Succinct Indexes for Reporting Discriminating and Generic Words*

Sudip Biswas, Manish Patil, Rahul Shah, and Sharma V. Thankachan

Louisiana State University, USA
{sbiswas,mpatil,rahul,thanks}@csc.lsu.edu

Abstract. We consider the problem of indexing a collection $\mathcal{D}$ of D strings (documents) of total n characters from an alphabet set of size σ, such that whenever a pattern P (of p characters) and an integer $\tau \in [1, D]$ comes as a query, we can efficiently report all (i) *maximal generic words* and (ii) *minimal discriminating words* as defined below:

- *maximal generic word* is a maximal extension of P occurring in at least τ documents..
- *minimal discriminating word* is a minimal extension of P occurring in at most τ documents.

These problems were introduced by Kucherov et al. [8], and they proposed linear space indexes occupying $O(n \log n)$ bits with query times $O(p + output)$ and $O(p + \log \log n + output)$ for Problem (i) and Problem (ii) respectively. In this paper, we describe succinct indexes of $n \log \sigma + o(n \log \sigma) + O(n)$ bits space with near optimal query times i.e., $O(p + \log \log n + output)$ for both these problems.

1 Introduction and Related Work

Let $\mathcal{D}=\{d_1, d_2, d_3, ..., d_D\}$ be a collection of D strings (which we call as documents) of total n characters from an alphabet set Σ of size σ. For simplicity we assume, that every document ends with a special character \$ which does not appear any where else in the documents. Our task is to index $\mathcal{D}$ in order to compute all (i) *maximal generic words* and (ii) *minimal discriminating words* corresponding to the given query pattern P (of length p) and threshold τ. The document frequency $df(.)$ of a pattern P is defined as the number of distinct documents in $\mathcal{D}$ containing P. Then, a generic word is an extension $\bar{P}$ of P with $df(\bar{P}) \geq \tau$, and is maximal if $df(P') < \tau$ for all extensions P' of $\bar{P}$. Similarly, a discriminating word is an extension $\bar{P}$ of P with $df(\bar{P}) \leq \tau$, and is called a minimal discriminating word if $df(P') > \tau$ for any proper prefix P' of $\bar{P}$ (i.e., $P' \neq \bar{P}$). These problems were introduced by Kucherov et al. [8], and they proposed indexes of size $O(n \log n)$ bits or $O(n)$ words. The query processing time is optimal $O(p + output)$ for reporting all maximal generic words, and is near optimal $O(p + \log \log n + output)$ for reporting all minimal discriminating words.

* This work is supported in part by National Science Foundation (NSF) Grants CCF–1017623 and CCF–1218904.

E. Moura and M. Crochemore (Eds.): SPIRE 2014, LNCS 8799, pp. 89–100, 2014.

Later on Gawrychowski et al. [6] gave $O(n)$ words space with optimal query time index for minimal discriminating words problem. In this paper, we describe succinct indexes of $n \log \sigma + o(n \log \sigma) + O(n)$ bits space with $O(p + \log \log n + output)$ query times for both these problems.

These problems are motivated from applications in computational biology. For example, it is an interesting problem to identify words that are exclusive to the genomic sequences of one species or family of species [3]. Such patterns that appear in a small set of biologically related DNA sequences but do not appear in other sequences the collection often carries a biological significance. Discriminating and generic words also find applications in text mining and automated text classification.

2 Preliminaries

2.1 Suffix Trees and Generalized Suffix Trees

For a text $S[1...n]$, a substring $S[i...n]$ with $i \in [1, n]$ is called a suffix of T. The suffix tree [13,9] of S is a lexicographic arrangement of all these n suffixes in a compact trie structure of $O(n)$ words space, where the i-th leftmost leaf represents the i-th lexicographically smallest suffix of S. For a node i (i.e., node with pre-order rank i), $path(i)$ represents the text obtained by concatenating all edge labels on the path from root to node i in a suffix tree. The locus node i_P of a pattern P is the node closest to the root such that the P is a prefix of $path(i_P)$. The suffix range of a pattern P is given by the maximal range $[sp, ep]$ such that for $sp \leq j \leq ep$, P is a prefix of (lexicographically) j-th suffix of S. Therefore, i_P is the lowest common ancestor of sp-th and ep-th leaves. Using suffix tree, the locus node as well as the suffix range of P can be computed in $O(p)$ time, where p denotes the length of P. Let $T = d_1 d_2 ... d_D$ be the text obtained by concatenating all documents in $\mathcal{D}$. Recall that each document is assumed to end with a special character \$. The suffix tree of T is called the generalized suffix tree (GST) of $\mathcal{D}$.

Encoding of GST with the goal of supporting navigation and other tree operations has been extensively studied in the literature. We use the data structure by Sadakane and Navarro [12] with focus on following operations:

- $lca(i, j)$: the lowest common ancestor of two nodes i, j
- $child(i, k)$: k-th child of node i
- $level\text{-}ancestor(i, d)$: ancestor j of i such that depth(j)=depth(i)-d
- $subtree\text{-}size(i)$: number of nodes in the subtree of node i

Lemma 1. *[12] An ordinal tree with m nodes can be encoded by $2m + O(\frac{m}{polylog(m)})$ bits supporting lca, k-th child, level-ancestor, and subtree-size queries in constant time.*

Define $count(i) = df(path(i))$. Using the data structure by Sadakane [11] we can answer $count(i)$ query efficiently for any input node i. Following lemma summarizes the result in [11].

Lemma 2. *[11] Generalized suffix tree (GST) with n leaves can be encoded by* $2n + o(n)$ *bits, supporting count*(i) *query in constant time.*

2.2 Marking Scheme in GST

Here we briefly explain the marking scheme introduced by Hon et al. [7] which will be used later in the proposed succinct index. We identify certain nodes in the GST as marked nodes and prime nodes with respect to a parameter g called the *grouping factor*. The procedure starts by combining every g consecutive leaves (from left to right) together as a group, and marking the lowest common ancestor (LCA) of first and last leaf in each group. Further, we mark the LCA of all pairs of marked nodes recursively. We also ensure that the root is always marked. At the end of this procedure, the number of marked nodes in GST will be not more than $2n/g$. Hon et al. [7] showed that, given any node u with u^* being its highest marked descendent (if exists), number of leaves in $GST(u \backslash u^*)$ i.e., the number of leaves in the subtree of u, but not in the subtree of u^* is at most $2g$.

Prime nodes are the children of marked nodes (illustrated in figure 1). Corresponding to any marked node u^* (except the root node), there is a *unique prime* node u', which is its closest prime ancestor. In case u^*'s parent is marked then $u' = u^*$. For every prime node u', the corresponding closest marked descendant u^* (if it exists) is unique.

2.3 Segment Intersection Problem

In [8] authors have shown how the problem of identifying minimal discriminating words can be reduced to orthogonal segment intersection problem. In this article, we rely on this key insight for both the problems under consideration and use the result summarized in lemma below for segment intersection.

Lemma 3. *[2] A given set* $\mathcal{I}$ *of n vertical segments of the form* $(x_i, [y_i, y'_i])$, *where* $x_i, y_i, y'_i \in [1, n]$ *can be indexed in* $O(n)$*-word space (in Word RAM model), such that whenever a horizontal segment* $s_q = ([x_q, x'_q], y_q)$ *comes as a query, all those vertical segments in* $\mathcal{I}$ *that intersect with* s_q *can be reported in* $O(\log \log n +$ *output*$)$ *time.*

2.4 Range Maximum Query

Let A be an array of length n, a range maximum query (RMQ) asks for the position of the maximum value between two specified array indices $[i, j]$. i.e., the RMQ should return an index k such that $i \leq k \leq j$ and $A[k] \geq A[x]$ for all $i \leq x \leq j$. We use the result captured in following lemma for our purpose.

Lemma 4. *[4,5] By maintaining a* $2n + o(n)$ *bits structure, range maximum query (RMQ) can be answered in* $O(1)$ *time (without accessing the array).*

3 Computing Maximal Generic Words

In this section, we first review a linear space index, which is based on the ideas from the previous results [8]. Later we show how to employ sampling techniques to achieve a space efficient solution.

3.1 Linear Space Index

Let i_P be the locus node of the query pattern P. Then, our task is to return all those nodes j in the subtree of i_P such that $count(j) \geq \tau$ and $count(.)$ of every child node of j is less than τ. Note that corresponding to each such output j, $path(j)$ represents a maximal generic word with respect to the query (P, τ). The mentioned task can be performed efficiently by reducing the original problem to a segment intersection problem as follows. Each node i in GST is mapped to a vertical segment $(i, [count(i_{max}) + 1, count(i)])$, where i_{max} is the child of i with the highest $count(.)$ value. If i is a leaf node, then we set $count(i_{max}) = 0$. Moreover, if $count(i) = count(i_{max})$ we do not maintain such a segment as it can not possibly lead to a generic word. The set $\mathcal{I}$ of these segments is then indexed using a linear space structure as described in Section 2.3. Additionally we maintain the GST of $\mathcal{D}$ as well.

The maximal generic words corresponding to a query (P, τ) can be computed by issuing an orthogonal segment intersection query on $\mathcal{I}$ with $s_q = ([i_P, i'_P], \tau)$ as the input, where i_P is the locus node of pattern P and i'_P represents the rightmost leaf in the subtree of i_P. It can be easily verified than $path(j)$ corresponding to each retrieved interval $(j, [count(j_{max}) + 1, count(j)])$ is a maximal generic word. In conclusion, we have a linear space index of $O(n)$ words with $O(\log \log n + output)$ query time. By combining with the space for GST, and the initial $O(p)$ time for pattern search where p is the size of pattern, we have the following lemma.

Lemma 5. *There exists an $O(n)$ word data structure for reporting maximal generic word queries in $O(p + \log \log n + output)$ time.* □

3.2 Succinct Space Index

Our succinct space index for computing maximal generic words has the following key components:

- For finding maximal generic words corresponding to the marked nodes, we store a data structure similar to the one in [8] described above only for the marked nodes, which takes space linear to the number of marked nodes.
- To capture the outputs falling in the path between two marked nodes, we store a segment intersection index along with encoding of the path between a marked node and its unique lowest prime ancestor.
- The remaining output fall in small subtrees which can be efficiently found using bit encodings of subtrees and table structure.

At first,, we begin by extending the marking scheme (illustrated in figure 1) of Hon et al. [7] described earlier in Section 2.2 and then discuss our succinct space index. We introduce the notion of *orphan* and *maximal orphan* nodes in GST based on the marking scheme of Hon et al. [7] as follows:

1. *Orphan node* is a node with no marked node in its subtree. Note that the number of leaves in the subtree of an orphan node is at most g.
2. *maximal orphan* is an orphan node with *non-orphan* parent. Therefore, every orphan node has a unique maximal orphan ancestor. The number of leaves in the subtree of any maximal orphan node is at most g and the number of maximal orphan nodes can be $\omega(n/g)$.

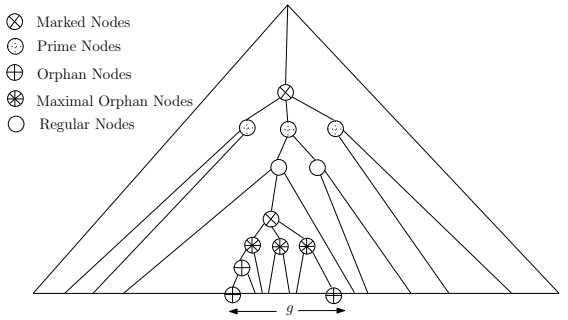

Fig. 1. Marking Scheme

We now describe a structure to solve a variant of computing maximal generic words summarized in lemma below that forms the basis of our final space-efficient index. We choose $g = \lfloor \frac{1}{8} \log n \rfloor$ as a grouping parameter in the marking scheme of Hon et al. [7] and nodes in GST are identified by their pre-order rank.

Lemma 6. *The collection $\mathcal{D}$=$\{d_1, d_2, d_3, ..., d_D\}$ of strings of total n characters can be indexed in $(n \log \sigma + O(n))$ bits, such that given a query (P, τ), we can identify all marked nodes j^* satisfying either of the following condition in $O(p + \log \log n + output)$ time.*

- *$path(j^*)$ is a maximal generic word for input (P, τ)*
- *there is at least one node in $N(j^*) = GST(j'/j^*)$ that corresponds to desired maximal generic word, where j' is a unique lowest prime ancestor of j^**
- *j^* is a highest marked descendant of locus node for pattern P*

Proof. It can be noted that, $N(j^*)$ is essentially a set of all nodes in the subtree of j' but not in the subtree of j^* ($N(j^*)$ does not include node j^*). To be able to retrieve the required marked nodes in succinct space, we maintain index consisting of following components:

- Instead of GST, we use its space efficient version i.e., a compressed suffix array (CSA) of T. There are many versions of CSA's available in literature, however for our purpose we use the one in [1] that occupies $n \log \sigma + o(n \log \sigma) + O(n)$ bits space and retrieves the suffix range of pattern P in $O(p)$ time.
- A $4n + o(n)$ bits encoding of GST structure (Lemma 1) to support the (required) tree navigational operations in constant time.
- We keep a bit vector $B_{mark}[1...2n]$, where $B_{mark}[i] = 1$ iff node i is a marked node, with constant time rank-select supporting structures over it in $O(n)$ bits space [10]. This bit vector enables us to retrieve the unique highest marked descendant of any given node in GST in constant time.
- We map each marked node i^* to a vertical segment $(i^*, [count(i^*_{max}) + 1, count(i^*)])$, where i^*_{max} is the child of i^* with the highest $count(.)$ value. The number of such segments can be bounded by $O(n/g)$. The set $\mathcal{I}_1$ of these segments is then indexed using a linear space structure as before in $O(n/g) = O(n/\log n)$ words or $O(n)$ bits space. We note that segments with $count(i^*_{max}) = count(i^*)$ are not included in set $\mathcal{I}_1$, as marked nodes corresponding to those segments can not possibly lead to a generic word.
- We also maintain $O(n)$ bit structure for set $\mathcal{I}_2$ of segments of the form $(i', [count(i^*) + 1, count(i')])$, where i^* is a marked node and i' is the unique lowest prime ancestor of i^*. Once again the segments with $count(i^*) = count(i')$ are not maintained.

Given an input (P, τ), we first retrieve the suffix range in $O(p)$ time using CSA and the locus node i_P in another $O(1)$ time using GST structure encoding. Then we issue an orthogonal segment intersection query on $\mathcal{I}_1$ and $\mathcal{I}_2$ with $s_q = ([i_P, i'_P], \tau)$ as the input, i'_P being the rightmost leaf in the subtree of i_P. Any marked node that corresponds to a maximal generic word for query (P, τ) is thus retrieved by querying set $\mathcal{I}_1$. Instead of retrieving non-marked nodes corresponding to the maximal generic word, the structure just described returns their representative marked nodes instead.

For a segment in $\mathcal{I}_2$ that is reported as an answer, corresponding to a marked node j^* with j' being its lowest prime ancestor, we have $count(j') \geq \tau$ and $count(j^*) < \tau$. Therefore, there must exist a parent-child node pair (u, v), on the path from j' to j^* such that $count(u) \geq \tau$ and $count(v) < \tau$. As before, let u_{max} be the child of node u with the highest $count(.)$ value. It can be easily seen that if $(v = u_{max})$ or $(v \neq u_{max}$ with $count(u_{max}) < \tau)$, then $path(u)$ is a maximal generic word with respect to (P, τ). Otherwise, consider the subtree rooted at node u_{max}. For this subtree $count(u_{max}) \geq \tau$ and $count(.) = 1$ for all the leaves and hence it is guaranteed to contain at least one maximal generic word for query (P, τ).

We highlight that the segment intersection query on set $\mathcal{I}_2$ will be able to capture the node pairs (j^*, j') where both j^* and j' are in the subtree of locus node i_P as the segments in $\mathcal{I}_2$ use (pre-order rank of) prime node as their x coordinate. The case when both j^*, j' are outside the subtree of i_P can be ignored as in this case none of the nodes in $N(j^*)$ will be in the subtree of i_P and

hence can not lead to a desired output. Further we observe that for the remaining scenario when locus node i_P is on the path from j' to j^* (both exclusive), there can be at-most one such pair (j^*, j'). This is true due to the way nodes are marked in GST and moreover j^* will be the highest marked descendant of i_P (if exists). Such j^* can be obtained in constant time by first obtaining the marked node using query $select_1(rank_1(i_P)+1)$ on B_{mark} and then evaluating if it is in the subtree of i_P. We note that in this case $N(j^*)$ (j^* being the highest marked descendant of i_P) may or may not result in a maximal generic word for query (P, τ), however we can afford to verify it irrespective of the output due to its uniqueness. □

If a marked node j^* reported by the data structure just described is retrieved from set $\mathcal{I}_1$ then $path(j^*)$ can be returned as an maximal generic word for query (P, τ) directly. However, every other marked node retrieved needs to be decoded to obtain the actual maximal generic word corresponding to one or more non-marked nodes it represents. Before we describe the additional data structures that enable such decoding, we classify the non-marked answers as orphan and non-orphan outputs based on whether or not it has any marked node in its subtree as defined earlier. Let j^* be a marked node and j' be its lowest prime ancestor. A non-marked node that is an output is termed as orphan if it is not on the path from j' to j^*. Due to Lemma 6, every marked node retrieved from $\mathcal{I}_2$ leads to either a orphan output or non-orphan outputs or both. Below, we describe how to report all orphan and non-orphan outputs for a given query (P, τ) and a marked node j^*. We first append the index in Lemma 6 by data structure by Sadakane (Lemma 2) without affecting its space complexity to answer $count(.)$ query in constant time for any GST node.

Retrieving Non-orphan Outputs: To be able to report a non-orphan output of query (P, τ) for a given marked node (if it exists), we maintain a collection of bit vectors as follows:

- We associate a bit vector to each marked node i^* that encodes $count(.)$ information of the nodes on the (top-down) path from i' to i^*, i' being the (unique) lowest prime ancestor of i^*. Let $x_1, x_2, ..., x_r$ be the nodes on this path inclusive of both i' and i^*. Note that $r \leq g$, $\delta_i = count(x_{i-1}) - count(x_i) \geq 0$, and $count(x_1) - count(x_r) \leq 2g$ from the properties of the marking scheme. Now we maintain a bit vector $B_{i^*} = 10^{\delta_1}10^{\delta_2}...10^{\delta_r}$ along with constant time rank-select supporting structures at marked node i^*. Number of 0 in the bit-vector is $count(x_1) - count(x_2) + count(x_2) - count(x_3) + ... + count(x_{r-1}) - count(x_r) = count(x_1) - count(x_r)$. As length of the bit vector is bounded by $O(2g)$ and number of marked nodes is bounded by $O(n/g)$, total space required for these structures is $O(n)$ bits.
- Given a node i we would like to retrieve the node i_{max} i.e., child of node i with the highest $count(.)$ value in constant time. To enable such a lookup we maintain a bit vector $B = 10^{\delta_1}10^{\delta_2}...10^{\delta_n}$, where $child(i, \delta_i) = i_{max}$. If i is leaf then we assume $\delta_i = 0$. As each node contributes exactly one bit with

value 1 and at most one bit with value 0, length of the bit vector B is also bounded by $2g$ and subsequently occupies $O(n)$ bits.

Given a marked node j^* and a query (P, τ), we need to retrieve a parent-child node pair (u, v) on the path from j' to j^* such that $count(u) \geq \tau$ and $count(v) < \tau$. We can obtain the lowest prime ancestor j' of j^* in constant time to begin with [7]. Then to obtain a node u, we probe bit vector B_{j^*} by issuing a query $rank_1(select_0(count(j') - \tau))$. We note that these rank-select operations only returns the distance of the node u from j^* which can be then used along with level-ancestor query on j^* to obtain u. To verify if $path(u)$ indeed corresponds to a maximal generic word, we need to check if $count(.) \leq \tau$ for all the child nodes of u. To achieve this, we retrieve the $j_{max} = child(u, select_1(u+1) - select_1(u) - 1)$ and obtain its count value $count(j_{max})$ using a data structure by Sadakane (Lemma 2) in constant time. Finally, if $count(j_{max}) < \tau$ then u can be reported as an maximal generic word for (P, τ) query. If the input node j^* is highest marked descendant of locus node i_P then we need to verify if node u is within the subtree of i_P before it can be reported as an maximal generic word. Thus overall time spent per marked node to output the associated maximal generic word (if any) is $O(1)$. We note that unsuccessfully querying the marked node for non-orphan output does not hurt the overall objective of optimal query time since, such a marked node is guaranteed to generate orphan outputs (possibly except the highest marked ancestor of locus node i_P).

Retrieving Orphan Outputs: In this part we take advantage of the smaller size of the subtrees rooted at any maximal orphan node. We use bit encodings for every possible combination of the subtrees rooted at any maximal orphan node and table for storing all the answers for each possible subtree. Query procedure follows efficiently finding the encoding of the subtree and retrieving the answers from the table.

Instead of retrieving orphan outputs of the query based on the marked nodes as we did for non-orphan outputs, we retrieve them based on maximal orphan nodes by following two step query algorithm: (i) identify all maximal orphan nodes i in the subtree of the locus node i_P of P, with $count(i) \geq \tau$ and (ii) explore the subtree of each such i to find out the actual (orphan) outputs. If $count(i) \geq \tau$, then there exists at least on output in the subtree of i, otherwise there will not be any output in the subtree of i.

Since an exhaustive search in the subtree of a maximal orphan node is prohibitively expensive, we rely on the following insight to achieve the optimal query time. For a node i in the GST, let $subtree\text{-}size(i)$, $leaves\text{-}size(i)$ represents the number of nodes and number of leaves in the subtree rooted at node i respectively. The subtree of i can be then encoded (simple balanced parenthesis encoding) in $2subtree\text{-}size(i)$ bits. Also the $count(.)$ values of all nodes in the subtree of i in GST can be encoded in $2leaves\text{-}size(i)$ bits using the encoding scheme by Sadakane [11]. Therefore $2subtree\text{-}size(i) + 2leaves\text{-}size(i)$ bits are sufficient to encode the subtree of any node i along with the $count(.)$ information. Since $subtree\text{-}size(i) < 2leaves\text{-}size(i)$ and there are less than g leaves in

the subtree of a maximal orphan node, for a maximal orphan node i we have $2\mathit{subtree\text{-}size}(i) + 2\mathit{leaves\text{-}size}(i) < 6g = \frac{3}{4}\log n$. This implies the number of distinct maximal orphan nodes possible with respect to the above encoding is bounded by $\sum_{k=1}^{\frac{3}{4}\log n} 2^k = \Theta(n^{3/4})$.

To be able to efficiently execute the two step algorithm to retrieve all orphan outputs we maintain following components:

- A bit vector $B_{orph}[1...2n]$, where $B_{orph}[i] = 1$ iff node i is a maximal orphan node, along with constant time rank-select supporting structures over it occupying $O(n)$ bits space.
- Define an array $E[1...]$, where $E[i] = \mathit{count}(\mathit{select}_1(i))$ with *select* operation applied to a bit vector B_{orph} (i.e., the count of i-th maximal orphan node). Array E is not maintained explicitly, instead a $2n+o(n)$ bits RMQ structure over it is maintained.
- For each distinct encoding of a maximal orphan node out of total $\Theta(n^{3/4})$ of them, we shall maintain the list of top-g answers for $\tau = 1, 2, 3, ..., g$. Note that for each τ, number of answers is bounded by g. Overall space is therefore $O(n^{3/4}g^2) = o(n)$ bits.
- The total $n^{3/4}$ distinct encodings of maximal orphan nodes can be thought to be categorized into groups of size 2^k for $k = 1, ..., \frac{3}{4}\log n$. Encodings for all possible distinct maximal orphan nodes i having $k = 2\mathit{subtree\text{-}size}(i) + 2\mathit{leaves\text{-}size}(i)$ are grouped together and let L_k be this set. Then for a given maximal orphan node i in GST with $k = 2\mathit{subtree\text{-}size}(i) + \mathit{count}(i)$, we maintain a pointer so as to enable the lookup of all answers (at most g) corresponding to the encoding of subtree of i among all the encoding in set L_k. With number of bits required to represent such a pointer being proportional to the number leaves in the subtree of a maximal orphan node i i.e. $2\mathit{subtree\text{-}size}(i) + \mathit{count}(i)$, overall space can be bounded by $O(n)$ bits.

Query processing can now be handled as follows. Begin by identifying the x-th and y-th maximal orphan nodes, which are the first and last maximal orphan nodes in the subtree of the locus node i_P in $O(1)$ time as $x = 1 + \mathit{rank}_1(i_P - 1)$ and $y = \mathit{rank}_1(i'_P)$ using bit vector B_{orph}, where i'_P is the rightmost leaf of the subtree rooted at i_P. Then, all those $z \in [x, y]$ where $E[z] \geq \tau$ can be obtained in constant time per z using recursive range maximum queries on E as follows: obtain $z = \mathit{RMQ}_E(x, y)$, and if $E[z] < \tau$, then stop recursion, else recurse the queries on intervals $[x, z-1]$ and $[z+1, y]$. Recall that even if $E[z]$ is not maintained explicitly, it can be obtained in constant time using B_{orph} as $E[z] = \mathit{count}(\mathit{select}_1(z))$. Further, the maximal orphan node corresponding to each z can be obtained in constant time as $\mathit{select}_1(z)$. In conclusion, step (i) of query algorithm can be performed in optimal time. Finally, for each of these maximal orphan nodes we can find the list of pre-computed answers based on given τ and report them in optimal time. It can be noted that, for a given maximal orphan node i, we first obtain $\mathit{subtree\text{-}size}(i)$ and $\mathit{count}(i)$ in constant time using Lemma 1 and 2 respectively and than use the pointer stored as in index in the set L_k, with $k = \mathit{subtree\text{-}size}(i) + \mathit{count}(i)$.

Combining all pieces together we achieve the result summarized in following theorem.

Theorem 1. *The collection $\mathcal{D}=\{d_1, d_2, d_3, ..., d_D\}$ of strings of total n characters can be indexed in $(n \log \sigma + O(n))$ bits, such that given a query (P, τ), all maximal generic words can be reported in $O(p + \log \log n + output)$ time.*

4 Computing Minimal Discriminating Words

In the case of minimal discriminating words, given a query pattern P and a threshold τ, the objective is to find all nodes i in the subtree of locus node i_P such that $count(i) \leq \tau$ and $count(i_{parent}) > \tau$, where i_{parent} is the parent of a node i. Then each of these nodes represent a minimal discriminating word given by $path(i_{parent})$ concatenated with the first leading character on the edge connecting nodes i_{parent} and i. A linear space index with same query bounds as summarized in Lemma 5 can be obtained for minimal discriminating word queries by following the description in Section 3.1, except in this scenario, we map each node i in GST to a vertical segment $(i, [count(i), count(i_{parent}) + 1])$. Similarly, the succinct space solution can be obtained by following the same index framework as that of maximal generic words. Below we briefly describe the changes required in the index and query algorithm described in Section 3.2 so as to retrieve minimal discriminating words instead of maximal generic words.

We need to maintain all the components of index listed in the proof for Lemma 6 with a single modification. The set $\mathcal{I}_1$ consists of segments obtained by mapping each marked node i^* to a vertical segment $(i^*, [count(i^*), count(i^*_{parent}) + 1])$. By following the same arguments as before, we can rewrite the Lemma 6 as follows:

Lemma 7. *The collection $\mathcal{D}=\{d_1, d_2, d_3, ..., d_D\}$ of strings of total n characters can be indexed in $(n \log \sigma + O(n))$ bits, such that given a query (P, τ), we can identify all marked nodes j^* satisfying either of the following condition in $O(p + \log \log n + output)$ time.*

- *$path(j^*_{parent})$ appended with leading character on edge j^*_{parent}-j^* is a minimal discriminating word for input (P, τ)*
- *there is at least one node in $N(j^*) = GST(j'/j^*)$ that corresponds to desired minimal discriminating word, j' being the unique lowest prime ancestor of marked node j^**
- *j^* is a highest marked descendant of i_P*

We append the components required in the above lemma by data structure by Sadakane (Lemma 2) to answer $count(.)$ query in constant time for any GST node and retrieve the non-orphan, orphan outputs separately as before.

Though we maintain same collection of bit vectors as required for maximal generic words to retrieve non-orphan outputs, query processing differs slightly in this case. Let i^* be the input marked node and i' be its lowest prime ancestor.

Then we can obtain parent-child node pair (u, v) on the path from i' to i^* such that $count(u) > \tau$ and $count(v) \leq \tau$ in constant time. Node v can now be returned as an answer since concatenation of $path(u)$ with first character on edge u-v will correspond to a minimal discriminating word. Thus, every marked node obtained by segment intersection query on set $\mathcal{I}_2$ produces a non-orphan output in this case as opposed to the case of maximal generating words where it may or may not produce a non-orphan output. Also if the input node i^* is highest marked descendant of locus node i_P then we need to verify if node v is within the subtree of i_P before it can be reported as an output.

Data structures and query algorithm to retrieve the orphan outputs remain the same described earlier in Section 3.2. It is to be noted that top-g answers to be stored for $\tau = 1, 2, 3, ..., g$ corresponding to each of the distinct maximal orphan node encoding now corresponds to the minimal discriminating word.

Based on the above description, following theorem can be easily obtained.

Theorem 2. *The collection $\mathcal{D}$=$\{d_1, d_2, d_3, ..., d_D\}$ of strings of total n characters can be indexed in $(n \log \sigma + O(n))$ bits, such that given a query (P, τ), all minimal discriminating words can be reported in $O(p + \log \log n + output)$ time.*

5 Concluding Remarks

In this paper, we revisited the maximal generic word and minimal discriminating word problem and proposed a first succinct index for both the problems. It would be interesting to see if succinct index can be obtained for these problems achieving optimum query time.

References

1. Belazzougui, D., Navarro, G.: Alphabet-independent compressed text indexing. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 748–759. Springer, Heidelberg (2011)
2. Chan, T.M.: Persistent predecessor search and orthogonal point location on the word ram. In: SODA, pp. 1131–1145 (2011)
3. Fadiel, A., Lithwick, S., Ganji, G., Scherer, S.W.: Remarkable sequence signatures in archaeal genomes. Archaea 1(3), 185–190 (2003)
4. Fischer, J., Heun, V.: A New Succinct Representation of RMQ-Information and Improvements in the Enhanced Suffix Array. In: Chen, B., Paterson, M., Zhang, G. (eds.) ESCAPE 2007. LNCS, vol. 4614, pp. 459–470. Springer, Heidelberg (2007)
5. Fischer, J., Heun, V., Stühler, H.M.: Practical Entropy-Bounded Schemes for O(1)-Range Minimum Queries. In: IEEE DCC, pp. 272–281 (2008)
6. Gawrychowski, P., Kucherov, G., Nekrich, Y., Starikovskaya, T.: Minimal discriminating words problem revisited. In: Kurland, O., Lewenstein, M., Porat, E. (eds.) SPIRE 2013. LNCS, vol. 8214, pp. 129–140. Springer, Heidelberg (2013)
7. Hon, W.-K., Shah, R., Vitter, J.S.: Space-efficient framework for top-k string retrieval problems. In: FOCS, pp. 713–722 (2009)

8. Kucherov, G., Nekrich, Y., Starikovskaya, T.: Computing discriminating and generic words. In: Calderón-Benavides, L., González-Caro, C., Chávez, E., Ziviani, N. (eds.) SPIRE 2012. LNCS, vol. 7608, pp. 307–317. Springer, Heidelberg (2012)
9. McCreight, E.M.: A space-economical suffix tree construction algorithm. J. ACM 23(2), 262–272 (1976)
10. Raman, R., Raman, V., Rao, S.S.: Succinct Indexable Dictionaries with Applications to Encoding k-ary Trees and Multisets. In: ACM-SIAM SODA, pp. 233–242 (2002)
11. Sadakane, K.: Succinct data structures for flexible text retrieval systems. J. Discrete Algorithms 5(1), 12–22 (2007)
12. Sadakane, K., Navarro, G.: Fully-functional succinct trees. In: SODA, pp. 134–149 (2010)
13. Weiner, P.: Linear pattern matching algorithms. In: SWAT (FOCS), pp. 1–11 (1973)