Algorithmica (2020) 82:107–117
https://doi.org/10.1007/s00453-019-00605-5

Succinct Non-overlapping Indexing

Arnab Ganguly[1] · Rahul Shah[2] · Sharma V. Thankachan[3]

Received: 4 November 2015 / Accepted: 8 July 2019 / Published online: 30 July 2019

Abstract
Text indexing is a fundamental problem in computer science. The objective is to preprocess a text T, so that, given a pattern P, we can find all starting positions (or simply, occurrences) of P in T efficiently. In some cases, additional restrictions are imposed. We consider two variants, namely the *non-overlapping indexing* problem, and the *range non-overlapping indexing* problem. Given a text T having n characters, the non-overlapping indexing problem is defined as follows: pre-process T into a data structure, such that for any pattern P, containing $|P|$ characters, we can report a set containing the maximum number of non-overlapping occurrences of P in T. Cohen and Porat (in: Algorithms and computation, 20th international symposium, ISAAC 2009, Honolulu, Hawaii. Proceedings, 2009) showed that by maintaining a linear space index in which the suffix tree of T is augmented with an $O(n)$ word data structure, a query P can be answered in optimal time $O(|P|+nocc)$, where $nocc$ is the number of occurrences reported. We present the following new result. Let CSA (not necessarily a compressed suffix array) be an index of T that can compute (i) the suffix range of P in $\mathsf{search}(P)$ time, and (ii) a suffix array or an inverse suffix array value in $\mathsf{t_{SA}}$ time. By using CSA alone, we can answer a query P in $\mathsf{search}(P)+\mathsf{sort}(nocc)+O(nocc\cdot \mathsf{t_{SA}})$ time. The function $\mathsf{sort}(k)$ denotes the time for sorting k numbers in $\{1, 2, \ldots, n\}$. In the range non-overlapping indexing problem, along with the pattern P, two integers a and b, $b \geq a$, are provided as input. The task is to report a set containing the maximum number of non-overlapping occurrences of P that lie within the range $[a, b]$. For any arbitrarily small positive constant ϵ, we present an $O(n \log^{\epsilon} n)$ word index with $O(|P|+nocc_{a,b})$ query time, where $nocc_{a,b}$ is the number of occurrences reported. Our index improves upon the result of Cohen and Porat [6].

Keywords Succinct data structures · Range queries · Suffix trees · String algorithms

✉ Arnab Ganguly
gangulya@uww.edu

Extended author information available on the last page of the article

1 Introduction and Related Work

The rapid growth of textual data and the increasing need to extract information from it has led to numerous applications of pattern matching in fields such as Bioinformatics, Data Mining, Web Mining, Computational Biology, and Information Retrieval in general. Given a text T and a pattern P, the pattern matching problem is to find all starting positions (or simply, occurrences) of sub-strings of T that match exactly with P. Earlier works [4,18,20] concentrated on the scenario in which both text and pattern were provided at query time. In most cases, however, the text is static, and patterns come in a query. This motivated the development of *full-text-indexes* so as to facilitate pattern matching efficiently. More specifically, the objective is to pre-process a text T and build a data structure, such that whenever a pattern P comes as a query, all occurrences of P in T can be reported efficiently. *Suffix tree* [9,25,26] (resp. *suffix array* [22]) are the most well known full-text-indexes supporting $O(|P| + occ)$ (resp. $O(|P| + \log n + occ)$) query time, where $|P|$ is the length of P and occ is the output size (i.e., the number of occurrences of P in T). The query time for suffix arrays can be reduced to $O(|P| + occ)$ by using a modified form of it, called *enhanced suffix arrays* [1]. Both suffix trees and suffix arrays require $\Theta(n \log n)$ bits of space, which is too large for most practical purposes. Consequently, the focus became to design indexes which occupy space close to the size of the text and achieve the full functionality of suffix trees/arrays, even if it results in a small penalty in query time. Grossi and Vitter [14], and Ferragina and Manzini [10] addressed this by presenting space efficient indexes named Compressed Suffix Array and FM-Index respectively. Subsequently, an exciting field of compressed text indexing was established. We refer the reader to [23] for an excellent survey on compressed text indexing.

Unfortunately, traditional pattern matching, on its own, does not capture many real-world applications. This led to the formulation of many variants, and the subsequent design of data structures to handle them. Among the notable, two variants are (i) pattern matching with errors/don't-cares [7], where the pattern and a sub-string of the text match if they differ by a bounded number of errors/don't cares, (ii) parameterized pattern matching [3] in which the match is defined by a one-to-one function that renames characters from the matched sub-string of the text to that of the pattern.

The first problem addressed in this paper is a variant of the pattern matching problem. It asks to report the non-overlapping occurrences of P in T. Apart from its theoretical interest, the problem finds a lot of potential applications [6] in areas related to data compression, speech recognition, and linguistics. For instance, one can compress a text by replacing each non-overlapping occurrence of P by a single character from a new alphabet, where each symbol in the new alphabet is the image obtained by applying a hash-function on P. The following is a formal definition of the problem.

Problem 1 (Non-overlapping indexing) *Given a text T of n characters, pre-process T into a data structure, such that for any input pattern P, we can report a set of starting positions of P of maximum size, where any two distinct reported positions are at least $|P|$ characters apart.*

Cohen and Porat [6] presented the first optimal time solution to this problem. Their data structure consists of a suffix tree of T, which is augmented with an additional $O(n)$-word data structure. However, it was left unanswered, whether Problem 1 can be handled in succinct (or, compact) space, or not. We answer this affirmatively by showing that the problem can be solved efficiently using any index of T alone, as summarized in the following theorem.[1]

Theorem 1 *Let* CSA *(not necessarily a compressed suffix array) be a full-text-index of T, that can compute (i) the suffix range of a pattern P in* $\mathsf{search}(P)$ *time and (ii) suffix array or an inverse suffix value in* $\mathsf{t_{SA}}$ *time. By using* CSA *alone, we can answer a query P in* $\mathsf{search}(P) + \mathsf{sort}(nocc) + O(nocc \cdot \mathsf{t_{SA}})$ *time, where nocc is the number of occurrences reported. The function* $\mathsf{sort}(k)$ *denotes the time for sorting k numbers in* $\{1, 2, \ldots, n\}$.

Henceforth, σ is the size of the alphabet from which the characters in T are drawn and $\epsilon > 0$ is an arbitrarily small constant.

Yet another variant of the traditional pattern matching problem is the well known *position restricted substring searching* problem of Mäkinen and Navarro [21], where the only occurrences to be reported are those which lie within a specified range of T. The second problem addressed in this paper can be seen as a variation of this problem and can be stated as follows.

Problem 2 (Range non-overlapping indexing) *Given a text T of n characters, pre-process T into a data structure, such that whenever a pattern P, and a range $[a, b]$, $1 \leq a \leq b \leq n$, are provided as input, we can report a set of maximum size containing the starting positions of P in the range $[a, b]$ where any two distinct reported positions are at least $|P|$ characters apart.*

For Problem 2, Keller et al. [19] presented an $O(n \log n)$ space and $O(|P| + nocc_{a,b} \log \log n)$ time data structure, where $nocc_{a,b}$ is the number of (non-overlapping) occurrences reported. Crochemore et al. [8] presented an $O(n^{1+\epsilon})$ space and $O(|P| + nocc_{a,b})$ time solution. Nekrich and Navarro [24] presented a linear space and $O(|P| + nocc_{a,b} \log^{\epsilon} n)$ time solution. Cohen and Porat [6] improved upon the solution of Keller et al. by presenting an $O(n \log^{\epsilon} n)$ space and $O(|P| + \log \log n + nocc_{a,b})$ time index. The following theorem summarizes our solution to Problem 2.

Theorem 2 *We can pre-process T to create data structures of total size $O(n \log^{\epsilon} n)$ words, such that a query $\langle P, a, b \rangle$ can be answered in optimal $O(|P| + nocc_{a,b})$ time, where $nocc_{a,b}$ is the number of occurrences reported.*

Organization of the paper The rest of the paper is dedicated to proving Theorems 1 and 2. In Sect. 2, we introduce notations and terminologies. We prove Theorem 1 in Sect. 3. In Sect. 4, we prove Theorem 2. Finally, we conclude the paper in Sect. 5.

[1] A preliminary version [11] of this paper appeared in the 26th Annual Symposium on Combinatorial Pattern Matching (CPM) 2015.

2 Preliminaries and Notations

We refer the reader to [15] for standard definitions and terminologies. Throughout this paper, T is a text having n characters, and P is a pattern having $|P|$ characters. We assume the standard Word-RAM model of computation where the word size $\omega \geq \log n$. Also, assume that T terminates in a special character \$ that does not appear at any other position in the document. Denote by $T[t, t']$ the substring of T from t to t' (both inclusive), and by ϵ an arbitrarily small positive constant. A pattern P is said to occur at a position t in T if P starts at t. Let $nocc$ denote the size of a set containing the maximum number of non-overlapping occurrences of P in T. Likewise, $nocc_{a,b}$ is the size of a set containing the maximum number of non-overlapping occurrences in the range $[a, b]$ i.e., in the sub-string $T[a, b]$.

2.1 Suffix Tree and Suffix Array

A suffix tree, denoted by ST, is a compact trie that stores all the suffixes of T. The leaves in a suffix tree are numbered in the lexicographic order of the suffix they represent. The locus of a pattern P is the highest node u such that P is a prefix of the string formed by the concatenation of the edge labels from the root to u. The suffix range of P is denoted by $[sp, ep]$, where sp (resp. ep) is the leftmost (resp. rightmost) leaf in the subtree of ST rooted at the locus of P. Using a suffix tree, the locus node (or equivalently, the suffix range) of any pattern P can be computed in time $O(|P|)$.

A suffix array, denoted by SA, is an array of size n that maintains the lexicographic arrangement of all the suffixes of the text. More specifically, if the ith smallest suffix of T starts at j, then $\mathsf{SA}[i] = j$ and $\mathsf{SA}^{-1}[j] = i$. Using suffix arrays, the locus node of any pattern P (or equivalently, the suffix range of P), can be found in $O(|P| + \log n)$ time. Using enhanced suffix arrays [1], this can be done in $O(|P|)$ time. Moving forward, we use the term suffix array to denote enhanced suffix arrays. The suffix value $\mathsf{SA}[\cdot]$ and the inverse suffix value $\mathsf{SA}^{-1}[\cdot]$ can be found in constant time.

In general, suffix trees (arrays) require $O(n)$ words for storage. *Compressed Suffix Arrays* reduce this space to $O(n \log \sigma)$ bits (or close to the size of the text) with a slowdown in query time.

In what follows, we use CSA to denote a full-text-index of T (not necessarily a compressed index) that can compute the suffix range of P in $\mathsf{search}(P) = \Omega(|P|)$ time, and can compute a suffix array or inverse suffix array value in $\mathsf{t_{SA}} = \Omega(1)$ time. We assume that $\mathsf{search}(P)$ is proportional to the length of the pattern P.

Lemma 1 *Given the suffix range $[sp, ep]$ of pattern P, using CSA, we can verify in time $\mathsf{t_{SA}}$ whether P appears at a text-position t, or not.*

Proof The lexicographic position ℓ of the suffix $T[t, n]$ (i.e., $\mathsf{SA}^{-1}[t]$) can be found in $\mathsf{t_{SA}}$ time. The lemma follows by observing that P occurs at t iff ℓ lies in the range $[sp, ep]$. □

2.2 Periodicity

Definition 1 The period of a pattern P is its shortest non-empty prefix Q, such that P can be written as the concatenation of several (say $\alpha > 0$) number of copies of Q and a (possibly empty) prefix Q' of Q. Specifically, $P = Q^\alpha Q'$. The period of P can be computed in $O(|P|)$ time using the failure function of the KMP algorithm [15,20].

For example, if $P = abcabcab$, then $Q = abc$, $\alpha = 2$, and $Q' = ab$. If $P = aaa$, then $Q = a$, $\alpha = 3$, and Q' is empty. If $P = abc$, then $Q = abc$, $\alpha = 1$, and Q' is empty.

Definition 2 We say that the pattern $P = Q^\alpha Q'$ is periodic if $\alpha \geq 2$, else P is aperiodic.

2.3 Simplifying Assumptions

Observe that if P does not occur in T, both Problems 1 and 2 can be trivially answered using any full-text-index of T. This condition can be verified in $\mathsf{search}(P)$ time using a full-text-index of T. Also, observe that P can overlap itself iff there is a proper suffix of P which is also its (proper) prefix; in this case, Q is a proper prefix of $P = Q^\alpha Q'$. If this condition does not hold, which can be verified in $O(|P|)$ time using the KMP algorithm [15,20], we can find the desired occurrences using traditional pattern matching. So, moving forward, we assume that P appears in T and that there is a proper suffix of P which is also its (proper) prefix.

3 Non-overlapping indexing

In this section, we present our solution (in Theorem 1) to Problem 1. Based on whether the input pattern is periodic or aperiodic, we have the following two cases.

3.1 Case 1: Input Pattern $P = Q^\alpha Q'$ is Aperiodic ($\alpha < 2$)

Note that the desired non-overlapping occurrences can be found as follows: (i) first report all occ occurrences in the sorted order, (ii) report the last occurrence, and then perform a right to left scan of the occurrences, and report an occurrence if it is at least $|P|$ characters away from the previously reported occurrence. The complexity of this procedure is $O(\mathsf{search}(P) + occ \cdot \mathsf{t_{SA}} + \mathsf{sort}(occ))$. We observe that $occ \leq 2 \cdot nocc$, when the pattern is aperiodic. Therefore, the above time complexity is in line with Theorem 1.

3.2 Case 2: Input Pattern $P = Q^\alpha Q'$ is Periodic ($\alpha \geq 2$)

Definition 3 (*Critical occurrence of a periodic pattern)* A position t in the text T is called a critical occurrence of a periodic pattern $P = Q^\alpha Q'$ iff t is an occurrence of P but the position $t + |Q|$ is not. Here Q is the period of P.

Definition 4 (*Span of a critical occurrence*) Let t_c be a critical occurrence of the pattern P in the text T. Let $t' \leq t_c$ be the maximal position such that $t', t' + |Q|, t' + 2|Q|, \ldots, t_c$ are occurrences of P but the position $t' - |Q|$ is not. The span of t_c is $\mathsf{span}(t_c) = [t', t + |P| - 1]$.

For example, let the text $T[1, 18]$ be $XYZabcabcabcabXYZ\$$. Then $t = 7$ is a critical occurrence of $P = abcabcab$, but $t = 4$ is not. Also, $\mathsf{span}(7) = [4, 14]$. Following are some crucial observations.

Observation 1 *Let t be a critical occurrence of P. Then, t is the rightmost occurrence of P in $\mathsf{span}(t)$. Also, the spans of two critical occurrences cannot overlap by more than $|Q| - 1$ characters.*

Observation 2 *Let t_{i-1}, t_i, and t_{i+1} be 3 consecutive critical occurrences, where $t_{i-1} < t_i < t_{i+1}$. Then, $\mathsf{span}(t_{i-1})$ and $\mathsf{span}(t_{i+1})$ do not overlap. Thus, the number of critical occurrences of P is at most $2 \cdot nocc$.*

It follows from Observations 1 and 2 that in order to find the desired non-overlapping positions of P in the text T, it suffices to find the non-overlapping occurrences of P in the range of every critical occurrence of P, and ensure that such a reported occurrence does not overlap with an occurrence in the span of an adjoining critical occurrence.

At this point, we assume that we have located all the critical occurrences of P. We sort them in $\mathsf{sort}(nocc)$ time (based on Observation 2). Let $\{t_1, t_2, \ldots, t_k\}$ be the critical occurrences in sorted order. Essentially, to report all the non-overlapping occurrences of P, we report the rightmost occurrence (which is the critical occurrence t_k) and then scan from right to left skipping over the occurrences that overlap with the last reported occurrence within the span of the current critical occurrence. Once we cross the span, we repeat the process in the span of the previous critical occurrence. More specifically, we use the following algorithm:

1. Let $t_0 = 0$, $c = k$, and $t = t_k$.
2. Report t as a non-overlapping occurrence of P. (Conceptually, t will keep track of the last reported occurrence.)
3. Let $t' = t - (\alpha + 1)|Q|$. If $t' > t_{c-1}$ and if P appears at t' (verified using Lemma 1), then t' is the rightmost occurrence in $\mathsf{span}(t_c)$ that does not overlap with t; we set $t = t'$, and repeat from Step 2. Otherwise, goto Step 4.
4. If $c = 1$ then stop, else, $c = c - 1$. If $t - t_c < |P|$, then $\mathsf{span}(t_c)$ contains t, and we set $t = t_c - |Q|$, else we set $t = t_c$. Repeat from Step 2.

It is easy to see that the above algorithm will output a set of the maximum number of non-overlapping occurrences of P. Note that once the suffix range of P is known, reporting each position (by checking whether it is an occurrence of P or not) takes $\mathsf{t_{SA}}$ time using Lemma 1. Summarizing, we have the following lemma.

Lemma 2 *Given a periodic pattern $P = Q^\alpha Q'$, the suffix range of P, and the critical occurrences of P, we can find a set of the maximum number of non-overlapping occurrences of P in time $\mathsf{sort}(nocc) + O(nocc \cdot \mathsf{t_{SA}})$.*

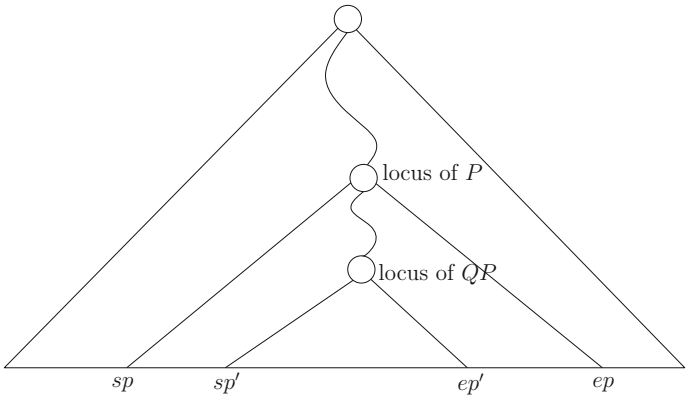

Fig. 1 Illustration of Lemma 4. Since Q' is a prefix of Q, the locus of $P = Q^{\alpha}Q'$ lies on the path from the root to the locus of QP. For each leaf ℓ in $[sp, sp' - 1] \cup [ep' + 1, ep]$, the text position $\mathsf{SA}[\ell]$ is a critical occurrence of P in T

Lemma 3 *A critical occurrence of a periodic pattern P is the same as the text position of a leaf which belongs to the suffix range of P, but not of QP.*

Proof We prove the lemma by contradiction. Observe that since Q' is a prefix of Q, the suffix range of QP is contained within that of P. Say there exists a critical occurrence t_c in text order, such that $\mathsf{SA}^{-1}[t_c]$ lies in the suffix range of $QP = Q^{\alpha+1}Q'$. Clearly, there is an occurrence of P at the position $t = t_c + |Q|$, a contradiction. □

Our task is now to find all critical occurrences. The following lemma shows how to achieve this.

Lemma 4 *Given a periodic pattern $P = Q^{\alpha}Q'$, we can find all critical occurrences of P in T in time bounded by $O(\mathsf{search}(P) + nocc \cdot \mathsf{t_{SA}})$.*

Proof Observe that since Q' is a prefix of Q, the suffix range of QP is contained within that of P. Applying Lemma 4, our objective translates to locating the suffix range of P, say $[sp, ep]$, and of QP, say $[sp', ep']$. This can be achieved in time $\mathsf{search}(QP)$, which can be bounded by $O(\mathsf{search}(P))$. We assume that $\mathsf{search}(P)$ is proportional to $|P|$. See Fig. 1 for an illustration.

Note that for each leaf ℓ lying in $[sp, sp' - 1] \cup [ep' + 1, ep]$, the text position $\mathsf{SA}[\ell]$ is a (distinct) critical occurrence of P. Thus, the total number of these leaves is the same as the number of critical occurrences of P in T. By Observation 2, the number of critical occurrences is at most $2 \cdot nocc$. For every leaf, we can find the corresponding critical occurrence (i.e., its text position) in time $\mathsf{t_{SA}}$ using $\mathsf{SA}[\cdot]$. Therefore, once the suffix ranges of P and QP are located, all the critical occurrences are found in time $O(nocc \cdot \mathsf{t_{SA}})$. □

This completes the proof of Theorem 1. By avoiding the use of any additional data structures, we ensure that various space and time trade-offs can be easily obtained. Also notice that an optimal-time linear-space solution, much simpler than that of Cohen and Porat [6], can also be obtained as follows: maintain a linear-space sorted range

reporting structure [5] over the suffix array. With this, $\mathsf{sort}(\cdot)$ tasks in our algorithm can be implemented in linear time. Essentially, for the aperiodic case we simply obtain all the occurrences in sorted order using this data structure in time proportional to the number of occurrences. For the periodic case, we obtain the critical occurrences in sorted order using this data structure in time proportional to the number of critical occurrences. After that we follow the same querying process as described for the two cases previously.

4 Range Non-overlapping Indexing

In the range non-overlapping indexing problem, a range $[a, b]$ is provided as input in addition to the pattern P, and we are required to report a set containing the maximum number of non-overlapping occurrences of P that start within the range $[a, b]$. For this problem, we use suffix trees or (enhanced) suffix arrays as the full-text-indexes for T. Therefore, $\mathsf{search}(P) = \Theta(|P|)$ and $\mathsf{t_{SA}} = \Theta(1)$.

Based on the length of the pattern P, we divide the proof of Theorem 2 into two cases. We say that P is *long* if $|P| > \log\log n$, and *short*, otherwise. Cohen and Porat [6] showed that one can answer Problem 2 using $O(n \log^{\epsilon} n)$ words, in $O(|P| + \log\log n + nocc)$ time. Thus, for long patterns, the bounds in Theorem 2 follow directly from this result. We prove the following lemma in the remainder of this section, which leads to Theorem 2.

Lemma 5 *There exists a data structure that takes $O(n \log\log n)$ words of space, such that given a range $[a, b]$, we can report a set of the maximum number non-overlapping occurrences of P in this range in time $O(|P| + nocc_{a,b})$, where $nocc_{a,b}$ is the output size and $|P| \leq \log\log n$.*

4.1 Proof of Lemma 5

It is to be observed that the index in [6] will report non-overlapping occurrences in $[a, b]$ even for short patterns i.e., when $|P| \leq \log\log n$. In this case, however, the query complexity will incur an extra $\log\log n$ factor (because $|P|$ does not necessarily cascade $\log\log n$). The motivation, therefore, is to find them in time bounded by $O(nocc_{a,b})$, once the suffix range of P is found.

To develop the intuition, we first note that we can find all the occurrences in the following way. First locate the first occurrence of P in the range $[a, b]$, and report it. Now, repeat the following process until we reach an occurrence outside the range $[a, b]$: jump to next occurrence (from the last reported one) in text-order that is at least $|P|$ characters away, and report it. Clearly, the query time is bounded by $O(nocc_{a,b})$ as long as given the suffix range of P, we can find the first occurrence in $O(nocc_{a,b})$ time, and given an occurrence, jump to the next non-overlapping one in $O(1)$ time. We show how to achieve this using an $O(n \log\log n)$-word data structure for patterns having length at most $\log\log n$, thereby completing the proof of Lemma 5.

Let d be a parameter to be defined later. Consider a string P_d of length d formed by concatenating edge-labels on any path from *root*. We store every occurrence of P_d in T in the data structure of the following lemma. Call this data structure A_{P_d}.

Lemma 6 [2] *Given a set $\mathcal{P}$ of m integers from $\{0, 1, \cdots, 2^{\omega} - 1\}$, where $\omega \geq \log m$ is the word size, there exists an $O(m)$-word data structure, such that given a range $R = [x_1, x_2]$, we can report an integer (if exists) of $\mathcal{P}$ that lies in R in $O(1)$ time.*

Additionally for P_d, we store another data structure B_{P_d} that contains all occurrences of P_d in sorted order. Also, for each occurrence we store the following pointers:

- Type I: a pointer to the previous occurrence of P_d that is at least d characters away. If the previous occurrence is not contained within B_{P_d}, then we point to the first occurrence in B_{P_d}. For the first occurrence in B_{P_d}, we point to itself.
- Type II: a pointer to the next occurrence of P_d that is at least d characters away.
- Type III: a pointer to the next (not necessarily non-overlapping) occurrence of P_d.

We augment the suffix tree by storing at each node u the number of leaves that lie to the left of the subtree of u. Finally, we maintain two pointers corresponding to d from the locus of P_d to the corresponding stored data structures A_{P_d} and B_{P_d}.

Denote by $\mathsf{leaf}(u)$ the set of leaves in the subtree of ST rooted at u, and by $\mathcal{U}_k$ the set of all nodes at depth k. Since the number of leaves in ST is n, it follows that $\sum_{u \in \mathcal{U}_k} |\mathsf{leaf}(u)| \leq n$. Therefore, the total space required for storing the data structure of Lemma 6 for every d length string P_d is $O(n)$ words. We maintain this data structure for every $d \in [1, \lceil \log \log n \rceil]$. The total space is bounded by $O(n \log \log n)$ words.

To answer queries for a short pattern P, we first locate the structure A_P from the locus node of P. Using this structure, we first find an occurrence of P within the range $[a, b]$ in $O(1)$ time. Now, using this occurrence, say t, the inverse suffix array, and the number of leaves lying to the left of the subtree of the locus node, we locate the position corresponding to t in B_P. Now, we use Type I pointers in B_P to repeatedly jump to the previous non-overlapping occurrence of P, until:

- We cross the boundary a and arrive at an occurrence, say t'. Starting from t', we successively follow Type III pointers to the next occurrence until we arrive at the first occurrence, say t_f, after a. Note that we will need to follow at most $|Q| - 1$ such pointers (due to Observation 1).
- We arrive at the first occurrence, say t_f, of P_d in A_P.

Now, we report a set of the maximum non-overlapping occurrences of P within $[a, b]$ by simply following Type II pointers starting from t_f until we cross b. Since each pointer jump requires $O(1)$ time, the total time in addition to locating the locus node of P is $O(nocc_{a,b})$. This completes the proof of Lemma 5.

5 Conclusion

In this paper, we revisit the problem of reporting a set containing the maximum number of non-overlapping occurrences of a pattern P in a text T. We show that by maintaining only a full-text-index on T, we can find all $nocc$ occurrences in time $\mathsf{search}(P) + \mathsf{sort}(nocc) + O(nocc \cdot \mathsf{t_{SA}})$, where $\mathsf{search}(P)$ is the time required to find the suffix range of T, and $\mathsf{t_{SA}}$ is the time required to find suffix value or inverse suffix value. Very recently Hooshmand et al. [17] presented an I/O optimal data structure in the cache-oblivious model with space $O(n \log n)$ words. An interesting open question

is, can we design I/O optimal data structure in the cache-aware model with space $o(n \log n)$ words. An interesting problem is to design an efficient data structure for answering the counting queries: i.e. to compute the value $nocc$ in time proportional to $|P|$. It is also interesting to see whether non-overlapping (succinct) indexes could be designed for some the variants of the pattern matching problem, such as parameterized matching [3,12,13].

For the range-reporting version of the problem, where a range $[a, b]$ is provided as input in addition to the pattern P, we present an $O(n \log^{\epsilon} n)$ space index which can report all $nocc_{a,b}$ occurrences in this range in optimal time $O(|P| + nocc_{a,b})$. We remark that it is highly unlikely to have an efficient succinct data structure for this problem, based on the hardness result of the position restricted substring searching problem [16].

Acknowledgements This research is funded in part by National Science Foundation (NSF) Grant CCF 1218904.

References

1. Abouelhoda, M.I., Kurtz, S., Ohlebusch, E.: Replacing suffix trees with enhanced suffix arrays. J. Discrete Algorithms **2**(1), 53–86 (2004)
2. Alstrup, S., Brodal, G.S., Rauhe, T.: Optimal static range reporting in one dimension. In: Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6–8, 2001, Heraklion, Crete, Greece, pp. 476–482 (2001)
3. Baker, B.S.: A theory of parameterized pattern matching: algorithms and applications. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16–18, 1993, San Diego, CA, USA, pp. 71–80 (1993)
4. Boyer, R.S., Moore, J.S.: A fast string searching algorithm. Commun. ACM **20**(10), 762–772 (1977)
5. Brodal, G.S., Fagerberg, R., Greve, M., López-Ortiz, A.: Online sorted range reporting. In: Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16–18, 2009. Proceedings, pp. 173–182 (2009)
6. Cohen, H., Porat, E.: Range non-overlapping indexing. In: Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings, pp. 1044–1053 (2009)
7. Cole, R., Gottlieb, L.-A., Lewenstein, M.: Dictionary matching and indexing with errors and don't cares. In: Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13–16, 2004, pp. 91–100 (2004)
8. Crochemore, M., Iliopoulos, C.S., Kubica, M., Rahman, M.S., Walen, T.: Improved algorithms for the range next value problem and applications. In: STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21–23, 2008, Proceedings, pp. 205–216 (2008)
9. Farach, M.: Optimal suffix tree construction with large alphabets. In: 38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19–22, 1997, pp. 137–143 (1997)
10. Ferragina, P., Manzini, G.: Indexing compressed text. J. ACM **52**(4), 552–581 (2005)
11. Ganguly, A., Shah, R., Thankachan, S.V.: Succinct non-overlapping indexing. In: Combinatorial Pattern Matching—26th Annual Symposium, CPM 2015, Ischia Island, Italy, June 29–July 1, 2015, Proceedings, pp. 185–195 (2015)
12. Ganguly, A., Shah, R., Thankachan, S.V: pBWT: achieving succinct data structures for parameterized pattern matching and related problems. In: Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 397–407. Society for Industrial and Applied Mathematics (2017)

13. Ganguly, A., Shah, R., Thankachan, S.V.: Structural pattern matching-succinctly. In: 28th International Symposium on Algorithms and Computation, ISAAC 2017, December 9–12, 2017, Phuket, Thailand, pp. 35:1–35:13 (2017)
14. Grossi, R., Vitter, J.S.: Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract). In: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21–23, 2000, Portland, OR, USA, pp. 397–406 (2000)
15. Gusfield, D.: Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, Cambridge (1997)
16. Hon, W.-K., Shah, R., Thankachan, S.V., Vitter, J.S.: On position restricted substring searching in succinct space. J. Discrete Algorithms **17**, 109–114 (2012)
17. Hooshmand, S., Abedin, P., Külekci, M.O., Thankachan, S.V.: Non-overlapping indexing: cache obliviously. In: Annual Symposium on Combinatorial Pattern Matching, CPM 2018, July 2–4, 2018 - Qingdao, China, pp. 8:1–8:9 (2018)
18. Karp, R.M., Rabin, M.O.: Efficient randomized pattern-matching algorithms. IBM J. Res. Dev. **31**(2), 249–260 (1987)
19. Keller, O., Kopelowitz, T., Lewenstein, M.: Range non-overlapping indexing and successive list indexing. In: Algorithms and Data Structures, 10th International Workshop, WADS 2007, Halifax, Canada, August 15–17, 2007, Proceedings, pp. 625–636 (2007)
20. Knuth, D.E., Morris Jr., J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM J. Comput. **6**(2), 323–350 (1977)
21. Mäkinen, V., Navarro, G.: Position-restricted substring searching. In: LATIN 2006: Theoretical Informatics, 7th Latin American Symposium, Valdivia, Chile, March 20–24, 2006, Proceedings, pp. 703–714 (2006)
22. Manber, U., Myers, E.W.: Suffix arrays: a new method for on-line string searches. SIAM J. Comput. **22**(5), 935–948 (1993)
23. Navarro, G., Mäkinen, V.: Compressed full-text indexes. ACM Comput. Surv. **39**, 1 (2007)
24. Nekrich, Y., Navarro, G.: Sorted range reporting. In: Algorithm Theory—SWAT 2012: 13th Scandinavian Symposium and Workshops, Helsinki, Finland, July 4–6, 2012. Proceedings, pp. 271–282 (2012)
25. Ukkonen, E.: On-line construction of suffix trees. Algorithmica **14**(3), 249–260 (1995)
26. Weiner, P.: Linear pattern matching algorithms. In: 14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15–17, 1973, pp. 1–11 (1973)

Affiliations

Arnab Ganguly[1] · Rahul Shah[2] · Sharma V. Thankachan[3]

Rahul Shah
rahul@csc.lsu.edu

Sharma V. Thankachan
sharma.thankachan@ucf.edu

1 Department of Computer Science, University of Wisconsin - Whitewater, Whitewater, USA

2 Department of Computer Science, Louisiana State University, Baton Rouge, USA

3 Department of Computer Science, University of Central Florida, Orlando, USA