings of π including an incremental version where new intervals are inserted into π, a decremental version where intervals are deleted from π, a fully dynamic version where intervals may be inserted into or deleted from π, and a batched insertions version where sets of intervals are inserted into π. In particular, the batched version provides a new (optimal) algorithm for the static case.

1. Introduction

In many pattern matching applications there are some special characteristics or *properties* that can be established for various locations in a given text. In this context, a *property* for a string is the set of intervals corresponding to the parts of the string satisfying the conceptual property or characteristics. In the *Property matching* problem involving a text T over a constant[1] alphabet Σ, a pattern P, and a set of property intervals π, the goal is to output the locations of substrings of T which both match P and are completely contained within some interval in π. Such locations conceptually correspond to appearances of P in T at locations which have the desired property. Some examples come from molecular biology, where it has long been a practice to consider special areas of the genome by their structure. Examples are repetitive genomic structures [1] such as *tandem repeats*, *LINEs* (Long Interspersed Nuclear Sequences) and *SINEs* (Short Interspersed Nuclear Sequences) [2]. Many problems in biology can be expressed as property matching problems, for example, finding all occurrences of a given pattern in a genome (the text), provided it appears in a SINE, or LINE (the property intervals).

Property matching has also proven to be a useful tool for solving pattern matching problems, under various definitions of matchings, on *weighted sequences* where characters in a text of size n are given by independent distributions over the alphabet Σ, and the goal is to find matches that occur with probability larger than some given threshold. Amir et al. [3]

☆ This work is supported in part by NSF grants CCF-1217338, CNS-1318294, and CCF-1514383.
E-mail address: kopelot@gmail.com.

[1] The assumption that the alphabet is constant is only made for simplification of the presentation. If the alphabet is not constant then the time costs of the algorithms remain the same by using dynamic lookup tables.

http://dx.doi.org/10.1016/j.tcs.2016.02.033
0304-3975/

showed that if the probability threshold is ϵ, then the pattern matching problem on weighted sequences can be converted to a property pattern matching problem on a text of size $O(\frac{n}{\epsilon^2}\log\frac{1}{\epsilon})$. This allows one to solve property pattern matching for various definitions of matching (see [3] for examples).

Property indexing Clearly, there is no great challenge in sequential property pattern matching since the intersection of the property intervals and matching locations can be done in linear time. However, the problem becomes more complex when it is required to *index* a text T of size n with property π, so that given a (relatively short) query pattern P of size m one can quickly find all occurrences of P in T that are contained within intervals of π. Typically one is interested in a linear preprocessing time and a query time which depends linearly on m and the size of the query output.

The indexing version of the classic pattern matching problem and its many variants have been central in pattern matching (e.g., [4–17]). However, when one wishes to index a text with properties, a new dilemma is presented. If one were to use the somewhat naive approach by using the conventional indexing techniques to locate the matching locations and then intersect them with the intervals of the property, the query time will depend on the number of matching locations of P in the T, which could be much larger than the size of the output.

Thus, several papers have tackled this problem by building the *Property Suffix Tree* (or PST for short), which is essentially a suffix tree where each suffix is truncated to its smallest prefix which is contained in an interval from π. Amir et al. [3] introduced the PST and showed how one can construct the PST in $O(|\pi| + n\log\log n)$ time where $|\pi|$ is the number of intervals in π. The construction method is based on weighted ancestor queries [18,19]. Iliopoulos and Rahman [20] showed how one can construct the PST in $O(|\pi| + n)$ time using an approach based on range minima queries, which was later clarified by Juan, Liu, and Wang [21]. Moreover, it is possible to combine the ideas in Amir et al. [3] with the recent results of [22] in order to obtain another linear time construction. Finally, Hon et al. [23] considered a compressed construction of the PST.

Dynamic properties The question of interest here is to consider the dynamic version of the problem, where intervals are inserted into π or removed from π. The variants discussed here are *incremental* updates where new intervals are inserted into π, *decremental* updates where existing intervals are deleted from π, *fully dynamic* updates where both insertions and deletions are allowed, and *batched* updates where a set of intervals are inserted together into π.

One approach for solving these dynamic variants is to convert the approaches used for the static case while using dynamic versions of the internal data structures used as black boxes in the static solutions (such as dynamic range minimum queries or dynamic weighted ancestors). However, such approaches suffer from a time cost overhead when considering the dynamic variants of these data structures. Instead, a new approach for constructing the PST is presented here, which, as will be described, turns out to be extremely useful for the dynamic variants under consideration. This new approach makes use of the *suffix links* of the PST which allows one to quickly jump between the parts in the PST that need to be changed due to an update to π. This is described in more detail in Section 3. Our new approach also provides a new static construction of the PST which runs in $O(n + |\pi|)$ time, matching the known optimal upper bounds. This is summarized in Section 6.1.

Summary of results In the rest of this paper we prove the following.

Theorem 1. *(See Proof in Section 3.) There exists a linear sized data structure that maintains the PST on a text T and property π with linear preprocessing time, while supporting insertion operations in $O(f - s + 1)$ time, where $[s, f]$ is the interval being inserted into π.*

Theorem 2. *(See Proof in Section 4.) There exists a linear sized data structure that maintains the PST on a text T and property π with linear preprocessing time, while supporting deletion operations in $O(f - s + 1)$ time, where $[s, f]$ is the interval being deleted from π.*

Theorem 3. *(See Proof in Section 5.) There exists a linear sized data structure that maintains the PST on a text T of size n and property π with linear preprocessing time, while supporting insertion and deletion operations in $O(f - s + \log\log n)$ time, where $[s, f]$ is the interval being inserted into or deleted from π.*

Theorem 4. *(See Proof in Section 6.) There exists a linear sized data structure that maintains the PST on a text T and property π with linear preprocessing time, while supporting batched insertion operations in $O(\text{cover-size}(I) + |I|)$ time, where $I = \{[s_1, f_1], [s_2, f_2], \ldots, [s_\ell, f_\ell]\}$ is the batch of intervals being added to π and $\text{cover-size}(I) = |\{1 \le i \le n : \exists [s, f] \in I \text{ such that } i \in [s, f]\}|$.*

2. Preliminaries and definitions

For a string $T = t_1 \cdots t_n$, denote by $T_{i\cdots j}$ the substring $t_i \cdots t_j$. Denote by $T^i = T_{i\cdots n}$ the suffix of T starting at location i.

Definition 1. A property π of a string $T = t_1 \ldots t_n$ is a set of intervals $\pi = \{[s_1, f_1], \ldots, [s_t, f_t]\}$ given by the end points of each interval, where for each $1 \le i \le t$: (1) $s_i, f_i \in \{1, \ldots, n\}$, and (2) $s_i \le f_i$. The size of property π, denoted by $|\pi|$, is the number of intervals t in π.