Journal of Discrete Algorithms 17 (2012) 109–114

Contents lists available at SciVerse ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

On position restricted substring searching in succinct space ☆

Wing-Kai Hon [a], Rahul Shah [b], Sharma V. Thankachan [b,*], Jeffrey Scott Vitter [c]

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

ARTICLE INFO

Article history:
Received 25 June 2012
Received in revised form 27 September 2012
Accepted 27 September 2012
Available online 2 October 2012

Keywords:
Succinct data structures
Pattern matching
Range searching

ABSTRACT

We study the position restricted substring searching (PRSS) problem, where the task is to index a text $T[0\ldots n-1]$ of n characters over an alphabet set Σ of size σ, in order to answer the following: given a query pattern P (of length p) and two indices ℓ and r, report all $occ_{\ell,r}$ occurrences of P in $T[\ell\ldots r]$. Known indexes take $O(n\log n)$ bits or $O(n\log^{1+\epsilon} n)$ bits space, and answer this query in $O(p+\log n+occ_{\ell,r}\log n)$ time or in optimal $O(p+occ_{\ell,r})$ time respectively, where ϵ is any positive constant. The main drawback of these indexes is their space requirement of $\Omega(n\log n)$ bits, which can be much more than the optimal $n\log\sigma$ bits to store the text T. This paper addresses an open question asked by Mäkinen and Navarro [LATIN, 2006], which is whether it is possible to design a succinct index answering PRSS queries efficiently. We first study the hardness of this problem and prove the following result: a succinct (or a compact) index cannot answer PRSS queries efficiently in the pointer machine model, and also not in the RAM model unless bounds on the well-researched orthogonal range query problem improve. However, for the special case of sufficiently long query patterns, that is for $p=\Omega(\log^{2+\epsilon} n)$, we derive an $|CSA_f|+|CSA_r|+o(n)$ bits index with optimal query time, where $|CSA_f|$ and $|CSA_r|$ are the space (in bits) of the compressed suffix arrays (with $O(p)$ time for pattern search) of T and $\overleftarrow{T}$ (the reverse of T) respectively. The space can be reduced further to $|CSA_f|+o(n)$ bits with a resulting query time will be $O(p+occ_{\ell,r}+\log^{3+\epsilon} n)$. For the general case, where there is no restriction on pattern length, we obtain an $O(\frac{1}{\epsilon^3}n\log\sigma)$ bits index with $O(p+occ_{\ell,r}+n^{\epsilon})$ query time. We use suffix sampling techniques to achieve these space-efficient indexes.

1. Introduction

Given a text $T[0\ldots n-1]$ of size n over an alphabet set Σ of size σ, the fundamental problem in text indexing is to preprocess T and maintain an index such that whenever a pattern P (of length p) comes as a query, all *occ* occurrences of P in T can be reported efficiently. Classical indexes such as suffix trees and suffix arrays can answer this query in $O(p+occ)$ and $O(p+\log n+occ)$ time respectively [18,17,16]. 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 the human genome data ($\Sigma=\{A,G,C,T\}$), $\log\sigma$ is 2 whereas $\log n$

☆ Early part of this work appeared in the *Proceedings of Data Compression Conference*, 2008 [6] (Chien et al., 2008). 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 (Rahul Shah).
* Corresponding author. Postal address: 291 Coates Hall, LSU Computer Science, Baton Rouge, LA 70803, USA. Tel.: +1 225 578 4355; fax: +1 225 578 1465.
E-mail addresses: wkhon@cs.nthu.edu.tw (W.-K. Hon), rahul@csc.lsu.edu (R. Shah), thanks@csc.lsu.edu (S.V. Thankachan), jsv@ku.edu (J.S. Vitter).

1570-8667/$ – see front matter
http://dx.doi.org/10.1016/j.jda.2012.09.002

is around 30. This gives a clear motivation to design succinct[1] or compressed indexes, as such indexes may fit in faster but space-limited main memory. This long standing problem was positively answered via the compressed suffix array by Grossi and Vitter [10] and the FM-index by Ferragina and Manzini [8] (see [15] for an excellent survey). While these indexes handle general pattern matching queries efficiently in compressed space, our focus is on designing space efficient indexes for a special problem called *Position-restricted substring searching* (PRSS) defined as follows:

> The query input consists of a pattern P (of length p) and two indices ℓ and r, and the task is to report all $occ_{\ell,r}$ occurrences of P in $T[\ell\ldots r]$.

PRSS queries are fundamental in many text searching applications, where the objective is to search only a part of the text collection. Examples include restricting the search to a subset of dynamically chosen documents in a document database, restricting the search to only parts of a long DNA sequence, etc. [14]. This problem also finds applications in information retrieval.

1.1. Related work

The PRSS problem was introduced by Mäkinen and Navarro [14]. They proposed an $O(n\log n)$ bits index with a query time of $O(p+\log n+occ_{\ell,r}\log n)$, where $occ_{\ell,r}$ is the number of occurrences of P in $T[\ell\ldots r]$. Similar work has been done by Hon et al. [12]. Mäkinen and Navarro [14] proposed an $O(n\log^{1+\epsilon} n)$ bits index as well (where ϵ is any positive constant) with a near optimal query time of $O(p+\log\log n+occ_{\ell,r})$, which was improved to $O(p+\log\log\sigma+occ_{\ell,r})$ by Kopelowitz et al. [13] and further to optimal $O(p+occ_{\ell,r})$ by Bille and Gørtz [3]. Another optimal query time solution is by Crochemore et al. [7], however its space requirement is $O(n^{1+\epsilon})$ bits.

The counting version of *PRSS* is also a well studied problem. The $O(n\log n)$ bits index by Mäkinen and Navarro [14] can count the number of occurrences $occ_{\ell,r}$ in $O(p+\log n)$ time. A solution by Bille and Gørtz [3] takes the space of $O(n\log^2 n/\log^2\log n)$ bits and can answer queries in $O(p+\log\log n)$ time. Another solution by Kopelowitz et al. [13] slightly improves this query time to $O(p+\log\log\sigma)$, while maintaining the same space bounds. Recently, Gagie and Gawrychowski proposed an $O(n\log n)$ bits solution with $(p+\log\log n)$ query time, which can be improved to optimal $O(p)$ if $\sigma=\log^{O(1)} n$ [9].

1.2. Our contributions

In this paper, we investigate the possibility of deriving space efficient indexes (i.e., taking $O(n\log\sigma)$ bits space instead of $\Omega(n\log n)$ bits) for the PRSS problem. The following are our main results:

- Designing an $O(n\log\sigma)$ bits index which can answer PRSS queries in *poly-logarithmic* time is at least as hard as designing a linear space structure performing 3-dimensional orthogonal range reporting in *poly-logarithmic* time.
- On a pointer machine, we derive a lower bound of $\Omega(n\log^2 n/\log^2\log n)$ bits of space for any index with $O(p\log^{O(1)} n+occ_{\ell,r})$ query time.
- For the special case when $p=\Omega(\log^{2+\epsilon} n)$, we develop an $|CSA_f|+|CSA_r|+o(n)$ bits index with optimal $O(p+occ_{\ell,r})$ query time, where $|CSA_f|$ and $|CSA_f|$ are the space (in bits) of the compressed suffix arrays (with $O(p)$ time for pattern search) of T and $\overleftarrow{T}$ (the reverse of T) respectively and ϵ is any positive constant. The index space can be further reduced to $|CSA_f|+o(n)$ bits, with a resulting query time of $O(p+occ_{\ell,r}+\log^{3+\epsilon} n)$.
- We develop an $O(\frac{1}{\epsilon^3}n\log\sigma)$ bits index that can answer PRSS queries in $O(p+occ_{\ell,r}+n^{\epsilon})$ time without any restriction on pattern length.

1.3. Organization of the paper

Section 2 is dedicated to preliminaries. In Section 3, we prove the hardness of the PRSS problem and also derive a space lower bound in the pointer machine model for any index with query time $O(p\log^{O(1)}+occ_{\ell,r})$. In Section 4, we design our compressed index with optimal query time for the special case when the query pattern is sufficiently long. Finally, in Section 5, we introduce a succinct (compact) index which does not have any restriction on pattern length. However, in this case, the query time is sub-linear rather than poly-logarithmic.

[1] In this paper, we use the terms *succinct* and *compact* interchangeably to mean $O(n\log\sigma)$ bit index. The term *compressed* refers to entropy based compression.

2. Preliminaries

2.1. Suffix arrays and compressed suffix arrays

For the text $T[0 \ldots n-1]$ to be indexed, each substring $T[i \ldots n-1]$, with $i \in [0, n-1]$, is called a *suffix* of T. The suffix array $SA[0 \ldots n-1]$ is an array of length n, where $SA[i]$ is the starting position (in T) of the ith lexicographically smallest suffix of T [16]. An important property of SA is that the starting positions of all suffixes with the same prefix are always stored in a contiguous region in SA. Based on this property, we define the *suffix range* of a pattern P to be the maximal range $[a, b]$ in SA, such that for all $j \in [a, b]$, $SA[j]$ is the starting point of a suffix of T with P as a prefix. The suffix array along with a data structure called the longest common prefix (LCP) array takes $O(n \log n)$ bits space and can find the suffix range of a pattern P in $O(p + \log n)$ time.

The compressed suffix array (CSA) is a space-efficient version of the suffix array which takes space close to the size of the text [10,8,1,15]. For our purpose, we shall assume the $nH_k + O(n) + o(n \log \sigma)$-bits CSA by Belazzougui and Navarro [1], which can find the locus of P in $O(p)$ time, even when the alphabet size is large. Here H_k represents the kth order empirical entropy of the text. Using backward search, the suffix ranges of all suffixes of the pattern P can be computed in $O(p)$ time.

2.2. Orthogonal range reporting in 3-D (RR3D)

Let S be a given set of n points of the form (x_i, y_i, z_i) in a $[0, n-1] \times [0, n-1] \times [0, n-1]$ grid. An orthogonal range reporting query consists of three input ranges $(x', x''), (y', y'')$ and (z', z'') and the task is to output all those points (x_j, y_j, z_j) such that, $x' \leqslant x_j \leqslant x'', y' \leqslant y_j \leqslant y''$ and $z' \leqslant z_j \leqslant z''$. The best known data structure for this problem in the RAM model is by Chan et al. [4], having a space requirement of $O(n \log^{2+\epsilon} n)$ bits and query time of $O(\log \log n + |output|)$, where $|output|$ is the output size.

3. Hardness of PRSS problem

In this section, we first prove the hardness of the PRSS problem.

Theorem 1. *Designing a succinct (compact) index answering PRSS queries in poly-logarithmic time is at least as hard as designing a linear space data structure performing 3-dimensional orthogonal range reporting in poly-logarithmic time.*

Assuming a poly-logarithmic query time structure for PRSS in succinct space, we shall show how 3-dimensional range searching can be made poly-logarithmic in linear space.

Let S be a set of n points of the form (x_i, y_i, z_i) for $i = 0, 1, 2, \ldots, n-1$ in a $[0, n-1] \times [0, n-1] \times [0, n-1]$ grid, such that $z_i \leqslant z_{i+1}$. Note that each point in S can be represented using $O(\log n)$ bits. Let $\langle s \rangle$ denote the binary string representing an integer $s \in [0, n-1]$ and let $\overleftarrow{\langle s \rangle}$ denote the reverse of $\langle s \rangle$. Now we construct the following string T', which consists of $O(n \log n)$ characters taken from an alphabet set $\Sigma = \{0, 1, \#, \star\}$ and is formed by concatenating the following:

$$T' = \overleftarrow{\langle y_1 \rangle} \# \langle x_1 \rangle \star \overleftarrow{\langle y_2 \rangle} \# \langle x_2 \rangle \star \cdots \star \overleftarrow{\langle y_n \rangle} \# \langle x_n \rangle \star$$

Assume there exists a succinct (or a compact) data structure answering PRSS queries in *poly-logarithmic* time. Then, we first index T' (having constant alphabet size) in $O(|T'|) = O(n \log n)$ bits space. Now consider a 3-dimensional range reporting query (RR3D) on S, where we need to output all the points within the query box $[x', x''] \times [y', y''] \times [z', z'']$. We prove that this RR3D on S can be reduced to $O(\log^2 n)$ PRSS queries on T'. The following lemma shows how to generate patterns corresponding to PRSS queries.

Lemma 1. *A given range $[a, b]$, where $0 \leqslant a \leqslant b \leqslant n-1$ can be represented by a set $S' = \{s_1, s_2, \ldots, s_k\}$ of $k \leqslant 2 \log n$ binary strings. None of these strings is a prefix of another. For any integer j, $j \in [a, b]$ if and only if there exists $s_i \in S'$ such that s_i is a prefix of $\langle j \rangle$.*

Proof. Let Λ be a trie of binary representations of all integers $\in [0, n-1]$, such that the ith leftmost leaf represents integer $i-1$. Now any range $[a, b]$ on leaves in Λ can be split into $k \leqslant 2 \log n$ non-overlapping sub-ranges such that each of these sub-range represents the complete sub-tree of a unique node u in Λ. Then S' is the set of *paths*(u)'s of all such nodes, where *path*(u) is the concatenation of edge labels (0 or 1) in the path from root to u. □

Let S'_X, and S'_Y represent the set of binary strings (constructed using Lemma 1) corresponding to the intervals $[x', x'']$ and $[y', y'']$ respectively. Then, we generate a set S'_P of binary strings as follows: $S'_P = \{\overleftarrow{s_y} \# s_x | s_x \in S'_X, s_y \in S'_Y\}$. Note that we can have $O(\log^2 n)$ combinations of s_x and s_y, hence S'_P consists of $O(\log^2 n)$ distinct binary strings of length $O(\log n)$.

Lemma 2. *If an occurrence of a string $P \in S'_P$ overlaps with the ith # symbol in T', then $x' \leqslant x_i \leqslant x''$ and $y' \leqslant y_i \leqslant y''$.*

Proof. Let $P = \overleftarrow{s_y}\#s_x$, where $s_x \in S'_X$ and $s_y \in S'_Y$. Then s_x is a prefix of $\langle x_i \rangle$ and s_y is a prefix of $\langle y_i \rangle$. Hence the result follows from Lemma 1. $\square$

Let ℓ and r be such that $z_\ell < z' \leqslant z_{\ell+1} \leqslant z_r \leqslant z'' < z_{r+1}$. In order to report the points within $[x', x''] \times [y', y''] \times [z', z'']$, it is enough to find those occurrences of $P \in S'_P$ within $T'[\ell', r']$, where ℓ' is the starting location of the ℓth $\star$ in T' and r' is the starting location of the rth $\star$ in T' (proof follows from the definition of T' and Lemma 2). Thus we have a total of $O(\log^2 n)$ PRSS queries with patterns of length $O(\log n)$. Therefore, if there exists an $O(n\log\sigma)$ bits index with $O(p\log^c n + |output|\log^d n)$ query time for PRSS queries, then there exists an $O(n\log n)$ bits data structure performing RR3D in $O(\log^{c+3} n + |output|\log^d n)$ time, where c and d are any constants. This completes the proof of Theorem 1.

Theorem 2. *Any data structure supporting PRSS queries in $O(p\log^{O(1)} n + occ_{\ell,r})$ time in pointer machine model has to use $\Omega(n\log^2 n/\log^2\log n)$ bits space.*

Proof. Assume that there exists an $o(n\log^2 n/\log^2\log n)$ bits index answering PRSS queries in $O(p\log^c n + occ_{\ell,r})$ time. Then we can construct an $o(n\log^2 n/\log^2\log n)$ words data structure supporting RR3D queries in time $O(\log^{c+3} n + |output|)$ (follows from the proof of Theorem 1). In the pointer machine model, this contradicts the following lower bound by Chazelle: any data structure supporting RR3D queries in $O(\log^{O(1)} n + |output|)$ has to use at least $\Omega(n\log^2 n/\log^2\log n)$ words space [5]. $\square$

4. Optimal-time compressed index for long patterns

In this section, we prove the following.

Theorem 3. *There exists an $|CSA_f| + |CSA_r| + o(n)$ bits index supporting PRSS queries in optimal $O(p + occ_{\ell,r})$ time for $p = \Omega(\log^{2+\epsilon} n)$.*

Let $\alpha = \Theta(\log^{2+\epsilon/2} n)$ be a sampling factor. We introduce the following definitions:

- *α-sampled suffix*: $T[x \ldots n-1]$ is an α-sampled suffix if $x \equiv 0 \pmod{\alpha}$.
- *α-sampled prefix*: $T[0 \ldots x-1]$ is an α-sampled prefix if $x \equiv 0 \pmod{\alpha}$.
- *Offset-t occurrence*: An occurrence of P at position i in T (i.e., $P = T[i \ldots i+p-1]$) is an offset-t occurrence, if $i \equiv t \pmod{\alpha}$.

Therefore, an offset-0 occurrence of P is always a prefix of an α-sampled suffix of T. The following is true for an offset-t occurrence with $1 \leqslant t < p$ (hence it is true for $p \geqslant \alpha$ as $\alpha > t$): the prefix $P[0 \ldots t-1]$ of P (of length t) is a suffix of an α-sampled prefix $T[0 \ldots x-1]$ and the suffix $P[t \ldots p-1]$ of P (of length $p-t$) is a prefix of an α-sampled suffix $T[x \ldots n-1]$.

4.1. Our index

The construction of our index is based on the assumption that $p \geqslant \alpha$. It consists of the following components:

- *CSA_f*: compressed suffix array of T.
- *CSA_r*: compressed suffix array of $\overleftarrow{T}$, where $\overleftarrow{T}$ is the reverse of T (i.e., $\overleftarrow{T}[i] = T[n-1-i]$).
- *RR3D Structure:* For each α-sampled suffix $T[x \ldots n-1]$, we define a triplet (x, y, z) such that y is the lexicographic rank of $T[x \ldots n-1]$ among all suffixes of T and z be the lexicographic rank of $T[0 \ldots x-1]$ among the reverse of all prefixes of T. Therefore, the yth leftmost entry (i.e., corresponding to the yth leftmost leaf in its suffix tree) in CSA_f corresponds to $T[x \ldots n-1]$ and the zth leftmost entry in CSA_r corresponds to $T[0 \ldots x-1]$. Since we have $\Theta(n/\alpha)$ α-sampled suffixes, the number of triplets is bounded by $O(n/\alpha)$. The size of an RR3D structure [4] maintained over these triplets can be bounded by $O((n/\alpha)\log^{2+\epsilon'} n) = o(n)$ bits (assume $\epsilon' < \epsilon/2$).

Putting all space terms together, we have the following lemma.

Lemma 3. *The total space of our index is $|CSA_f| + |CSA_r| + o(n)$ bits.*

4.2. Query answering

Let $[L^f_t, R^f_t]$ be the suffix range of $P[t \ldots p-1]$ in CSA_f. Using backward search on CSA_f, $[L^f_t, R^f_t]$ for $t = 0, 1, 2, \ldots, p-1$ can be computed in $O(p)$ time. Similarly, let $[L^r_t, R^r_t]$ be the suffix range of $P[0 \ldots t-1]$ (where $t \geqslant 1$) in CSA_r. Then, $[L^r_t, R^r_t]$

for $t = 1, 2, \ldots, \alpha - 1$ can be computed using backward search on CSA_r in $O(\alpha)$ time. Our query answering algorithm consists of α steps and in the tth step for $t = 0, 1, 2, \ldots, \alpha - 1$, we retrieve all offset-t occurrences of P in $T[\ell, r]$.

First, we show how to retrieve all offset-0 occurrences. That is, we show how to find all α sampled suffixes $T[x \ldots n-1]$ satisfying the conditions $x \equiv 0 \pmod{\alpha}$, $T[x \ldots x+p-1] = P[0 \ldots p-1]$ and $\ell \leqslant x \leqslant r$. As we have defined a tuple (x, y, z) for each α sampled suffix, the above PRSS query can be reduced to the following geometric range searching problem: report all those tuples (x, y, z) such that $\ell \leqslant x \leqslant r$ and $L_0^f \leqslant y \leqslant R_0^f$. This query can be answered in $O(\log\log n + occ_0)$ time using the RR3D structure, where occ_0 is the number of offset-0 occurrences. Let (x, y, z) be a reported point, then x is an answer to PRSS query. Now, we generalize this to finding offset-t occurrences. Finding all offset-t occurrences for $1 \leqslant t \leqslant \alpha - 1$ can be reduced to the following geometric problem: report all those tuples (x, y, z) such that $\ell + t \leqslant x \leqslant r + t$, $L_t^f \leqslant y \leqslant R_t^f$ and $L_t^r \leqslant z \leqslant R_t^r$. This query can be answered in $O(\log\log n + occ_t)$ time using the RR3D structure, where occ_t is the number of offset-t occurrences. Let (x, y, z) be a reported point, then $x - t$ is an answer to PRSS query. That is, $T[x - t \ldots x - t + p - 1] = P$ and $\ell \leqslant x - t \leqslant r$.

Lemma 4. *For $p = \Omega(\log^{2+\epsilon} n)$, the PRSS query can be answered in optimal $O(p + occ_{\ell,r})$ time, where $occ_{\ell,r}$ is the number of occurrences of P in $T[\ell \ldots r]$.*

Proof. The total time for computing the suffix ranges in CSA_f is $O(p)$ and the time for finding the suffix ranges of $\alpha - 1$ prefixes of P in CSA_r is $O(\alpha)$. The RR3D structure takes $O(\log\log n + occ_t)$ time for reporting all offset-t occurrences within $T[\ell, r]$. Therefore the total time is $O(p + \alpha + \sum_{t=0}^{\alpha-1}(\log\log n + occ_t)) = O(p + \log^{2+\epsilon/2} n \log\log n + occ_{\ell,r}) = O(p + occ_{\ell,r})$ (note that $p = \Omega(\log^{2+\epsilon} n)$). $\square$

By combining Lemmas 3 and 4, we have Theorem 3.

The index space can be further reduced and the following is our result.

Theorem 4. *There exists an $|CSA_f| + o(n)$ bits index supporting PRSS queries in $O(p + occ_{\ell,r} + \log^{3+\epsilon} n)$ time, for $p = \Omega(\log^{2+\epsilon} n)$.*

Proof. Hon et al. [11] showed that CSA_r can be maintained as a sparse suffix tree taking $O(n \log n/\beta)$ bits in addition to CSA_f with an $O(\beta)$ slow down in pattern search. Therefore CSA_r can be replaced by an $o(n)$ bits sparse suffix tree by choosing $\beta = \Theta(\log^{1+\epsilon/2} n)$. The resulting query time will be $O(p + occ_{\ell,r} + \alpha\beta)$. $\square$

5. A compact index for all patterns

In this section, we introduce a compact index for PRSS queries which works without any restriction on the length of the query pattern. This can be seen as an extension of the 2-D technique by Grossi and Vitter [10] to 3 dimensions. The main result is captured in the following theorem.

Theorem 5. *There exists an $O(\frac{1}{\epsilon^3} n \log\sigma)$ bits space and $O(p + occ_{\ell,r} + n^\epsilon)$ query time index for the PRSS problem.*

Firstly, we introduce a new sampling factor $\alpha' = \epsilon \log_\sigma n$, where $0 < \epsilon < 1/2$. Then, an occurrence of P at position i in T is an offset-t occurrence for some $t = i \bmod (\alpha') \in [0, \alpha' - 1]$. Based on pattern length, we categorize the occurrences into two cases and handle both cases separately.

5.1. Case A: offset-t occurrences for t < p

In this case a suffix of P will be a prefix of some α'-sampled suffix. Therefore all such occurrences can be reported using a similar data structure described in the previous section. Because of the new sampling factor α', the number of triplets to be maintained as an RR3D structure is $O(n/\alpha') = O(n \log\sigma/(\epsilon \log n))$. Note that Chan et al.'s RR3D structure is not affordable in this scenario. In Section 5.1, we will introduce an RR3D structure taking $O(\frac{1}{\epsilon^2} n \log n)$ bits space which can answer queries in $O(n^\epsilon + |output|)$ time, where ϵ is any positive constant. By using this data structure, our index space can be bounded by $O((n/\alpha')\frac{1}{\epsilon^2}\log n) = O(\frac{1}{\epsilon^3} n \log\sigma)$ bits.

5.2. Case B: offset-t occurrences for t ⩾ p

This case is necessary when the pattern size $p < \alpha'$, the sampling factor. We call a segment $T[x \ldots x + \alpha' - 1]$ of T of length α' a block $B(x)$ if $x \bmod (\alpha') = 0$. A block is always a prefix (of length α') of an α'-sampled suffix. The number of blocks of T is $\Theta(n/\alpha') = \Theta(n \log\sigma/(\epsilon \log n))$, whereas the number of distinct blocks is $\gamma = \sigma^{\alpha'} = O(n^\epsilon)$ and these are represented as $B_1, B_2, B_3, \ldots, B_\gamma$. Let L_i be the sorted list of starting locations of the block B_i in T. That is, it is the sorted list of all x such that $B(x) = B_i$. Clearly $\sum_{i=1}^{\gamma} |L_i| = O(n/\alpha')$ and the total space for maintaining all such L_i's is $O((n/\alpha')\log n) = O(\frac{1}{\epsilon} n \log\sigma)$ bits. We may call an occurrence of P within the block B_λ as a type-λ occurrence.

Now all the answers to a PRSS query, which are of type-λ (for $1 \leqslant \lambda \leqslant \gamma$) can be retrieved as follows: first search for P in B_λ and find all its occurrences using any standard string searching algorithm taking $O(\alpha') = O(\epsilon \log_\sigma n)$ time. If there is an occurrence of P at position j in $B_\lambda[0 \ldots \alpha' - 1]$, then P occurs at positions $x + j$ for all those x with $B(x) = B_\lambda$. Therefore by using the list L_i, all those type-λ occurrences of P within $T[\ell \ldots r]$ can be quickly computed in $O(\log n + |output|)$ time ($O(\log n)$ is needed for an initial binary search of ℓ in L_i). Repeat this procedure for $\lambda = 1, 2, 3, \ldots, \gamma$, and the total query time can be bounded by $O(p + \gamma \log n + occ_{\ell,r}) = O(p + occ_{\ell,r} + n^\epsilon)$ (by adjusting the constants).

5.3. RR3D structure: $O(\frac{1}{\epsilon^2} n \log n)$ bits space and $O(n^\epsilon + |output|)$ query time

We note that similar structure also appears in [2], but we describe it here for the sake of completion.

We first describe a 2-dimensional range reporting structure (RR2D). The idea is to use a B-tree with $B = \Theta(\epsilon n^\epsilon)$. We may associate each point with a unique leaf node such that, the point with the ith smallest x-coordinate value (ties broken arbitrary) is stored in the ith leftmost leaf. Therefore, at leaf level, the points are maintained in the sorted order of x-coordinate values. An internal node stores all those points associated with the leaves in its sub-tree, which are maintained in the sorted order of y-coordinate. Now, any x range $[x, x'']$ can be decomposed into $O(\frac{1}{\epsilon} B) = O(n^\epsilon)$ non-overlapping ranges, such that the points within each range are maintained at some internal node in the sorted order of y-coordinate. Therefore, an RR2D query can be now decomposed into $O(n^\epsilon)$ RR1D queries on a sorted array. Hence, the time for RR2D can be bounded by $O(n^\epsilon \log n + |output|)$. The tree height is bounded by $O(1/\epsilon)$, hence the total space is $O(\frac{1}{\epsilon} n \log n)$ bits.

Using the RR2D structure described above, our RR3D structure can be constructed as follows: maintain a B-tree with $B = \Theta(\epsilon n^\epsilon)$ and associate each point with a unique leaf node, based on its z-coordinate value. Hence, at leaf level, all points are in the sorted order of z-coordinate. At every internal node, we maintain an RR2D structure (on x and y coordinates) of all those points associated with the leaves in its sub-tree. Using similar arguments as before, any RR3D query can now be decomposed into $O(n^\epsilon)$ RR2D queries, taking $O(n^{2\epsilon} \log^2 n + |output|)$ time. The index space will be $O(1/\epsilon)$ times the space of the RR2D structure, i.e., $O(\frac{1}{\epsilon^2} n \log n)$ bits. The factor of $n^{2\epsilon} \log^2 n$ can be made to n^ϵ by scaling ϵ up by factor of 3. This does not affect the space bounds in terms of the big-O factor.

6. Conclusions

We showed that the position restricted substring searching problem is hard if one wants to derive a succinct or compressed index. However, there are special cases when the pattern size is long or when we are allowed an additive n^ϵ factor in search times. In these cases it is possible to derive space-efficient indexes. There are other unsolved hard problems like 2-dimensional matching or parameterized matching where no succinct solutions exist. An open question is: Can we achieve similar lower bounds and upper bounds (may be for special cases) for some of these problems?

References

[1] D. Belazzougui, G. Navarro, Alphabet-independent compressed text indexing, in: ESA, 2011, pp. 748–759.
[2] J.L. Bentley, H.A. Maurer, Efficient worst-case data structures for range searching, Acta Informatica 13 (1980) 155–168.
[3] P. Bille, L.L. Gørtz, Substring range reporting, in: CPM, 2011, pp. 299–308.
[4] T.M. Chan, K.G. Larsen, M. Patrascu, Orthogonal range searching on the RAM, revisited, in: SoCG, 2011, pp. 1–10.
[5] B. Chazelle, Lower bounds for orthogonal range searching, I: the reporting case, Journal of the ACM 37 (1990) 200–212.
[6] Y.F. Chien, W.K. Hon, R. Shah, J.S. Vitter, Geometric Burrows–Wheeler transform: linking range searching and text indexing, in: DCC, 2008, pp. 252–261.
[7] M. Crochemore, C.S. Iliopoulos, M. Kubica, M.S. Rahman, T. Walen, Improved algorithms for the range next value problem and applications, in: STACS, 2008, pp. 205–216.
[8] P. Ferragina, G. Manzini, Indexing compressed text, Journal of the ACM 52 (4) (2005) 552–581.
[9] T. Gagie, P. Gawrychowski, Linear-space substring range counting over polylogarithmic alphabets. CoRR, arXiv:1202.3208, 2012.
[10] R. Grossi, J.S. Vitter, Compressed suffix arrays and suffix trees with applications to text indexing and string matching, SIAM Journal on Computing 35 (2) (2005) 378–407.
[11] W.K. Hon, T.H. Ku, R. Shah, S.V. Thankachan, J.S. Vitter, Compressed text indexing with wildcards, in: SPIRE, 2011, pp. 267–277.
[12] W.K. Hon, R. Shah, J.S. Vitter, Ordered pattern matching: towards full-text retrieval, Technical Report TR-06-008, Purdue University, March 2006.
[13] T. Kopelowitz, M. Lewenstein, E. Porat, Persistency in suffix trees with applications to string interval problems, in: SPIRE, 2011, pp. 67–80.
[14] V. Mäkinen, G. Navarro, Position-restricted substring searching, in: LATIN, 2006, pp. 703–714.
[15] V. Mäkinen, G. Navarro, Compressed full-text indexes, ACM Computing Surveys 39 (1) (2007).
[16] U. Manber, G. Myers, Suffix arrays: a new method for on-line string searches, SIAM Journal on Computing 22 (5) (1993) 935–948.
[17] E.M. McCreight, A space-economical suffix tree construction algorithm, Journal of the ACM 23 (2) (1976) 262–272.
[18] P. Weiner, Linear pattern matching algorithms, in: SWAT, 1973, pp. 1–11.