On Hardness of Several String Indexing Problems

Kasper Green Larsen[1], J Ian Munro[2],
Jesper Sindahl Nielsen [1], and Sharma V. Thankachan[2]

[⋆] MADALGO, [1]Aarhus University, Denmark {larsen,jasn}@cs.au.dk
[2] University of Waterloo, Canada {imunro,thanks}@uwaterloo.ca

Abstract. Let $\mathcal{D} = \{d_1, d_2, ..., d_D\}$ be a collection of D string documents of n characters in total. The two-pattern matching problems ask to index $\mathcal{D}$ for answering the following queries efficiently.

- report/count the unique documents containing P_1 *and* P_2.
- report/count the unique documents containing P_1*, but not* P_2.

Here P_1 and P_2 represent input patterns of length p_1 and p_2 respectively. Linear space data structures with $O(p_1 + p_2 + \sqrt{nk} \log^{O(1)} n)$ query cost are already known for the reporting version, where k represents the output size. For the counting version (i.e., report the value k), a simple linear-space index with $O(p_1 + p_2 + \sqrt{n})$ query cost can be constructed in $O(n^{3/2})$ time. However, it is still not known if these are the best possible bounds for these problems. In this paper, we show a strong connection between these string indexing problems and the boolean matrix multiplication problem. Based on this, we argue that these results cannot be improved significantly using purely combinatorial techniques. We also provide an improved upper bound for a related problem known as *two-dimensional substring indexing*.

1 Introduction

Document listing is a fundamental problem in information retrieval, where the task is to index a collection of documents, such that whenever a pattern P comes as a query, we can efficiently find the unique documents containing P as a substring. This problem was introduced by Matias et al. and they provide a linear space and near-optimal time solution [15]. Later Muthukrishnan improved the result by providing a linear-space and optimal query time index [16]. The counting case asks to find the number of documents containing the query pattern. See [17] for an excellent survey on more results and extensions of document retrieval problems. In this paper, our focus is on the case where the query consists of two patterns (known as *two-pattern query problems*). The formal definitions of the problems under consideration are given below.

[⋆] Work supported in part by the Danish National Research Foundation grant DNRF84 through Center for Massive Data Algorithmics (MADALGO)

Problem 1 Given a set of strings $\mathcal{D} = \{d_1, d_2, \dots, d_D\}$ with $\sum_{i=1}^{D} |d_i| = n$, preprocess $\mathcal{D}$ to answer queries: given two strings P_1 and P_2 report all i's where both P_1 and P_2 occur in d_i.

Problem 2 Given a set of strings $\mathcal{D} = \{d_1, d_2, \dots, d_D\}$ with $\sum_{i=1}^{D} |d_i| = n$, preprocess $\mathcal{D}$ to answer queries: given two strings P^+ and P^- report all i's where P^+ occurs in string d_i and P^- does not occur in string d_i.

Problem 3 Let $\mathcal{D} = \{(d_{1,1}, d_{1,2}), (d_{2,1}, d_{2,2}), \dots, (d_{D,1}, d_{D,2})\}$, be a set of pairs of strings with $\sum_{i=1}^{D} |d_{i,1}| + |d_{i,2}| = n$. Preprocess $\mathcal{D}$ to answer queries: given two strings P_1 and P_2 report all i's where P_1 occurs in $d_{i,1}$ and P_2 occurs in $d_{i,2}$.

Problem 1 was introduced by Muthukrishnan [16]. He presented a data structure using $O(n^{1.5} \log^{O(1)} n)$-space (in words) with $O(p_1 + p_2 + \sqrt{n} + k)$ time for query processing, where $p_1 = |P_1|$ and $p_2 = |P_2|$ and k is the output size[1]. Later Cohen and Porat [6] presented a space efficient structure of $O(n \log n)$-space, but with a higher query time of $O(p_1 + p_2 + \sqrt{nk \log n} \log^2 n)$. The space and the query time of was improved by Hon et al. [11] to $O(n)$ words and $O(p_1 + p_2 + \sqrt{nk \log n} \log n)$ time. See [13] for a succinct space solution for this problem. Problem 2 is known as the forbidden (or excluded) pattern query problem. This was introduced by Fischer et al. [8], where they presented an $O(n^{3/2})$-bit solution with query time $O(p_1 + p_2 + \sqrt{n} + k)$. Immediately, Hon et al. [12] improved its space occupancy to $O(n)$ words, but with a higher query time of $O(p_1 + p_2 + \sqrt{nk \log n} \log^2 n)$. They presented an $O(n)$-space and $O(p_1 + p_2 + \sqrt{n} \log \log n)$ query time structure for the counting version of Problem 2 (i.e., just report the value k). We remark that the same framework can be adapted to handle the counting version of Problem 1 as well. Also the $O(\log \log n)$ term in the query time can be removed by replacing predecessor search queries in their algorithm by range emptiness queries. In summary, we have $O(n)$-space and $\tilde{\Omega}(\sqrt{n})$ query time solutions for the reporting/counting versions of these problems. However, the question whether these are the best possible bounds remains unanswered.

Problem 3 is a generalization of Problem 1 known as the *two-dimensional substring indexing* problem, and was introduced by Ferragina et al. [7]. They reduced it to another problem known as the *common colors query* problem, where the task is to preprocess an array of colors and maintain a data structure, such that whenever two ranges comes as a query, we can output the unique colors which are common to both ranges. Based on their solution for this new problem, they presented an $O(n^{2-\epsilon})$ space and $O(n^{\epsilon} + k)$ query time solution for Problem 3, where ϵ is any constant in $(0, 1]$. Later Cohen and Porat [6] presented a space efficient solution for the common colors query problem of space $O(n \log n)$ words and query time $O(\sqrt{nk \log n} \log^2 n)$. Therefore, the current best data structure for *two-dimensional substring indexing* problem occupies $O(n \log n)$ space and processes a query in $O(p_1 + p_2 + \sqrt{nk \log n} \log^2 n)$ time.

[1] Specifically k is the maximum of 1 and the output size.

1.1 Our Results

In this paper, we use the Word-RAM model of computation with word size $w = \Omega(\log n)$. The following summarizes our main results.

- We present a strong connection between the counting versions of the string indexing problems (Problem 1, 2, and 3) and the boolean matrix multiplication problem. Specifically, we show that multiplying two $\sqrt{n} \times \sqrt{n}$ boolean matrices can be reduced to the problem of indexing $\mathcal{D}$ (in Problem 1, Problem 2, or Problem 3) and answering n counting queries. However, matrix multiplication is a well known hard problem and this connection gives us a hardness result for the pattern matching problems under considerations.
- We present an improved upper bound for the common colors query problem, where the space and query time are $O(n)$ and $O(\sqrt{nk}\log n)$ respectively. Therefore, we now have a linear-space and $O(p_1 + p_2 + \sqrt{nk}\log n)$ query time index for the two-dimensional substring indexing problem (Problem 3).

2 Hardness Results

The hardness results are reductions from *boolean matrix multiplication*. Through this section we use similar techniques to [3–5]. In the boolean matrix multiplication problem we are given two $n \times n$ matrices A and B with $\{0, 1\}$ entries. The task is to compute the boolean product of A and B, that is replace multiplication by logical and, and replace addition by logical or. Letting $a_{i,j}, b_{i,j}, c_{i.j}$ denote entry i, j of respectively A, B and C the task is to compute for all i, j

$$c_{i,j} = \bigvee_{k=1}^{n} (a_{i,k} \wedge b_{k,j}).$$

The asymptotically fastest algorithm known for matrix multiplication currently uses $O(n^{2.3728639})$ time [9]. This bound is achieved using algebraic techniques (like in Strassen's matrix multiplication algorithm) and the fastest combinatorial algorithm is still cubic divided by some poly-logarithmic factor [1].

In this section, we prove that the problem of multiplying two $\sqrt{n} \times \sqrt{n}$ boolean matrices A and B can be reduced to the problem of indexing $\mathcal{D}$ (in Problem 1 or 2) and answering n counting queries. This is evidence that unless better matrix multiplication algorithms are discovered we should not expect to be able to preprocess the data and answer the queries much faster than $\Omega((\sqrt{n})^{\omega})) = \Omega(n^{\omega/2})$ (ignoring poly-logarithmic factors) where ω is the matrix multiplication exponent. In other words one should not expect to be able to have small preprocessing and query time simultaneously. Currently we cannot achieve better than $\Omega(n^{1.18635})$ preprocessing time and $\Omega(n^{0.18635})$ query time simultaneously.

We start the next section with a brief discussion on how to view boolean matrix multiplication as solving many set intersection problems. Then we give the reductions from the matrix multiplication problem to Problem 2 and describe how to adapt it for Problem 1.

2.1 Boolean Matrix Multiplication

A different way to phrase the boolean matrix multiplication problem is that entry $c_{i,j} = 1$ if and only if $\exists k : a_{i,k} = b_{k,j} = 1$. For any two matrices A and B let $A_i = \{k \mid a_{i,k} = 1\}$ and similarly let $B_j = \{k \mid b_{k,j} = 1\}$. It follows that $c_{i,j} = 1$ if and only if $A_i \cap B_j \neq \emptyset$. In this manner we view each row of matrix A as a set containing the elements corresponding to the indices where there is a 1, and similarly for columns in B. For completeness we also use $\overline{A_i} = \{k \mid a_{i,k} = 0\}$ and $\overline{B_i} = \{k \mid b_{k,j} = 0\}$.

A naive approach to solving Problem 1 would be to index the documents such that we can find all documents containing a query pattern fast. This way a query would be to find all the documents that P_1 occurs in and the documents that P_2 occurs in separately and then return the intersection of the two result sets. This is obviously not a good solution in the worst case, but it illustrates that the underlying challenge is to solve set intersection.

We observed that boolean matrix multiplication essentially solves set intersection between rows of A and columns of B, so the idea for the reductions is to use the fact that queries for Problems 1 and 2 essentially also solve set intersection. We now give the reductions.

2.2 The Reductions

We first relax the data structure problems. Instead of returning a list of documents satisfying the criteria we just want to know whether the list is empty or not, i.e. return 0 if empty and 1 if nonempty.

Let A and B be two $\sqrt{n} \times \sqrt{n}$ boolean matrices and suppose we have an algorithm for building the data structure for the relaxed Problem 2. We now wish to create a set of strings $\mathcal{D}$ based on A and B, build the data structure on $\mathcal{D}$ and do n queries, one for each entry in the product of A and B. In the following we need to represent a subset of $\{0, 1, \ldots, 2\sqrt{n}\}$ as a string, which we do in the following manner.

Definition 1. *Let $X = \{x_1, x_2, \ldots, x_\ell\} \subseteq \{0, 1, \ldots, 2\sqrt{n}\}$, then we represent the set X as $\mathrm{str}(X) = \mathrm{bin}(x_1)\#\mathrm{bin}(x_2)\#\cdots\#\mathrm{bin}(x_\ell)\#$ where $\mathrm{bin}(\cdot)$ gives the binary representation of the number $\cdot$ using $\lceil \frac{1}{2}\log n + 1 \rceil$ bits.*

For the matrix A we define $\sqrt{n}$ strings: $d_1^A, d_2^A, \ldots, d_{\sqrt{n}}^A$ and similarly for B we define $d_1^B, d_2^B, \ldots, d_{\sqrt{n}}^B$. Construct $d_j^B = \mathrm{str}(\{k + \sqrt{n} \mid k \in \overline{B_j}\})$ and $d_i^A = \mathrm{str}(A_i)$. We construct the $\sqrt{n}$ strings in $\mathcal{D}$ as: $d_\ell = d_\ell^A d_\ell^B$ for $1 \leq \ell \leq \sqrt{n}$.

Lemma 1. *Each string in $\mathcal{D}$ is at most $O(\sqrt{n}\log n)$ characters long.*

Proof. There are at most $\sqrt{n}$ elements in A_ℓ and at most $\sqrt{n}$ elements in $\overline{B_\ell}$. Each element in A_ℓ and $\overline{B_\ell}$ contributes exactly one number to the string d_ℓ and one '#'. Each number uses exactly $\lceil \frac{1}{2}\log n + 1 \rceil$ characters. In total we get $\left(|A_\ell + |\overline{B_\ell}|\right)\left(\lceil \frac{1}{2}\log n + 1 \rceil + 1\right) \leq 2\sqrt{n}\lceil \frac{1}{2}\log n + 1 \rceil + 2\sqrt{n} = O(\sqrt{n}\log n)$ □

Corollary 1. *The total length of the strings in $\mathcal{D}$ is* $\displaystyle\sum_{i=1}^{\sqrt{n}} d_i = O(n \log n)$.

Proof. Follows since there are at most $\sqrt{n}$ strings in $\mathcal{D}$ and by Lemma 1 each string is at most $O(\sqrt{n}\log n)$ characters long. $\square$

We have now created the set of strings, $\mathcal{D}$, that we wish to build the data structure on. We now specify the queries and prove that using these queries we can solve the matrix multiplication problem.

Lemma 2. *The entry* $c_{i,j} = 1$ *if and only if the query* $P^+ = \mathrm{bin}(i)$, $P^- = \mathrm{bin}(\sqrt{n} + j)$ *returns* 1.

Proof. Suppose that $c_{i,j} = 1$. Then by a previous discussion there must exist a k such that $a_{i,k} = b_{k,j} = 1$. By construction the string $\mathrm{bin}(i)$ occurs as a substring in d_k^A since $a_{i,k} = 1$. If $b_{k,j} = 1$ we know that $k \in B_j$ and therefore $k \notin \overline{B_j}$ so the string $\mathrm{bin}(\sqrt{n} + j)$ is not in d_k^B. It follows that the string d_k satisfies the conditions and the query returns 1.

Suppose the query $(P^+,\ P^-)$ returns 1, then by definition there exists a $d_\ell \in \mathcal{D}$ such that $\mathrm{bin}(i)$ occurs in d_ℓ and $\mathrm{bin}(\sqrt{n} + j)$ does not occur in d_ℓ. All numbers in d_ℓ but the first are surrounded by '#' and all numbers are the same length as $|P^+|$. Furthermore any number in d_ℓ less than $\sqrt{n}$ is only there because of a 1 in column ℓ of A. In particular $a_{i,\ell} = 1$, otherwise d_ℓ would not satisfy the conditions. Additionally by construction the binary representation of any number $2\sqrt{n} \geq m > \sqrt{n}$ appears in d_ℓ if and only if $b_{\ell,m} = 0$. In particular $\mathrm{bin}(j + \sqrt{n})$ did not occur in d_ℓ, therefore $b_{\ell,j} = 1$. We now have a witness (ℓ) where $a_{i,\ell} = b_{\ell,j} = 1$, and we conclude $c_{i,j} = 1$. $\square$

We are now able to give the following theorems:

Theorem 1. *Let $P(n)$ be the preprocessing time for building the data structure for Problem 1 on a set of strings of total length n and let $Q(n)$ be the query time. In time $O(P(n\log n) + n \cdot Q(n \log n) + n \log n)$ we can compute the product of two $\sqrt{n} \times \sqrt{n}$ boolean matrices.*

Proof. Follows by the lemmas and the discussion above. $\square$

Similarly for Problem 2 we obtain:

Theorem 2. *Let $P(n)$ be the preprocessing time for building the data structure for Problem 2 on a set of strings of total length n and let $Q(n)$ be the query time. In time $O(P(n\log n) + n \cdot Q(n \log n) + n \log n)$ we can compute the product of two boolean matrices.*

Proof. In the discussion above substitute $\overline{B_j}$ with B_j, P^+ and P^- with P_1 and P_2 respectively.

As a side note, observe that if we replace the problems by their counting version (i.e. count the number of strings in $\mathcal{D}$ that satisfy the condition) then these problems solve matrix multiplication with 0/1 matrices and the regular addition and multiplication operations using the same reductions.

3 The Common Colors Query Problem

In this section, we present an improved upper bound for the common colors query problem and the main result is summarized below.

Theorem 3. *An array E of n colors can be indexed in $O(n)$-word space so that the following query can be answered in $O(\sqrt{nk}\log n)$ time: report the unique colors appearing in both $E[a...b]$ and $E[c...d]$, where a, b, c and d are input parameters and k is the output size.*

Corollary 2. *The two dimensional substring indexing problem (problem 3) can be solved in linear space and $O(p_1 + p_2 + \sqrt{nk}\log n)$ query time.*

Proof. Ferragina et al. showed that Problem 3 can be reduced to the common colors problem [7]. When combined with Theorem 3, we achieve the result. □

First we give an overview, and then present the details of the proposed data structure. Let $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, ..., \sigma_{|\Sigma|}\}$ be the set of colors appearing in E. Without loss of generality we assume $|\Sigma| \leq n$. The main structure is a binary tree Δ (not necessarily balanced) of $|\Sigma|$ nodes, where each color is associated with a unique node in Δ. Specifically, the color associated with a node u is given by $\sigma_{ino(u)}$, where $ino(u)$ is the in-order rank of u. Also we use $\Sigma(u)$ to represent the set of colors associated with the nodes in the subtree of u. Let $[q, r]$ and $[s, t]$ be two given ranges in E, then $Out_{q,r,s,t}$ denotes the set of colors present in both $[q, r]$ and $[s, t]$. We maintain auxiliary data structures for answering the following subqueries efficiently.

Subquery 1 *Given $i \in [1, |\Sigma|]$ and ranges $[q, r]$ and $[s, t]$ is $\sigma_i \in Out_{q,r,s,t}$?*

Subquery 2 *Given $u \in \Delta$ and ranges $[q, r]$ and $[s, t]$ is $\Sigma(u) \cap Out_{q,r,s,t}$ empty?*

3.1 Query Algorithm

To answer the query (i.e., find $Out_{a,b,c,d}$, where $[a, b]$ and $[c, d]$ are the input ranges), we perform a *preorder* traversal of Δ. Upon reaching a node u, we issue a Subquery 2: Is $\Sigma(u) \cap Out_{a,b,d,d}$ empty?

– If the answer is *yes* (i.e., empty), we can infer that none of the color associated with any node in the subtree of u is an output. Therefore, we skip the subtree of u and move to the next node in the preorder traversal.
– On the other hand if Subquery 2 at u returns *no*, there exists at least one node v in the subtree of u, where $\sigma_{ino(v)}$ is an output. Notice that v can be the node u itself. Therefore, we first check if $\sigma_{ino(u)} \in Out_{a,b,c,d}$ using Subquery 1. If the query returns *yes*, we report $\sigma_{ino(u)}$ as an output and continue the preorder traversal.

By the end of this procedure, all colors in $Out_{a,b,c,d}$ has been reported.

3.2 Details of the Data Structure

We now present the details. For any node u in Δ, we use n_u to represent the number of elements in E with colors in $\Sigma(u)$. i.e., $n_u = |\{i|E[i] \in \Sigma(u)\}|$. Then, we construct Δ as follows, maintaining the invariant:

$$n_u \leq n \left(\frac{1}{2}\right)^{depth(u)}$$

Here $depth(u) \leq \log n$ is the number of ancestors of u. (We remark that this property is essential to achieve the result in Lemma 4 stated below). The following recursive algorithm can be used for constructing Δ. Let f_i be the number of occurrences of σ_i in E. Initialize u as the root node and $\Sigma(u) = \Sigma$. Then, find the color $\sigma_z \in \Sigma(u)$, where

$$\sum_{i<z, \sigma_i \in \Sigma(u)} f_i \leq \frac{1}{2} \sum_{\sigma_i \in \Sigma(u)} f_i \quad \text{and} \quad \sum_{i>z, \sigma_i \in \Sigma(u)} f_i \leq \frac{1}{2} \sum_{\sigma_i \in \Sigma(u)} f_i$$

Partition $\Sigma(u)$ into three disjoint subsets $\Sigma(u_L), \Sigma(u_R)$ and $\{\sigma_z\}$, where

$$\Sigma(u_L) = \{\sigma_i | i < z, \sigma_i \in \Sigma(u)\}$$

$$\Sigma(u_R) = \{\sigma_i | i > z, \sigma_i \in \Sigma(u)\}$$

If $\Sigma(u_L)$ is not empty, then we add a left child u_L for u and recurse further from u_L. Similarly, if $\Sigma(u_R)$ is not empty, we add a right child u_R for u and recurse on u_R. This completes the construction of Δ. Since Δ is a tree of $O(|\Sigma|) = O(n)$ nodes, it occupies $O(n)$ words. The following lemmas summarize the results on the structures for handling Subquery 1 and Subquery 2.

Lemma 3. *Subquery 1 can be answered in $O(\log \log n)$ time using an $O(n)$-word structure.*

Proof. The array E can be stored using $n \log |\Sigma|(1 + o(1))$-bits (or $O(n)$-word of space), and supporting $rank_E(j, \sigma_i)$: find the number of occurrences of σ_i in $E[1...j]$ in $O(\log \log |\Sigma|) = O(\log \log n)$ time for any color $\sigma_i \in \Sigma$ [10]. The answer to Subquery 1 is *yes* if both $(rank_E(r, \sigma_i) - rank_E(q-1, \sigma_i))$ and $(rank_E(t, \sigma_i) - rank_E(s-1, \sigma_i))$ are nonzero and *no* otherwise. $\square$

Lemma 4. *There exists an $O(n)$-word structure for handling Subquery 2 in the following manner:*

- *If $\Sigma(u) \cap Out_{q,r,s,t} = \emptyset$, then return* yes *in time $O\left(\frac{\log n}{\log \log n} + \sqrt{n_u} \log^{\epsilon} n\right)$*
- *Otherwise, one of the following will happen*
  - *Return* no *in $O\left(\frac{\log n}{\log \log n}\right)$ time*
  - *Return the set $\Sigma(u) \cap Out_{q,r,s,t}$ in $O\left(\frac{\log n}{\log \log n} + \sqrt{n_u} \log^{\epsilon} n\right)$ time*

Proof. See Section 3.4.

By putting all the pieces together, the total space becomes $O(n)$ words.

3.3 Analysis of Query Algorithm

The structures described in Lemma 3 and Lemma 4 can be used as black boxes to support our query algorithm. However, we slightly optimize the algorithm as follows: when we issue Subquery 2 at a node u, and the structure in Lemma 4 returns the set $\Sigma(u) \cap Out_{a,b,c,d}$ (this includes the case where Subquery 2 returns *yes*), we do not recurse further in the subtree of u. Next we bound the total time for all Subqueries.

Let $k = |Out_{a,b,c,d}|$ be the output size and Δ' be the subtree of Δ consisting only of those nodes which we visited processing the query. Then we can bound the size of Δ':

Lemma 5. *The number of nodes in Δ' is $O(k \log(n/k))$*

Proof. The parent of any node in Δ' must be a node on the path from root to some node u, where $\sigma_{ino(u)} \in Out_{a,b,c,d}$. Since the height of Δ' is at most $\log n$, the number of nodes with depth at least $\log k$ on any path is at most $\log n - \log k = \log(n/k)$. Therefore, number of nodes in Δ' with depth at least $\log k$ is $O(k \log(n/k))$. Also the total number of nodes in Δ with depth at most $\log k$ is $O(k)$. $\square$

We spend $O(\log \log n)$ time for Subquery 1 in every node in Δ'. If a node u is an internal node in Δ', then Subquery 2 in u must have returned *yes* in $O\left(\frac{\log n}{\log \log n}\right)$ time (otherwise, the algorithm does not explore its subtree). On the other hand, if a node v is a leaf node in Δ', we spend a lot more time. Thus by combining all, the overall query processing time can be bounded as follows.

$$\begin{aligned}
&O\left(k \log(n/k) \frac{\log n}{\log \log n} + \sqrt{n} \log^{\epsilon} n \sum_{v \in leaves} 2^{-depth(v)/2}\right) \\
&= O\left(k \log(n/k) \frac{\log n}{\log \log n} + \sqrt{n} \log^{\epsilon} n \sqrt{\sum_{v \in leaves} 1^2 \sum_{v \in leaves} 2^{-depth(v)}}\right) \quad (1) \\
&= O\left(k \log(n/k) \frac{\log n}{\log \log n} + \sqrt{n} \log^{\epsilon} n \sqrt{k \log(n/k) \times 1}\right) \quad (2) \\
&= O\left(\sqrt{nk} \log n\right),
\end{aligned}$$

Here Equation (1) is by Cauchy-Schwarz's inequality,[2] while Equation (2) is using Kraft's inequality: for any binary tree, $\sum_{\ell \in leaves} 2^{-depth(\ell)} \leq 1$. This completes the proof of Theorem 3.

[2] $\sum_{i=1}^{n} x_i y_i \leq \sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{i=1}^{n} y_i^2}$.

3.4 Proof of Lemma 4

The following are the building blocks of our $O(n)$-word structure.

1. For every node u in Δ, we define (but not store) E_u as an array of length n_u, where $E_u[i]$ represents the ith leftmost color in E among all colors in $\Sigma(u)$. Thus for any given range $[x, y]$, the list of colors in $E[x...y]$, which are from $\Sigma(u)$ appears in a contiguous region $[x_u, y_u]$ in E_u, where
   - $x_u = 1+$ the number of elements in $E[1...x-1]$, which are from $\Sigma(u)$.
   - $y_u =$ the number of elements in $E[1...y]$, which are from $\Sigma(u)$.

   Notice that the set of colors in $\Sigma(u)$ can be represented as contiguous range of numbers, and the task of computing $[x_u, y_u]$ for any given x, y and u can be reduced to two orthogonal range counting queries in two dimensions. We therefore maintain $O(n)$-word structure for executing this in $O(\log n/\log\log n)$ time [14].
2. An $\sqrt{n_u} \times \sqrt{n_u}$ boolean matrix M_u, for every node u in Δ. For this, we first partition E_u into blocks of size $\sqrt{n_u}$, where the ith block is given by $E_u[1 + (i-1)\sqrt{n_u}, i\sqrt{n_u}]$. Notice that the number of blocks is at most $\sqrt{n_u}$. Then, $M_u[i][j] = 1$, iff there is at least one color, which appear in both the ith block and the jth block of E_u. We also maintain a two-dimensional range maximum query structure (RMQ) with constant query time [2] over each M_u. The total space required is $O(\sum_{u\in\Delta} n_u) = O(n\sum_{u\in\Delta} 2^{-depth(u)}) = O(n \log n)$ bits.
3. Finding the leftmost/rightmost element in $E_u[x...y]$, which is from $\Sigma(u)$, for any given x, y, and u can be reduced to an orthogonal successor/ predecessor query. We therefore maintain $O(n)$-word structure for supporting this query in $O(\log^{\epsilon} n)$ time [18].

We use the following steps to answer if $\Sigma(u) \cap Out_{q,r,s,t}$ is empty.

1. Find $[q_u, r_u]$ and $[s_u, t_u]$ in $O(\log n/\log\log n)$ time.
2. Find $[q'_u, r'_u]$ and $[s'_u, t'_u]$, the ranges corresponds to the longest spans of blocks within $E_u[q_u, r_u]$ and $E_u[s_u, t_u]$ respectively. Notice that $q'_u - q_u, r_u - r'_u, s'_u - s_u, t_u - t'_u \in [0, \sqrt{n_u})$. Check if there is at least one common in both $E_u[q'_u, r'_u]$ and $E_u[s'_u, t'_u]$ with the following steps.
   - Perform an RMQ on M_u with R as the input region, where $R = [1 + (q'-1)/\sqrt{n_u}, r'/\sqrt{n_u}] \times [1 + (s'-1)/\sqrt{n_u}, t'/\sqrt{n_u}]$.
   - If the maximum value within R is 1, then we infer that there is one color common in $E_u[q'_u, r'_u]$ and $E_u[s'_u, t'_u]$. Also we can return *no* as the answer to Subquery 2.

   The time spent so far is $O(\log n/\log\log n)$.
3. If the maximum value within R in the previous step is 0, we need to do some extra work. Notice that any color, which is an output must have an an occurrence in at least one of the following spans $E_u[q_u, q'_u - 1], E_u[r'_u + 1, r_u], E_u[s_u, s'_u - 1], E_u[t'_u + 1, t_u]$ of length at most $4\sqrt{n_u}$. Therefore, these colors can be retrieved using $O(\sqrt{n_u})$ successive orthogonal predecessor/successor queries and with Subquery 1, we can verify if a candidate belongs to the output. The total time required is $O(\sqrt{n_u}\log^{\epsilon} n)$.

This completes the proof of Lemma 4.

References

1. N. Bansal and R. Williams. Regularity lemmas and combinatorial algorithms. *Theory of Computing*, 8(1):69–94, 2012.
2. G. S. Brodal, P. Davoodi, and S. S. Rao. On space efficient two dimensional range minimum data structures. *Algorithmica*, 63(4):815–830, 2012.
3. T. M. Chan, S. Durocher, K. G. Larsen, J. Morrison, and B. T. Wilkinson. Linear-space data structures for range mode query in arrays. In *STACS*, volume 14 of *LIPIcs*, pages 290–301. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
4. T. M. Chan, S. Durocher, M. Skala, and B. T. Wilkinson. Linear-space data structures for range minority query in arrays. In *SWAT*, pages 295–306, 2012.
5. T. M. Chan, K. G. Larsen, and M. Patrascu. Orthogonal range searching on the ram, revisited. In *Symposium on Computational Geometry*, pages 1–10. ACM, 2011.
6. H. Cohen and E. Porat. Fast set intersection and two-patterns matching. *Theor. Comput. Sci.*, 411(40-42):3795–3800, 2010.
7. P. Ferragina, N. Koudas, S. Muthukrishnan, and D. Srivastava. Two-dimensional substring indexing. *J. Comput. Syst. Sci.*, 66(4):763–774, 2003.
8. J. Fischer, T. Gagie, T. Kopelowitz, M. Lewenstein, V. Mäkinen, L. Salmela, and N. Välimäki. Forbidden patterns. In *LATIN*, volume 7256 of *Lecture Notes in Computer Science*, pages 327–337. Springer, 2012.
9. F. L. Gall. Powers of tensors and fast matrix multiplication. *CoRR*, abs/1401.7714, 2014.
10. A. Golynski, J. I. Munro, and S. S. Rao. Rank/select operations on large alphabets: a tool for text indexing. In *SODA*, pages 368–373. ACM Press, 2006.
11. W.-K. Hon, R. Shah, S. V. Thankachan, and J. S. Vitter. String retrieval for multi-pattern queries. In *SPIRE*, volume 6393 of *Lecture Notes in Computer Science*, pages 55–66. Springer, 2010.
12. W.-K. Hon, R. Shah, S. V. Thankachan, and J. S. Vitter. Document listing for queries with excluded pattern. In *CPM*, volume 7354 of *Lecture Notes in Computer Science*, pages 185–195. Springer, 2012.
13. W.-K. Hon, R. Shah, S. V. Thankachan, and J. S. Vitter. Space-efficient framework for top-k string retrieval. In *JACM*, 2014.
14. J. JáJá, C. W. Mortensen, and Q. Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *ISAAC*, volume 3341 of *Lecture Notes in Computer Science*, pages 558–568. Springer, 2004.
15. Y. Matias, S. Muthukrishnan, S. C. Sahinalp, and J. Ziv. Augmenting suffix trees, with applications. In *ESA*, volume 1461 of *Lecture Notes in Computer Science*, pages 67–78. Springer, 1998.
16. S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *SODA*, pages 657–666. ACM/SIAM, 2002.
17. G. Navarro. Spaces, trees and colors: The algorithmic landscape of document retrieval on sequences. *CoRR*, abs/1304.6023, 2013.
18. Y. Nekrich and G. Navarro. Sorted range reporting. In *SWAT*, pages 271–282, 2012.