



# Reporting consecutive substring occurrences under bounded gap constraints <sup>☆</sup>



Gonzalo Navarro <sup>a,1</sup>, Sharma V. Thankachan <sup>b,\*</sup>

<sup>a</sup> Center of Biotechnology and Bioengineering, Department of Computer Science, University of Chile, Chile

<sup>b</sup> 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  $\alpha \leq j - i \leq \beta$ . We present an  $O(n \log n)$  space data structure with optimal  $O(p + k)$  query time, where  $k$  is the output size.

© 2016 Elsevier B.V. All rights reserved.

## 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  $\alpha$  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  $O(p + k)$ .<sup>2</sup> 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  $\sigma$  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  $\beta$ . For  $\beta$  fixed at indexing time, he obtained  $O(n)$  space and optimal  $O(p + k)$  time; the space raises to  $O(n \log n)$  when  $\beta$  is given as a part of the query. Finally, Brodal et al. [4] considered the related pattern mining problem:

<sup>☆</sup> 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).

<sup>1</sup> Funded with Basal Funds FB0001, CONICYT, Chile.

<sup>2</sup> This is optimal in the RAM model if we assume a general alphabet of size  $O(n)$ .

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

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  $i$ th leftmost character of  $T$  is denoted by  $T[i]$ , where  $1 \leq i \leq n$ . The sub-string starting at location  $i$  and ending at location  $j$  is denoted by  $T[i \dots j]$ . 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  $\ell_i$  to represent the  $i$ th 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  $\text{path}(u)$ . Then,  $\text{path}(\ell_i)$  corresponds to the  $i$ th lexicographically smallest suffix of  $T$ , and its starting position is denoted by  $\text{SA}[i]$ . The locus of a pattern  $P$  in  $T$ , denoted by  $\text{locus}(P)$ , is the highest node  $u$  in ST, such that  $P$  is a prefix of  $\text{path}(u)$ . The set of occurrences of  $P$  in  $T$  is given by  $\text{SA}[i]$  over all  $i$ 's, where  $\ell_i$  is in the subtree of  $\text{locus}(P)$ . 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  $\text{lca}(u, v)$  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  $\text{path}(\cdot)$  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  $\ell_i$  will be called the  $i$ -th heavy path (and identified simply by the number  $i$ ). For an internal node  $u$ , let  $\text{hp}(u)$  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  $\ell_j$  intersects with the  $i$ -th heavy path. That is,  $\mathcal{H}_i = \{j \mid \text{hp}(\text{lca}(\ell_j, \ell_i)) = i\}$ .

**Lemma 2.**  $\sum_{i=1}^n |\mathcal{H}_i| \leq n \log n$ .

**Proof.** For any particular  $j$ , path from root to  $\ell_j$  can intersect at most  $\log n$  heavy paths, by [Lemma 1](#). Therefore,  $j$  cannot be a part of more than  $\log n$  sets.  $\square$

## 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:  $\{(x_i, x'_i, y_i) \in \mathcal{S} \mid x_i \leq x'' \leq x'_i, y' \leq y_i \leq y''\}$ .

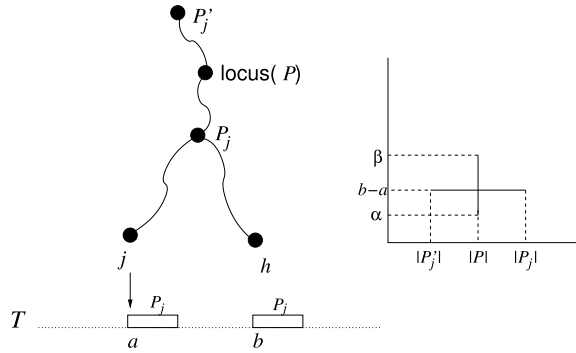


Fig. 1. Illustration of the main concepts of our data structure.

There exists an  $O(|\mathcal{S}|)$  space and  $O(\log \log |\mathcal{S}| + k)$  time solution for the segment intersection problem, where  $k$  is the output size [5, Cor. 4.2(a)]. We now proceed to describe the reduction.

### 3.1. Reduction

One of the main components of our data structure is the suffix tree  $ST$  of  $T$ , and is used only for finding the locus of  $P$ . Based on the heavy path on which the locus node is, we categorize the queries in different types.

**Definition 3.** A query with input pattern  $P$  is type- $h$  if  $h = \text{hp}(\text{locus}(P))$ .

Let  $G_h$  be the data structure handling type- $h$  queries, where  $G_h$  is a structure over a set  $\mathcal{I}_h$  of horizontal segments, that can efficiently answer segment intersection queries. The set  $\mathcal{I}_h$  is generated from  $\mathcal{H}_h$  using the following steps for each  $j \in \mathcal{H}_h$ :

1. Let  $P_j = \text{path}(\text{lca}(\ell_h, \ell_j))$ .
2. Let  $\text{suc}(j)$  be the first occurrence of  $P_j$  after the position  $\text{SA}[j]$  in  $T$  and let  $\text{pre}(j)$  be the last occurrence of  $P_j$  before the position  $\text{SA}[j]$  in  $T$ . Clearly, neither in  $[(\text{pre}(j) + 1) \dots (\text{SA}[j] - 1)]$ , nor in  $[(\text{SA}[j] + 1) \dots (\text{suc}(j) - 1)]$ ,  $P_j$  has an occurrence.
3. Now, obtain two segments w.r.t.  $j$  as follows:
  - (a) Let  $P'_j$  be the shortest prefix of  $P_j$  without any occurrence in  $[(\text{pre}(j) + 1) \dots (\text{SA}[j] - 1)]$ . Then, create segment  $(x_i, x'_i, y_i) = (|P'_j|, |P_j|, \text{SA}[j] - \text{pre}(j))$  and associate the pair  $(\text{pre}(j), \text{SA}[j])$  of consecutive occurrences of  $P_j$  as satellite information.
  - (b) Similarly, let  $P''_j$  be the shortest prefix of  $P_j$  without any occurrence in  $[(\text{SA}[j] + 1) \dots (\text{suc}(j) - 1)]$ . Then, create segment  $(x_i, x'_i, y_i) = (|P''_j|, |P_j|, \text{suc}(j) - \text{SA}[j])$  and associate it to the pair  $(\text{SA}[j], \text{suc}(j))$  of consecutive occurrences of  $P_j$  as satellite information.

Clearly,  $|\mathcal{I}_h| = 2|\mathcal{H}_h|$ . The central idea of our solution is summarized below. Fig. 1 illustrates the idea.

**Lemma 3.** Let  $P$  and  $[\alpha, \beta]$  be the input parameters of a query in Problem 1 and let  $h = \text{hp}(\text{locus}(P))$ . Then, the set of satellite information associated with all those horizontal segments in  $\mathcal{I}_h$ , which are stabbed by a vertical segment  $(p, \alpha, \beta)$  (i.e., the segment connecting the points  $(p, \alpha)$  and  $(p, \beta)$ ) forms the output to Problem 1.

**Proof.** First we prove that any satellite information  $(a, b)$  reported by the geometric query on  $G_h$  is an answer to the original query. Let  $[s, e]$  be the  $x$ -interval corresponding to the reported satellite information  $(a, b)$ . Then,  $s \leq p \leq e$  and  $\alpha \leq b - a \leq \beta$ . Here the condition  $e \geq p$  ensures that both  $\ell_{\text{SA}^{-1}[a]}$  and  $\ell_{\text{SA}^{-1}[b]}$  are leaves in the subtree of  $\text{locus}(P)$ . Therefore  $a$  and  $b$  are occurrences of  $P$ . The condition  $s \leq p$  ensures that there exists no occurrence of  $P$  in any location which is after  $a$ , but before  $b$  (i.e.,  $a$  and  $b$  are consecutive occurrences of  $P$ ). Finally the  $y$ -coordinate ensures that  $\alpha \leq b - a \leq \beta$ .

Now we prove that for every output  $(a, b)$  of Problem 1, there exists a segment  $(s, e, b - a)$  in  $\mathcal{I}_h$  with  $s \leq p \leq e$  and satellite information  $(a, b)$ . Without loss of generality, let  $\text{lca}(\ell_h, \ell_{\text{SA}^{-1}[a]})$  be either  $\text{lca}(\ell_h, \ell_{\text{SA}^{-1}[b]})$  or an ancestor of it. Then, let  $j = \text{SA}^{-1}[a]$ . Since  $P$  occurs at position  $a$ , the leaf  $j$  descends from the subtree of  $\text{locus}(P)$ , and since this node belongs to the heavy path  $h$ , we have that  $\text{lca}(\ell_h, \ell_j)$  descends from  $\text{locus}(P)$ , thus  $e \geq p$ . Since there is no occurrence of  $P$  between  $a$  and  $b$ , it holds  $s \leq p$ . Then, a segment of the form  $(s, e, b - a)$  will indeed be created while processing  $j \in \mathcal{H}_h$  during the construction of  $\mathcal{I}_h$ .  $\square$

In the light of Lemma 3, we have the following result.

**Lemma 4.** *There exists an  $O(n \log n)$  space and  $O(p + \log \log n + k)$  query time solution for Problem 1, where  $k$  is the output size.*

**Proof.** The space of ST is  $O(n)$  and the space required for maintaining the segment intersection structure over  $\mathcal{I}_h$ , for all values of  $h$ , is  $O(\sum_h |\mathcal{I}_h|) = O(\sum_h |\mathcal{H}_h|) = O(n \log n)$ . Thus, the total space is  $O(n \log n)$  words. To answer a query, we first find the locus of  $P$  in ST in  $O(p)$  time, and then query  $G_h$ , where  $h = \text{hp}(\text{locus}(P))$ , in  $O(\log \log n + k)$  time. Therefore, the query time is  $O(p + \log \log n + k)$ .  $\square$

The query time in Lemma 4 is optimal if  $p \geq \log \log n$ . To handle queries where  $p$  is shorter than  $\log \log n$ , we use a different approach.

### 3.2. Achieving optimal query time

We present an optimal query time data structure for  $p < \log \log n$ . Essentially, we associate a data structure  $D(u)$  with each node  $u$  in ST, whose string depth (i.e.,  $|\text{path}(u)|$ ) is at most  $\log \log n$ . Observe that the number of occurrences of  $\text{path}(u)$  in  $T$  is equal to  $\text{size}(u)$ , where  $\text{size}(u)$  is the number of leaves in the subtree of  $u$ . Therefore, the number of consecutive occurrences  $(i, j)$  of  $\text{path}(u)$  is  $\text{size}(u) - 1$ . Each such pair  $(i, j)$  can be mapped to a point  $(j - i)$  in one dimension along with the pair  $(i, j)$  as an associated satellite data. We then create a one-dimensional range reporting data structure over these  $(\text{size}(u) - 1)$  points and call it  $D(u)$ . Whenever the locus of  $P$  is  $u$ , the answer can be obtained by issuing a one dimensional range reporting query on  $D(u)$  with  $[\alpha, \beta]$  as the input range. The satellite data associated with each reported corresponds to an answer to Problem 1.

We use the data structure summarized in Lemma 5, by which queries can be answered in optimal time and the space of  $D(u)$  can be bounded by  $O(\text{size}(u))$  words.

**Lemma 5.** (See [1].) *One dimensional range reporting queries over a set of  $m$  points in  $\{0, 1, 2, \dots, 2^w\}$  can be answered in optimal time using an  $O(m)$  space data structure, where  $w$  is the word size.*

Note that the sum of all the  $\text{size}(u)$  terms for all the nodes  $u$  with the same string depth is  $n$ , and added over all the nodes with string depth up to  $\log \log n$  is  $n \log \log n$ . Thus the space for the  $D(\cdot)$  structures of all the nodes with string depth up to  $\log \log n$  is  $O(n \log \log n)$  words. This completes the proof of Theorem 1.

## 4. Conclusions

We have addressed a clean variant of the problem related to finding close occurrences of a pattern  $P[1 \dots p]$  in a text  $T[1 \dots n]$ : find pairs of consecutive occurrences that are within a distance range  $[\alpha, \beta]$  (given at query time). Our data structure uses  $O(n \log n)$  space and optimal  $O(p + k)$  query time.

It is not hard to extend our result to the case where we look for the occurrence of  $P$  followed (or preceded) by some function of  $P$ , such as its reverse string. We can build the geometric structure at each suffix tree node  $v$  considering the function of the string represented by  $v$ , instead of the string itself. However, extending our solution to the general case of two patterns [3] seems not possible.

An interesting open question is whether our optimal-time result can be obtained within  $o(n \log n)$  space. Another question is how our results can be extended to the document retrieval scenario, that is, listing the documents where two consecutive occurrences of  $P$  appear separated by a distance in  $[\alpha, \beta]$ . The current result [8] is similar to ours in space and time, but it is restricted to the case  $\alpha = 0$ . It is not clear if the problem is harder, and by how much, for an arbitrary value of  $\alpha$ .

## References

- [1] S. Alstrup, G. Stølting Brodal, T. Rauhe, Optimal static range reporting in one dimension, in: Proc. 33rd Annual ACM Symposium on Theory of Computing, STOC, 2001, pp. 476–482.
- [2] P. Bille, I.L. Gørtz, Substring range reporting, *Algorithmica* 69 (2) (2014) 384–396.
- [3] P. Bille, I.L. Gørtz, H.W. Vildhøj, S. Vind, String indexing for patterns with wildcards, *Theory Comput. Syst.* 55 (1) (2014) 41–60.
- [4] G.S. Brodal, R.B. Lyngsø, C.N.S. Pedersen, J. Stoye, Finding maximal pairs with bounded gap, in: Proc. 10th Annual Symposium on Combinatorial Pattern Matching, CPM, in: LNCS, vol. 1645, 1999, pp. 134–149.
- [5] T.M. Chan, Persistent predecessor search and orthogonal point location on the word RAM, *ACM Trans. Algorithms* 9 (3) (2013), article 22.
- [6] C.S. Iliopoulos, M.S. Rahman, Indexing factors with gaps, *Algorithmica* 55 (1) (2009) 60–70.
- [7] O. Keller, T. Kopelowitz, M. Lewenstein, Range non-overlapping indexing and successive list indexing, in: Proc. 10th International Workshop on Algorithms and Data Structures, WADS, in: LNCS, vol. 4619, 2007, pp. 625–636.
- [8] S. Muthukrishnan, Efficient algorithms for document retrieval problems, in: Proc. 13th Annual ACM–SIAM Symposium on Discrete Algorithms, SODA, 2002, pp. 657–666.
- [9] D.D. Sleator, R.E. Tarjan, A data structure for dynamic trees, *J. Comput. Syst. Sci.* 26 (3) (1983) 362–391.
- [10] P. Weiner, Linear pattern matching algorithms, in: Proc. 14th Annual Symposium on Switching and Automata Theory, 1973, pp. 1–11.