Computing discriminating and generic words

Gregory Kucherov, Yakov Nekrich, Tatiana Starikovskaya

Computing Discriminating and Generic Words

Gregory Kucherov[1], Yakov Nekrich[2], and Tatiana Starikovskaya[3,1]

[1] Laboratoire d'Informatique Gaspard Monge, Université Paris-Est & CNRS, Marne-la-Vallée, Paris, France, Gregory.Kucherov@univ-mlv.fr
[2] Department of Computer Science, University of Chile, Santiago, Chile, yakov.nekrich@googlemail.com
[3] Lomonosov Moscow State University, Moscow, Russia, tat.starikovskaya@gmail.com

Abstract. We study the following three problems of computing *generic* or *discriminating* words for a given collection of documents. Given a pattern P and a threshold d, we want to report *(i)* all longest extensions of P which occur in at least d documents, *(ii)* all shortest extensions of P which occur in less than d documents, and *(iii)* all shortest extensions of P which occur only in d selected documents. For these problems, we propose efficient algorithms based on suffix trees and using advanced data structure techniques. For problem *(i)*, we propose an optimal solution with constant running time per output word.

1 Introduction

Many text processing applications raise different variants of the following problem: given a collection of sequences, that we call *documents*, one wants to compute words (strings) that occur in a certain subset of these documents only, and therefore discriminate these documents against the others. Such words are called *discriminating* or *distinguishing* for the corresponding set of documents. A complementary problem is to compute words that are common to a selected subset of documents. Such words are called *generic* or *characteristic* for the corresponding subset.

In computational biology, for example, identifying words occurring exclusively in the genomic sequence of one species (or a family of species) is of interest (see e.g. [4]). Words common to a subset of biologically related DNA sequences (and preferably not occurring in other sequences under consideration) often carry a biological function related, in particular, to regulation, repair or evolutionary mechanisms. As a prominent illustration, the problem of identifying genomic words occurring in a given collection of upstream sequences of orthologous genes and absent in a set of upstream sequences of unrelated genes is one of the classical problems in computational biology [15]. Other applications include the identification

of genomic markers, or probe design for DNA microarrays. Besides computational biology applications, many methods of automated text categorization and text mining are based on discriminating and generic words.

In this paper, we study three problems related to discriminating and generic words. In all of them, we will be looking for strings which are *extensions* of a given pattern P (which may be the empty word), i.e. for strings which have P as a prefix. Assume we are given a collection of strings (documents) $T_1, T_2, \ldots, T_m$ of total length n.

The **first problem** is to compute all *maximal generic words*: given a pattern P and a threshold $d \leq m$, we want to report all maximal extensions $\overline{P}$ of P occurring in at least d distinct documents. *Maximal* here means that *any* extension of $\overline{P}$ should occurs in less than d documents.

In the **second problem**, called *minimal discriminating words*, we need to report, given a pattern P and a threshold $d \leq m$, all extensions $\overline{P}$ of P which occur in at most d documents and which are minimal, i.e. any prefix of $\overline{P}$ occurs in more than d documents.

Finally, the **third problem** computes all the minimal extensions $\overline{P}$ of P which occur only in documents within a given subset $T_{i_1}, T_{i_2}, \ldots, T_{i_d}$. Minimality condition means that any prefix of $\overline{P}$ must occur in documents other than $T_{i_1}, \ldots, T_{i_d}$.

As an example, consider $T_1 = ababa$, $T_2 = aabbba$, $T_3 = bbabcb$. The maximal generic words for $d = 2$ (and $P = \varepsilon$) are ab, bab and bba. Note that ab occurs in the three strings, but any of its extensions occurs in one string only. Minimal discriminating extensions of $P = b$ for $d = 2$ are bab and bb, where bab discriminates $\{T_1, T_3\}$ and bb discriminates $\{T_2, T_3\}$.

We are primarily interested in linear-space solutions to our problems. All our solutions are based on the generalized suffix tree of $T_1, T_2, \ldots, T_m$, denoted GST, that can be viewed as the suffix tree for the string $T_1\$_1T_2\$_2 \ldots T_m\$_m$. A leaf in the generalized suffix tree is associated with a suffix of some document T_i together with the index i of this document. It is well-known that GST can be computed in $O(n)$ time.

A summary of our results is as follows. For the first problem we propose a solution with the optimal time bound $O(|P| + output)$, based on a result for a variant of the *orthogonal range reporting* problem. Hereafter, *output* denotes the number of reported words. We consider a special type of three-dimensional orthogonal range queries over a set S of points (x, y, z) such that $1 \leq x \leq |S|$ and $1 \leq y, z \leq \log n$, i.e. two of the three coordinates are logarithmically bounded in n. We call it the *extended one-dimensional range reporting*. For this case, we show that a range query $Q = [a, b] \times [0, c] \times [0, d]$ can be answered in optimal time

$O(1 + npoints)$, where *npoints* is the number of reported points, using a data structure of $O(|S|)$ space and a universal table of $o(n)$ space.

For the *second problem*, we propose a solution with running time $O(|P| + \log\log n + output)$. The solution is based on a reduction to a problem from computational geometry. For the *third problem*, we propose a solution with time complexity $O(|P| + d \log\log m \cdot (1 + output))$. To obtain this, we consider a special variant of *weighted ancestor queries* problem, to which we propose an optimal solution inspired by the one proposed in [10] for a similar problem.

It is important to note that all our algorithms output the resulting words by reporting their loci in GST, rather then spelling the words themselves. This is because the latter may cost up to $\Omega(n^2)$ time, which is prohibitive, while the number of loci is obviously $O(n)$. On the other hand, an enumeration of the set of loci may be sufficient for many applications (possibly as a basis for further analysis, see, e.g., [10]). Note also that for the second and third problems, the additive term $|P|$ that appears in our complexity bounds comes from locating the locus of P in GST and can be deleted if P itself is specified by its locus in GST.

We assume familiarity with suffix trees. Given a suffix tree, the *locus* of a string P is defined as the highest explicit node labeled by an extension of P. The *string depth* of a node is the length of its label. For each node v of the generalized suffix tree GST, we store its weight $\mathsf{weight}(v)$ defined as the number of *distinct* documents whose suffixes occur in the subtree rooted at v. Values $\mathsf{weight}(v)$ can be computed in $O(n)$ time [3].

2 Maximal Generic Words

Consider a set of documents $T_1, \ldots, T_m$ of total length n. Our first problem problem is to efficiently answer the following queries: given a pattern P and a threshold $d \leq m$, enumerate all extensions $\overline{P}$ of P occurring in at least d documents and such that *any* extension of $\overline{P}$ occurs in less than d documents. We seek a solution using $O(n)$ space.

We present our solution in two parts. First, we reduce the solution to a special kind of three-dimensional orthogonal range reporting queries in which two of the three coordinates of the involved points are logarithmically bounded. We call these queries *extended one-dimensional range reporting*. The data structure supporting these queries will be described separately in the second part.

2.1 Main algorithm

For a node u of GST, we denote by $maxchild(u)$ the child node of u with maximum weight. For $j \in [1..m]$, we say that a node u is a *j-node* if $\mathsf{weight}(u) \geq j$ and $\mathsf{weight}(maxchild(u)) < j$. It is easily seen that for a given d, the loci of maximal generic words are precisely the d-nodes.

Consider sets L_i for $i = 0, 1, \ldots, m/s$ and $s = \lceil \log n \rceil$, where L_i contains all j-nodes for $j \in [is + 1..(i + 1)s]$. We further define trees $\mathcal{T}_i$, $i = 0, 1, \ldots, m/s$. Essentially, $\mathcal{T}_i$ is a compacted trie on labels of nodes from L_i. Nodes of $\mathcal{T}_i$ are the root of GST, nodes of L_i, and all nodes of GST which have at least two children containing a node from L_i in their subtrees. We connect nodes u, v with an edge if there is no other node of $\mathcal{T}_i$ on the path connecting u and v in GST. Furthermore, we label this edge with the string written along this path. Note that $\mathcal{T}_i$ contains at most $2L_i$ nodes and therefore occupies $O(|L_i|)$ space.

For every node $u \in \mathcal{T}_i$ we store its rank $\mathsf{preord}^i(u)$ in the pre-order traversal of $\mathcal{T}_i$, and the range $[\mathsf{minord}^i(u), \mathsf{maxord}^i(u)]$ where $\mathsf{minord}^i(u)$ and $\mathsf{maxord}^i(u)$ are respectively the minimal and the maximal ranks of nodes in the subtree of $\mathcal{T}_i$ rooted at u.

For each L_i, $i = 0, 1, \ldots, m/s$, we maintain a data structure E_i storing a set of three-dimensional points. For every node u of L_i, E_i contains a point p_u, where $p_u.x = \mathsf{preord}^i(u)$, $p_u.y = \min(s, \mathsf{weight}(u) - i \cdot s)$, and $p_u.z = \max(0, \mathsf{weight}(maxchild(u)) - i \cdot s)$.

Observe that x-coordinates of all points in E_i are distinct integers bounded by $2|L_i|$, and that y- and z-coordinates belong to the interval $[0, \lceil \log n \rceil]$. E_i will be defined in Section 2.2, it takes space $O(|L_i|)$.

Computing all maximal generic extensions of a query pattern P is done as follows. First we find the locus of P, denoted v, in the tree $\mathcal{T}_{d'}$ with $d' = \lfloor d/s \rfloor$ (i.e., the node of $\mathcal{T}_{d'}$ of minimal depth with the label starting with P). This takes time $O(|P|)$. Then, we compute all points $p_u \in E_{d'}$ belonging to the three-dimensional range

$$[\mathsf{minord}^{d'}(v), \mathsf{maxord}^{d'}(v)] \times [d - d's, s] \times [0, d - d's - 1]. \qquad (*)$$

We will show in Section 2.2 how such queries will be answered in time $O(1 + output)$, where $output$ is the number of reported points, on the data structure $E_{d'}$. A node u is a d-node if and only if $p_u.y \geq d - d's$ and $p_u.z \leq d - d's - 1$. Therefore, answering the above query provides all the d-nodes located in the subtree rooted at v, that are loci of the desired maximal extensions of P.

To show that the data structures E_i take space $O(n)$ altogether, it is sufficient to show that $\sum_i |L_i| = O(n)$. First note that this sum is

equal to $\sum_{u \in GST} \lceil (\mathsf{weight}(u) - \mathsf{weight}(\mathit{maxchild}(u)))/s \rceil$ since every node u participates in $\lceil \mathsf{weight}(u) - \mathsf{weight}(\mathit{maxchild}(u)))/s \rceil$ sets L_i. On the other hand, $\sum_{u \in GST} (\mathsf{weight}(u) - \mathsf{weight}(\mathit{maxchild}(u)))$ is equal to the total number of j-nodes for all $j \in [1..m]$. Since each j-node has at least j leaves in its subtree and no j-node is an ancestor of another one, the number of j-nodes is at most n/j for any fixed j. Therefore $\sum_i |L_i| = \frac{1}{s} \sum_{j=1}^{m} (n/j) + O(n) = O(n)$.

We also have to explain how trees $\mathcal{T}_i$ are constructed. Recall that a node u of GST belongs to L_i if $\mathsf{weight}(\mathit{maxchild}(u)) < (i+1)s$ and $\mathsf{weight}(u) \geq is+1$. Furthermore, u belongs to $\mathcal{T}_i$ if either it belongs to L_i or the weight of at least two children of u is bigger than $(is+1)$, which means that u has at least two children each containing a node from L_i in its subtree. Therefore, given a node u, each index i of a tree $\mathcal{T}_i$ that u belongs to is retrieved in constant time. Thus, we can perform one post-order traversal of GST and build all $\mathcal{T}_i$, $i = 0, 1, \ldots, m/s$, in time $O(n)$. Once the trees are built, we need $O(n)$ time to assign the rank $\mathsf{preord}^i(u)$ and the interval $[\mathsf{minord}^i(u), \mathsf{maxord}^i(u)]$ to every node u of every tree $\mathcal{T}_i$. Observe also that the label of an edge of $\mathcal{T}_i$ connecting nodes u, v of GST can be computed in $O(1)$ time if we know string depths of u, v and the label of the last edge on the path connecting u and v in GST. Therefore, edge labels can be computed in $O(n)$ time as well.

We conclude with the final result of Section 2. Its proof follows from the previous discussion, subject to the description of data structures E_i that will be given in the next section.

Theorem 1. *For any pattern P and an integer d, the loci of all maximal extensions of P can be found in time $O(|P| + \mathit{output})$, where output is the number of such extensions. The underlying indexing structure takes $O(n)$ space and can be constructed in $O(n)$ time, where $n = |T_1| + |T_2| + \ldots + |T_m|$.*

2.2 Extended One-Dimensional Range Reporting Queries

We now describe the data structures E_i that allow queries $(*)$ to be answered in constant time per output point. We reformulate the problem as follows. Suppose that a set S of 3D integer points is given and for each point $p \in S$, we have $1 \leq p.x \leq |S| \leq n$, $0 \leq p.y, p.z \leq \log n$. Our goal is to report all points of S within a 3D range $Q = [a, b] \times [0, c] \times [0, d]$ in $O(|Q \cap S| + 1)$ time using space $O(|S|)$. Moreover, we will also use a universal look-up table of size $o(n)$ shared by the instances of our data structure (i.e. by different E_i). We assume that all points of S have different x-coordinates, which is the case in our setting. Our approach is similar

to the solution of the external memory point location problem from [13]. Similar problems for $d \geq 2$ dimensions were studied in e.g. [9]

We first describe data structures for a small set of points in Proposition 1 and Lemmas 1, 2, that are used in the final result (Lemma 3).

Proposition 1. *If $|S| \leq \log^{2/3} n$, we can store a set S in an $O(|S|)$-space data structure so that for any $Q = [a, b] \times [0, c] \times [0, d]$ all points in $Q \cap S$ can be answered in $O(|S \cap Q| + 1)$ time. The data structure uses a universal lookup table of size $o(n)$.*

Proof. We can assume w.l.o.g. that coordinates of points in S belong to the rank space, i.e., coordinates are integers and $0 \leq p.x, p.y, p.z \leq \log^{2/3} n - 1$ for all p. If points are arbitrary integers, we can apply the reduction to rank space technique [7] and obtain a set of points that satisfies this condition. Answers to all possible queries for all such S are stored in a lookup table. There are less than $(3 \log^{2/3} n)^{\log^{2/3} n}$ different sets S, $\log^{8/3} n$ queries can be asked, and the answer to a query contains $O(\log^{2/3} n)$ points. Therefore the lookup table has $o(n)$ entries and can be stored in $o(n)$ space. □

Lemma 1. *Suppose that for every point $p \in S$, $1 \leq p.x \leq \log^2 n$, $0 \leq p.y, p.z \leq \log^{1/3} n$. There exists a data structure that uses $O(|S|)$ space and a universal table of size $o(n)$ while answering a query $Q = [a, b] \times [0, c] \times [0, d]$ in time $O(|S \cap Q| + 1)$.*

Proof. We divide S into blocks W_i such that each W_i contains $\lfloor \log^{2/3} n \rfloor$ points except possibly for the last block that may contain less. For any $p_i \in W_i$ and $p_j \in W_j$, $p_j.x > p_i.x$ iff $j > i$. For every pair $0 \leq i, j \leq \log^{1/3} n$, we store the list L_{ij}. If a block W_t contains points p such that $p.y \leq i$ and $p.z \leq j$, then L_{ij} contains one representative point $p_t \in W_t$, $p_t.y \leq i$ and $p_t.z \leq j$. Let m_t denote the minimal x-coordinate of a point $p \in W_t$. Since there are $O(\log^{4/3} n)$ blocks, we can search among m_i and find the biggest $m_i \leq a$ for any a in $O(1)$ time using a Q-heap data structure [6]. Furthermore, we can find all points in $W_t \cap Q$ for any query Q using Proposition 1.

Consider a query $Q = [a, b] \times [0, c] \times [0, d]$. We find the largest $m_l \leq a$, the largest $m_h \leq b$, and report all points in $Q \cap W_h$ and $Q \cap W_l$. Then we find all points $p \in L_{cd}$ such that $m_{l+1} \leq p.x \leq m_h$. For every such p we examine the block W_p containing p and report all points from $Q \cap W_p$. □

Lemma 2. *Suppose that for every point $p \in S$, $1 \leq p.x \leq \log^2 n$, $0 \leq p.y \leq \log n$, and $0 \leq p.z \leq \log n$. There exists a data structure that uses*

$O(|S|)$ *space and a universal table of size* $o(n)$ *while answering a query* $Q = [a, b] \times [0, c] \times [0, d]$ *in time* $O(|S \cap Q| + 1)$.

Proof. First, we consider the case when $p.x \leq \log^2 n$, $p.y \leq \log n$, and $p.z \leq \log^{1/3} n$. Our data structure is a range tree [1] $\mathbb{T}_y$ on y-coordinates. Each leaf of $\mathbb{T}_y$ contains one point and each internal node has $\log^{1/3} n$ children. Every internal node $v \in \mathbb{T}_y$ contains a data structure $\mathbb{F}_v$. For each point p in the range of v, $\mathbb{F}_v$ contains a point p' with coordinates $p'.x = p.x$, $p'.z = p.z$, and $p'.y = i$ so that p also belongs to the range of the i-th child v_i of v. In other words, the y-coordinate of p is replaced with the index of the child of v that also contains p. $\mathbb{F}_v$ is implemented according to Lemma 1.

Consider a query $Q = [a, b] \times [0, c] \times [0, d]$. We can find $O(1)$ nodes $u^1, \ldots, u^t$ such that some children $u^1_{i_1}, \ldots, u^1_{j_1}, \ldots, u^t_{i_t} \ldots, u^t_{j_t}$ of $u^1, \ldots, u^t$ respectively cover $[0, c]$. For every such u^f we answer a query $[a, b] \times [i_f, j_f] \times [0, d]$ using the data structure $\mathbb{F}_{u^f}$.

The case when $1 \leq p.x \leq \log^2 n$, $0 \leq p.y \leq \log n$, and $0 \leq p.z \leq \log n$ is handled using the same method. We construct a range tree $\mathbb{T}$ on z-coordinates. Each internal node u of $\mathbb{T}$ has degree $\log^{1/3} n$. Again, we replace the z-coordinate of each point p in the range of u with the index of the child of u that also contains p and then build a tree $\mathbb{T}^u_y$ on those points. A query is answered by reducing it to $O(1)$ queries on data structures $\mathbb{T}^u_y$ as described above. □

Lemma 3. *Suppose that for every point* $p \in S$, $1 \leq p.x \leq |S|$, $0 \leq p.y \leq \log n$ *and* $0 \leq p.z \leq \log n$. *There exists a data structure that uses* $O(|S|)$ *space and a universal table of size* $o(n)$ *and allows queries* $Q = [a, b] \times [0, c] \times [0, d]$ *to be answered in time* $O(|S \cap Q| + 1)$.

Proof. We divide the points into blocks according to their x-coordinates. Each block B_k, $k = 1, \ldots, \lceil |S| / \log^2 n \rceil$, contains all points p satisfying $(k-1)\lceil \log^2 n \rceil < p.x \leq k \lceil \log^2 n \rceil$. For every block B_s and each pair $0 \leq c, d \leq \log n$, we store a pointer to the last block B_r, $r < s$, that contains at least one point p such that $p.y \leq c$ and $p.z \leq d$. Since there are $O(\log^2 n)$ pointers in every block, all pointers use linear space. For each $1 \leq k \leq \lceil |S| / \log^2 n \rceil$, we also store a data structure $\mathbb{T}_k$ that supports queries $[a, b] \times [0, c] \times [0, d]$ on points that belong to a block B_k; $\mathbb{T}_k$ is implemented as described in Lemma 2.

To answer a query $[a, b] \times [0, c] \times [0, d]$, we report points in $B_l \cap Q$ and $B_r \cap Q$ using $\mathbb{T}_l$ and $\mathbb{T}_r$, where $l = \lceil a / \log^2 n \rceil$ and $r = \lceil b / \log^2 n \rceil$. If $r > l + 1$, we examine all blocks B_i, $l < i < r$, that contain at least one

point p, $p.y \leq c$ and $p.z \leq d$. In every such B_i, we report all $p \in B_i \cap Q$. Our search procedure visits every block B_i such that $B_i \cap Q \neq \emptyset$. By Lemma 2, we spend $O(|Q \cap B_i| + 1)$ time in every visited block B_i. Every visited B_i, except of B_l and B_r, contains at least one point $p \in Q$. Therefore the total query time is $O(\sum |Q \cap B_i| + 1) = O(|Q \cap S| + 1)$. □

3 Minimal Discriminating Words

We now turn to the problem of computing words that *discriminate* a subset of documents. Given a pattern P and a threshold $d \leq m$, we want to find all minimal extensions $\overline{P}$ of P which occur in at most d distinct documents. "Minimal" here means that no prefix of $\overline{P}$ satisfies this property. We describe a linear-space data structure for this problem.

Consider the generalized suffix tree GST. We start with locating the locus of P in GST in $O(|P|)$ time in the usual way. If v does not exist or $\mathsf{weight}(v) < d$, then, obviously, P has no desired extensions. Otherwise, we have to find all nodes u in the subtree rooted at v such that $\mathsf{weight}(u) \leq d$ and $\mathsf{weight}(p(u)) > d$, where $p(u)$ is the parent of u. Then, the desired extension of P will be the label of $p(u)$ extended by the first letter of the label of edge $(p(u), u)$.

Unfortunately, applying the solution of Section 2 to this task would take too much space. To illustrate this, let each of the documents $T_1, T_2, \ldots, T_m$ be a distinct letter of the alphabet. Then GST has a root of weight m and n leaves of weight one (here $m = n$). If we consider j-nodes as nodes of GST such that $\mathsf{weight}(u) \leq j \leq \mathsf{weight}(p(u))$, then each leave is a j-node for $j \in [1..m]$, and, therefore, each set L_i, $i = 0, 1, \ldots, m/\log n$, contains all n leaves of GST. That is, the total number of nodes in L_i, $i = 0, 1, \ldots, m/\log n$, will be $O(nm/\log n)$.

We then take a different approach and reduce the problem to the *orthogonal segment intersection problem*.

To each node u of GST, we associate a horizontal segment $[\mathsf{weight}(u), \mathsf{weight}(p(u)) - 1]$ placed on the two-dimensional quarter-plane with y-coordinate $\mathsf{preord}(u)$, where $\mathsf{preord}(u)$ is the rank of u in the preorder traversal of GST. Furthermore, for each node u, we store $\mathsf{minord}(u)$ and $\mathsf{maxord}(u)$ which are respectively the minimal and the maximal ranks of nodes of the subtree of GST rooted at u in the preorder sequence of GST. All nodes of the subtree rooted at u appear then in the interval $[\mathsf{minord}(u), \mathsf{maxord}(u)]$ of the preorder sequence.

Let $(x_s, [y_1, y_2])$ denote the vertical segment with endpoints (x_s, y_1) and (x_s, y_2). Our problem is then to identify all horizontal segments that

intersect with the vertical segment $(d, [\mathsf{minord}(v), \mathsf{maxord}(v)])$. These horizontal segments can be found in time $O(\log\log n + \mathit{output})$ and space $O(n)$ [2]. Therefore, the following theorem holds.

Theorem 2. *Given a pattern P and an integer d, the loci of all minimal discriminating extensions of P can be found in time $O(|P| + \log\log n + \mathit{output})$. The underlying indexing structure takes $O(n)$ space and can be constructed in $O(n \log n)$ time, where $n = |T_1| + |T_2| + \ldots + |T_m|$.*

4 Discriminating Words for Specified Documents

In Section 3, we showed how to compute words that discriminate d documents from among m documents, without prior knowledge of what these documents are. In many applications, we need to compute words that discriminate documents from a *given sample*. Consider a set of documents $T_1, \ldots, T_m$. Given a set of indices $\{i_1, i_2, \ldots, i_d\}$ and a pattern P, we want to find all extensions of P occurring *only* in documents $T_{i_1}, T_{i_2}, \ldots, T_{i_d}$ and such that any of their prefixes has at least one occurrence in a document which does not belong to this subset.

In this section, we propose a linear space data structure which allows to compute these extensions in time $O(|P| + d \log\log m \cdot (1 + \mathit{output}))$, where *output* is the number of such extensions.

We will need the following variant of the *weighted level ancestor* problem [5]. Given a tree $\mathcal{T}$ of size n, assume that each node u of $\mathcal{T}$ is assigned a positive integer $\mathsf{weight}(u) \in n^{O(1)}$ decreasing along every root-to-leaf path: if u is an ancestor of v, then $\mathsf{weight}(v) \leq \mathsf{weight}(u)$. The answer to an *approximate weighted ancestor query* (u, d), for a node u and an integer $d \geq \mathsf{weight}(u)$, is an ancestor w of u satisfying $d \leq \mathsf{weight}(w) < 2d$.

Lemma 4. *$\mathcal{T}$ can be stored in a linear space data structure so that approximate weighted ancestor queries on $\mathcal{T}$ can be answered in $O(1)$ time.*

Proof. The data structure is similar to the one from the proof of Theorem 1 in [10]. The only difference is that the data structure $E(\pi_j)$ that finds a predecessor on a path π_j is replaced by the structure of [11] that answers approximate predecessor queries: for any integer d, we can find a node $u \in \pi_j$ such that $d \leq \mathsf{weight}(u) < 2d$ in $O(1)$ time. □

We now describe the data structures we use for reporting the minimal extensions for specified documents. Consider the generalized suffix tree GST for $T_1, T_2, \ldots, T_m$. For each vertex v of GST, we store $L(v)$ and $R(v)$ defined to be the rank of respectively the leftmost and the rightmost

leaf in the subtree of v. These ranks can be computed in $O(n)$ time by post-processing GST. Moreover, for each leaf v of GST, we pre-compute its ancestor u of minimal depth such that $L(u) = v$ (u can coincide with v). Finally, we augment GST with a data structure that answers lowest common ancestor queries in constant time [14].

Each leaf of GST is associated with a suffix of some document T_k and the index k of this document. We store these document indices in an array D, called the *document array*, such that $D[i] = k$ if the i-th leaf of GST in the left-to-right order corresponds to a suffix coming from T_k.

We augment D with two data structures. The first one is an $O(n)$-space data structure that answers queries $\mathsf{rank}(k, i)$ (number of entries storing k before position i in D) and $\mathsf{select}(k, i)$ (i-th entry from the left storing k). Using the result of [8], we can support such rank and select queries in $O(\log\log m)$ and $O(1)$ time respectively. The second one is the linear-space data structure from [12]. It allows us to report all distinct document indices in an interval of D in time $O(1)$ per each output index.

We are now ready to describe the algorithm. We start with locating the locus v of P in GST in time $O(|P|)$ and retrieving the interval $[L(v)..R(v)]$. Starting from $\ell^{(0)} = L(v)$, the algorithm will process the interval $[\ell^{(i)}..R(v)]$ and compute a desired extension covering the leftmost subrange of suffixes of this interval, then it will iteratively proceed to a smaller interval $[\ell^{(i+1)}..R(v)]$, $\ell^{(i+1)} > \ell^{(i)}$.

To process an interval $[\ell^{(i)}..R(v)]$, we first check if it contains any of document indices $i_1, i_2, \ldots, i_d$, and locate the smallest of them as follows. Using d rank queries and d select queries on D we find the minimum rank *minrank* of any of $i_1, i_2, \ldots, i_d$ in $D[\ell^{(i)}..R(v)]$. This costs $O(d \log\log m)$ time. If no index of $i_1, i_2, \ldots, i_d$ occurs in $D[\ell^{(i)}..R(v)]$, the whole algorithm terminates.

Let s be the leaf of GST with rank *minrank*. Then the suffix corresponding to this leaf must have a prefix which is an extension we are looking for, whose locus is an ancestor x of s. We will find x in two steps: first, we find the longest interval of D starting at *minrank* and containing only indices $i_1, i_2, \ldots, i_d$, and then we will find the highest ancestor of s such that the ranks of all the suffixes in its subtree belong to this interval. This ancestor will be x.

First step. Using approximate weighted ancestor we can find an ancestor u of s such that $\frac{3}{2}d \leq \mathsf{weight}(u) < 3d$ in $O(1)$ time. (If u is an ancestor of v, we set $u = v$.) The interval $D[minrank..R(u)]$ includes the longest interval starting at *minrank* which contains only indices $i_1, i_2, \ldots, i_d$. On the other hand, interval $D[minrank..R(u)]$ contains indices of less than

$3d$ different documents, and we output these indices in time $O(d)$ using the data structure of [12]. For each of these documents which is not among $i_1, i_2, \ldots, i_d$, we compute the smallest rank greater than *minrank* using one rank and one select queries, and then take the minimum of them, denoted *minrank*$'$. We set *minrank*$' = R(u)$ if $D[minrank..R(u)]$ contains only indices $i_1, i_2, \ldots, i_d$. This step takes $O(d \log\log m)$ time. $[minrank, minrank' - 1]$ is the longest segment of D starting at *minrank* and containing only indices $i_1, i_2, \ldots, i_d$.

Second step. We now compute x which is the ancestor of s of minimal depth such that *(i)* $L(x) = s$, and *(ii)* $R(x) < minrank'$. We first retrieve the ancestor u' of s of minimal depth such that $L(u') = s$. Let u'' denote the node with greater depth among u and u'. If $R(u'') < minrank'$, then $x = u''$ and we are done. Otherwise, compute the lowest common ancestor w of s and s', where s' is the leaf of rank $(minrank' - 1)$. If $R(w) = s'$, then $x = w$. Otherwise, $x = w_1$, where w_1 is the leftmost child of w. The second step takes $O(1)$ time.

We then proceed to the next iteration with the interval $[\ell^{(i+1)}..R(v)]$, where $\ell^{(i+1)} = R(x) + 1$. Each iteration takes $O(d \log\log m)$ time; each iteration, except of possibly the last one, outputs at least one desired locus. This leads to the final result:

Theorem 3. *Given a subset of indices $\{i_1, i_2, \ldots, i_d\}$ and a pattern P, all minimal extensions of P which occur only in the documents $T_{i_1}, T_{i_2}, \ldots, T_{i_d}$ can be computed in time $O(|P| + d \log\log m \cdot (1 + output))$, where m is the total number of documents. The underlying indexing data structure occupies $O(n)$ space and can be constructed in $O(n)$ time, where $n = |T_1| + |T_2| + \ldots + |T_m|$.*

5 Concluding Remarks

Our solution to the first problem (Section 2) is optimal: it takes a linear space and a constant time per output item. It is an interesting question if the second problem can admit an optimal solution too. Improving the bound for the third problem is another interesting direction to study.

Acknowledgments: G.Kucherov has been partly supported by the Marie-Curie Intra-European Fellowship for Career Development. T.Starikovskaya has been supported by the mobility grant funded by the French Ministry of Foreign Affairs through the EGIDE agency and by a grant 10-01-93109-CNRS-a of the Russian Foundation for Basic Research. Part of this work has been done during a stay of Y.Nekrich at

the Marne-la-Vallée University supported by the BEZOUT grant of the French government.

References

1. J.L. Bentley. Multidimensional divide-and-conquer. *Comm. ACM*, 23(4):214–229, 1980.
2. T.M. Chan. Persistent Predecessor Search and Orthogonal Point Location on the Word RAM. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 1131–1145, 2011.
3. L. Chi Kwong Hui. Color set size problem with applications to string matching. In Alberto Apostolico, Maxime Crochemore, Zvi Galil, and Udi Manber, editors, *Combinatorial Pattern Matching*, volume 644 of *Lecture Notes in Computer Science*, pages 230–243. Springer Berlin / Heidelberg, 1992.
4. A. Fadiel, S. Lithwick, G. Ganji, and S. W. Scherer. Remarkable sequence signatures in archaeal genomes. *Archaea*, 1(3):185–190, Oct 2003.
5. M. Farach and M. Muthukrishnan. Perfect hashing for strings: Formalization and algorithms. In *Combinatorial Pattern Matching*, volume 1075 of *Lecture Notes in Computer Science*, pages 130–140. Springer Berlin / Heidelberg, 1996.
6. M.L. Fredman and D.E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48(3):533–551, 1994.
7. H.N. Gabow, J.L. Bentley, and R.E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th Annual ACM Symposium on Theory of Computing (STOC 1984)*, pages 135–143, 1984.
8. A. Golynski, J.I. Munro, and S.S. Rao. Rank/select operations on large alphabets: a tool for text indexing. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 368–373. ACM Press, 2006.
9. J. JáJá, C.W. Mortensen, and Q. Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Proceedings of the 15th International Symposium on Algorithms and Computation,*, pages 558–568, 2004.
10. G. Kucherov, Y. Nekrich, and T. Starikovskaya. Cross-document pattern matching. In J. Kärkkäinen and J. Stoye, editors, *Proceedings of the 23rd Annual Symposium on Combinatorial Pattern Matching, July 3-5, 2012, Helsinki (Finland)*, volume 7354 of *Lecture Notes in Computer Science*, pages 196–207. Springer Verlag, 2012.
11. Y. Matias, J.S. Vitter, and N.E. Young. Approximate data structures with applications. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 187–194, 1994.
12. S. Muthu Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.
13. Y. Nekrich. I/O-efficient point location in a set of rectangles. In *Procedings of the 8th Latin American Symposium on Theoretical Informatics*, pages 687–698, 2008.
14. B. Schieber and U. Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM Journal on Computing*, 17:111–123, 1988.
15. M. Tompa et al. Assessing computational tools for the discovery of transcription factor binding sites. *Nat. Biotechnol.*, 23(1):137–144, Jan 2005.