Theoretical Computer Science 638 (2016) 108–111

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Reporting consecutive substring occurrences under bounded gap constraints [☆]

Gonzalo Navarro [a,1], Sharma V. Thankachan [b,*]

[a] Center of Biotechnology and Bioengineering, Department of Computer Science, University of Chile, Chile
[b] School of Computational Science and Engineering, Georgia Institute of Technology, USA

ARTICLE INFO

Article history:
Received 31 March 2015
Received in revised form 14 October 2015
Accepted 5 February 2016
Available online 9 February 2016

Keywords:
Suffix trees
Geometric data structures
Heavy-path decomposition
Pattern matching

ABSTRACT

We study the problem of indexing a text $T[1 \dots n]$ such that whenever a pattern $P[1 \dots p]$ and an interval $[\alpha, \beta]$ come as a query, we can report all pairs (i, j) of consecutive occurrences of P in T with $\alpha \le j - i \le \beta$. We present an $O(n \log n)$ space data structure with optimal $O(p + k)$ query time, where k is the output size.

1. Introduction

Detecting close occurrences of patterns in a text is a problem that has been considered in various flavors. For example, Iliopoulos and Rahman [6] consider the problem of finding all the k occurrences of two patterns P_1 and P_2 (of total length p) separated by a fixed distance α known at indexing time. They gave a data structure using $O(n \log^{\epsilon} n)$ space and query time $O(p + \log\log n + k)$, for any constant $\epsilon > 0$. Bille and Gørtz [2] retained the same space and improved the time to the optimal $O(p + k)$.[2] The problem becomes, however, much messier when we allow the distance between P_1 and P_2 to be in a range $[\alpha, \beta]$, even if these are still known at indexing time. Bille et al. [3] obtained various tradeoffs, for example $O(n)$ space and $O(p + \sigma^{\beta} \log\log n + k)$ time, where σ is the alphabet size; $O(n \log n \log^{\beta} n)$ space and $O(p + (1+\epsilon)^{\beta} \log\log n + k)$ time; and $O(\sigma^{\beta^2} n \log^{\beta} \log n)$ space and $O((p + \beta)(\beta - \alpha) + k)$ time.

Variants of the simpler case where $P_1 = P_2 = P$ have been studied as well. Keller et al. [7] considered the problem of, given an occurrence of P in T, find the next one to the right. They obtained an index using $O(n \log^{\epsilon} n)$ space and $O(\log\log n)$ time. Another related problem they studied was to find a maximal set of nonoverlapping occurrences of P. They obtained the same space and $O(\log\log n + k)$ time. Muthukrishnan [8] considered a document-based version of the problem: T is divided into documents, and we want to report all the k documents where two occurrences of P appear at distance at most β. For β fixed at indexing time, he obtained $O(n)$ space and optimal $O(p + k)$ time; the space raises to $O(n \log n)$ when β is given as a part of the query. Finally, Brodal et al. [4] considered the related pattern mining problem:

[☆] A conference version of this paper appeared in *Proc. CPM 2015*.
[*] Corresponding author.
E-mail addresses: gnavarro@dcc.uchile.cl (G. Navarro), sharma.thankachan@gatech.edu (S.V. Thankachan).
[1] Funded with Basal Funds FB0001, CONICYT, Chile.
[2] This is optimal in the RAM model if we assume a general alphabet of size $O(n)$.

http://dx.doi.org/10.1016/j.tcs.2016.02.005
0304-3975/

find all z maximal patterns P that appear at least twice in T, separated by a distance in $[\alpha, \beta]$. They obtain $O(n \log n + z)$ time, within $O(n)$ space.

In this paper we focus on a rather clean variant of the problem, which (somewhat surprisingly) has not been considered before: find the pairs of consecutive positions of P in T, which are separated by a distance in the range $[\alpha, \beta]$. It is formally stated as follows.

Problem 1. Index a text $T[1 \ldots n]$, such that whenever a pattern $P[1 \ldots p]$ and a range $[\alpha, \beta]$ come as a query, we can report all pairs (i, j) of consecutive occurrences of P in T with $\alpha \le j - i \le \beta$.

Note that we are not finding pairs of occurrences at distances in $[\alpha, \beta]$ if they are not consecutive. For example, for $[\alpha, \beta] = [4, 6]$ and $P =$ abc, we will find the pair of positions $(7, 12)$ in $T =$ abcabcabcdeabc, but not $(1, 7)$, since the occurrences at 1 and 7, while within the distance range, are not consecutive.

By using heavy-path decompositions on suffix trees and geometric data structures, we obtain the following result.

Theorem 1. *There exists an $O(n \log n)$ space data structure with query time $O(p + k)$ for Problem 1, where k is the output size.*

2. Notation and preliminaries

The ith leftmost character of T is denoted by $T[i]$, where $1 \le i \le n$. The sub-string starting at location i and ending at location j is denoted by $T[i \ldots j]$. A suffix is a substring that ends at location n and a prefix is a string that starts at location 1.

The *suffix tree* (ST) of T is a compact representation of all suffixes of $T \circ \$$, except \$, in the form of a compact trie [10]. Here \$ a special symbol that does not appear anywhere in T and $T \circ \$$ is the concatenation of T and \$. The number of leaves in ST is exactly n. The degree of an internal node is at least two. We use ℓ_i to represent the ith leftmost leaf in ST. The edges are labeled with characters and the concatenation of edge labels on the path from root to a node u is denoted by $\mathsf{path}(u)$. Then, $\mathsf{path}(\ell_i)$ corresponds to the ith lexicographically smallest suffix of T, and its starting position is denoted by $\mathsf{SA}[i]$. The locus of a pattern P in T, denoted by $\mathsf{locus}(P)$, is the highest node u in ST, such that P is a prefix of $\mathsf{path}(u)$. The set of occurrences of P in T is given by $\mathsf{SA}[i]$ over all i's, where ℓ_i is in the subtree of $\mathsf{locus}(P)$. The space occupied by ST is $O(n)$ words and the time for finding the locus of an input pattern P is $O(|P|)$. Additionally, for two nodes u and v, we shall use $\mathsf{lca}(u, v)$ to denote their lowest common ancestor.

We now describe the concept of *heavy path* and *heavy path decomposition*. The heavy path of ST is the path starting from the root, where each node u on the path is the child with the largest subtree size (measured as number of leaves in it; ties are broken arbitrary). The *heavy path decomposition* is the operation where we decompose each off-path subtree of the heavy path recursively. As a result, any $\mathsf{path}(\cdot)$ in ST will be partitioned into disjoint heavy paths. Sleator and Tarjan [9] proved the following property; we will use $\log n$ to denote logarithm in base 2.

Lemma 1. *The number of heavy paths intersected by any root to leaf path is at most $\log n$, where n is the number of leaves in the tree.*

Each node belongs to exactly one heavy path and each heavy path contains exactly one *leaf* node. The heavy path containing ℓ_i will be called the i-th heavy path (and identified simply by the number i). For an internal node u, let $\mathsf{hp}(u)$ be the unique heavy path that contains u.

Definition 1. The set $\mathcal{H}_i$ is defined as the set of all leaf identifiers j, where the path from root to ℓ_j intersects with the i-th heavy path. That is, $\mathcal{H}_i = \{j \mid \mathsf{hp}(\mathsf{lca}(\ell_j, \ell_i)) = i\}$.

Lemma 2. $\sum_{i=i}^{n} |\mathcal{H}_i| \le n \log n$.

Proof. For any particular j, path from root to ℓ_j can intersect at most $\log n$ heavy paths, by Lemma 1. Therefore, j cannot be a part of more than $\log n$ sets. □

3. The data structure

The key idea is to reduce our pattern matching problem to an equivalent geometric problem. Specifically, to the *orthogonal segment intersection problem*.

Definition 2 *(Orthogonal segment intersection).* A horizontal segment (x_i, x'_i, y_i) is a line connecting the 2D points (x_i, y_i) and (x'_i, y_i). A segment intersection problem asks to pre-process a given set $\mathcal{S}$ of horizontal segments into a data structure, such that whenever a vertical segment (x'', y', y'') comes as a query, we can efficiently report all the horizontal segments in $\mathcal{S}$ that intersect with the query segment. Specifically, we can output the following set: $\{(x_i, x'_i, y_i) \in \mathcal{S} \mid x_i \le x'' \le x'_i, y' \le y_i \le y''\}$.