2013 Data Compression Conference

Faster Compressed Top-k Document Retrieval ⋆

Wing-Kai Hon[1], Rahul Shah[2], Sharma V. Thankachan[2], and Jeffrey Scott Vitter[3]

[1] National Tsing Hua University, Taiwan. wkhon@cs.nthu.edu.tw
[2] Louisiana State University, USA. {rahul,thanks}@csc.lsu.edu
[3] The University of Kansas, USA. jsv@ku.edu

Abstract. Let $\mathcal{D} = \{d_1, d_2, ...d_D\}$ be a given collection of D string documents of total length n, our task is to index $\mathcal{D}$, such that whenever a pattern P (of length p) and an integer k come as a query, those k documents in which P appears the most number of times can be listed efficiently. In this paper, we propose a compressed index taking $2|CSA| + D\log\frac{n}{D} + O(D) + o(n)$ bits of space, which answers a query with $O(t_{sa}\log k\log^{\epsilon} n)$ per document report time. This improves the $O(t_{sa}\log k\log^{1+\epsilon} n)$ per document report time of the previously best-known index with (asymptotically) the same space requirements [Belazzougui and Navarro, SPIRE 2011]. Here, $|CSA|$ represents the size (in bits) of the compressed suffix array (CSA) of the text obtained by concatenating all documents in $\mathcal{D}$, and t_{sa} is the time for decoding a suffix array value using the CSA.

1 Introduction

Top-k query processing is an emerging field in Databases, and it is a crucial part of practical web search engines using inverted indexes, such as Google, Yahoo and Bing. In string retrieval settings, we are given a set $\mathcal{D} = \{d_1, d_2, ...d_D\}$ of D string documents of total length n, our task is to index $\mathcal{D}$, such that whenever a pattern P (of length p) and an integer k come as a query, those k documents where P appear most times can be listed efficiently. We call this the *top-k frequent document retrieval* problem.

Research on the document retrieval problem was started by Matias et. al. [17] and Muthukrishnan [18], and the top-k frequent document retrieval problem was introduced in [7]. The recent flurry of activities in this area [1, 2, 5, 8, 13, 19–21, 9, 10, 22] comes with the pioneer work by Hon et al. [11]. Along with a linear space and near-optimal query time index, they proposed the first compressed index taking $2|CSA| + D\log\frac{n}{D} + O(D) + o(n)$ bits of space with per document report time $O(t_{sa}\log^{3+\epsilon} n)$. Here $|CSA|$ represents the size (in bits) of the compressed suffix array of $T = d_1d_2d_3...d_D$[4], the text obtained by concatenating all documents in $\mathcal{D}$, and t_{sa} is the time for computing a suffix array value using the CSA. This

⋆ This work is supported in part by Taiwan NSC Grant 99-2221-E-007-123 (W. Hon) and US NSF Grant CCF–1017623 (R. Shah).

[4] The last character of each document is fixed as a special symbol \$, which does not appear anywhere else in T.

1068-0314/13 $26.00 © 2013 IEEE
DOI 10.1109/DCC.2013.42

Table 1. Indexes for Top-k Frequent Document Retrieval

Source	Index Space (in bits)	Time per reported document
[7]	$O(n \log n + n \log^2 D)$	$O(1)$
[11]	$O(n \log n)$	$O(\log k)$
[2]	$\|CSA\| + n \log D(1 + o(1))$	Unbounded
[5]	$\|CSA\| + O(\frac{n \log D}{\log \log D})$	$O(\log^{3+\epsilon} n)$
[1]	$\|CSA\| + O(\frac{n \log D}{\log \log D})$	$O(\log k \log^{2+\epsilon} n)$
[1]	$\|CSA\| + O(n \log \log \log D)$	$O(\log k \log^{2+\epsilon} n)$
[14]	$O(n \log \sigma + n \log D)$	$O(1)$
[5]	$\|CSA\| + n \log D + o(n)$	$O(\log^{2+\epsilon} n)$
[1]	$\|CSA\| + n \log D + o(n)$	$O(\log k \log^{1+\epsilon} n)$
[10]	$\|CSA\| + 2n \log D(1 + o(1))$	$O(\log \log n)$
[10]	$\|CSA\| + n \log D(1 + o(1))$	$O((\log \sigma \log \log n)^{1+\epsilon})$
[11]	$2\|CSA\| + D \log \frac{n}{D} + O(D) + o(n)$	$O(\log^{4+\epsilon} n)$
[1]	$2\|CSA\| + D \log \frac{n}{D} + O(D) + o(n)$	$O(\log k \log^{2+\epsilon} n)$
Ours	$2\|CSA\| + D \log \frac{n}{D} + O(D) + o(n)$	$O(\log k \log^{1+\epsilon} n)$

query time has been improved by Gagie et al. [5], and currently the best-known bound of $O(t_{sa} \log k \log^{1+\epsilon} n)$ is by Belazzougui and Navarro [1]. In this paper, the query time is further improved to $O(t_{sa} \log k \log^{\epsilon} n)$. See Table 1 for the space-time trade-offs of all known indexes, where we set t_{sa} to be $O(\log^{1+\epsilon} n)$ upon the assumption of using the space-optimal CSA of [4].

2 Basic Components

2.1 Bit Vectors with Rank/Select Support

Let $B[1...U]$ be a bit-vector of length U with m 1's. The function $rank_B(i)$ is defined as the number of 1s in $B[1...i]$, and $select_B(j)$ is defined as the position of the jth 1 in B. Then, B can be maintained in $m \log \frac{U}{m} + O(m) + o(U)$ bits and can support rank/select queries in constant time [23]. Another representation takes only $m \log \frac{U}{m} + O(m)$ bits, where only select can be supported in constant time.

Lemma 1 *[6] A given set of m distinct integers drawn from a set $\{1, 2, 3, ..., U\}$ can be encoded in $O(m \log(U/m))$ bits and decoded in $O(m)$ time.*

Proof. Define a bit-vector $B[1...U]$ such that, for each integer i to be stored, set $B[i] = 1$. All the other bits are set to 0. This can be maintained in $m \log \frac{U}{m} + O(m) = O(m \log(U/m))$ bits with constant-time select supported. Then, all m integers can be decoded in $O(m)$ time by performing $select_B(j)$ for $j = 1, 2, ..., m$. □

2.2 Succinct Representation of Ordinal Trees

Any ordered tree with m nodes can be maintained in $2m + o(m)$ bits, such that the following operations can be supported in constant time [12, 25] (node i refers to the node with pre-order rank i):

- *parent*(i) returns the parent of node i
- *lca*(i, j) returns the lowest common ancestor of nodes i and j
- *left-leaf(i)/right-leaf(i)* returns the the leftmost/rightmost leaf in the subtree rooted at node i

2.3 Computing Arbitrary Term Frequencies

Let $T = d_1 d_2 d_3 ... d_D$, the text (of length n) obtained by concatenating all documents in $\mathcal{D}$. The last character of each document is fixed as a special symbol \$, which does not appear anywhere else in T. We define a bit-vector $\mathcal{B}_\mathcal{D}[1...n]$, such that $\mathcal{B}_\mathcal{D}[i] = 1$ if and only if the character at position i in T is \$. We say a suffix $T[i...n]$ belongs to document d_r if $r = 1 + rank_{\mathcal{B}_\mathcal{D}}(i)$.

The document array $D_A[1...n]$ is defined as follows: $D_A[i] = r$, if the lexicographically ith smallest suffix of T belongs to document d_r. Now,

- *rank*(r, i) returns the number of occurrences of r in $D_A[1...i]$;
- *select*(r, j) is -1 if $j > |d_r|$, else i where $D_A[i] = r$ and $rank(r, i) = j$

Hon et. al. [11] showed that *rank*/*select* operations on D_A can be simulated using the following structures: **(i)** compressed suffix array CSA of T (of size $|CSA|$ bits), where $SA[\cdot]$ and $SA^{-1}[\cdot]$ represent the suffix array and inverse suffix array values in CSA; **(ii)** compressed suffix array CSA_r of document d_r (of size $|CSA_r|$ bits) corresponding to every $d_r \in \mathcal{D}$, where $SA_r[\cdot]$ and $SA_r^{-1}[\cdot]$ represent the suffix array and inverse suffix array values in CSA_r; and **(iii)** the bit-vector $\mathcal{B}_\mathcal{D}$ maintained in $D \log \frac{n}{D} + O(D) + o(n)$ bits with constant-time rank/select supported (refer to Section 2.1). Hence the total space is $\sum_{r=1}^{D} |CSA_r| + D \log \frac{n}{D} + O(D) + o(n) \leq |CSA| + D \log \frac{n}{D} + O(D) + o(n)$ bits in addition to the $|CSA|$ bits of CSA.

For computing *select*(r, j), we first compute the jth smallest suffix in CSA_r, and obtain the position *pos* of this suffix within document d_r, from which we can easily obtain the position *pos*$'$ of this suffix within T. After that, we compute $SA^{-1}[pos']$ in CSA as the desired answer for *select*(r, j). This takes $O(t_{sa})$ time. The $rank(r, i) = j$ can be obtained in $O(t_{sa} \log n)$ time using a binary search on j such that $select(r, j) \leq i < select(r, j + 1)$.

The time for computing *rank*(r, i) can be improved to $O(t_{sa} \log \log n)$ as follows: At every $(\log^2 n)$th leaf of each CSA_r, we explicitly maintain its corresponding

position in CSA and a predecessor structure over it [28]. The size of this additional structure is $o(n)$ bits. Now, when we perform the query, we can first query on this predecessor structure to get an approximate answer, and the exact answer can be obtained by performing binary search on a smaller range of only $\log^2 n$ leaves.

Now given the suffix range $[L, R]$ of a pattern P in CSA, the term frequency $tf(P, d_r)$, defined as the number of occurrences of P in document d_r, will be equal to the number of occurrences of d_r in $D_A[L...R]$. This can be computed as $rank(r, R) - rank(r, L-1)$ in $O(t_{sa} \log\log n)$ time.

Lemma 1. *Given the suffix range $[L, R]$ of a pattern P in CSA, the $tf(P, d)$ value corresponding to any document d can be computed in $O(t_{sa} \log\log n)$ time, using a data structure of size $|CSA| + D \log \frac{n}{D} + O(D) + o(n)$ bits in addition to the CSA.*

2.4 Marked Tree Structures

Let GST denote the suffix tree of T, and g be a parameter called grouping factor. We identify certain nodes in GST as marked nodes based on the following scheme: group every g consecutive leaves in the GST together (from left to right), mark the lowest common ancestor of the first and last leaf in each group. Further, mark the lowest common ancestor of all pairs of marked nodes. Then, we shall mark the leftmost and the rightmost leaves within the subtree rooted at every marked node. Hon et al. [11] observed the following properties:

1. The number of marked nodes by the end of this procedure is $O(n/g)$.
2. For any node u with at least $2g$ leaves in its subtree, there exists a unique highest marked descendent node u^*, such that the number of leaves in the subtree of u, but not in the subtree of u^*, is at most $2g$.

The following is a useful lemma.

Lemma 2 *Suppose that the suffix range $[L, R]$ of a pattern, and its locus node u in GST, are given. Then, the pre-order rank of u's highest marked descendent node u^* (if it exists), and the corresponding suffix range $[L^*, R^*]$, can be computed in $O(\log n)$ time using a data structure GST_g of size $O((n/g) \log g)$ bits.*

Proof. GST_g consists of the following components:

- a compact tree obtained by retaining only those nodes in GST which are marked. Then corresponding to every marked node in GST, there exists a unique node in this trie and vice versa. This takes $O(n/g)$ bits of space [11].
- a bit-vector $B_{no}[1...2n]$, where $B_{no}[i] = 1$ if the ith node in GST is marked, else 0. This can be maintained in $O((n/g) \log g)$ bits and can support select in constant time and rank in $O(\log n)$ time (refer to Section 2.1).

– a bit-vector $B_{le}[1...n]$, where $B_{le}[i] = 1$ if the ith leftmost leaf in GST is marked, else 0. This can be maintained in $O((n/g)\log g)$ bits and can support select in constant time and rank in $O(\log n)$ time (refer to Section 2.1).

The total space can be bounded by $O((n/g)\log g)$ bits. Now given a suffix range $[L, R]$, then

$$L^* = L \text{ (respectively } R^* = R) \text{ if } L \text{ (respectively } R) \text{ is marked.}$$

Otherwise, the L^*th leaf is the first marked leaf towards the right side of L, and the R^*th leaf is the last marked leaf towards the left side of R. These can be computed as follows:

$$L^* = select_{B_{le}}(1 + rank_{B_{le}}(L-1)) \text{ and } R^* = select_{B_{le}}(rank_{B_{le}}(R)).$$

Note that $L^* - L < g$ and $R - R^* < g$. Next, to locate u^* in the GST, we may find the lowest common ancestor (lca) of the L^*th leftmost leaf and the R^*th leftmost leaf. However, as GST is not stored explicitly, we shall find u^* in an indirect way. First, we identify the leaf nodes corresponding to the above two leaves in the compact tree. Precisely, they will be the $(1 + rank_{B_{le}}(L-1))$th and the $(rank_{B_{le}}(R))$th leftmost leaves in the compact tree. After that, we find their lowest common ancestor (say, with pre-order rank x) in the compact tree, which corresponds to u^* in GST. It follows that u^* is the xth marked node in GST, and its pre-order rank in GST is given by $select_{B_{no}}(x)$. The procedure takes $O(\log n)$ time as it involves only a constant number of rank/select operations and an lca operation. Notice that the locus node u is mentioned only for the sake of explanation, and it is never computed explicitly. □

3 The Compressed Index

In this section, we briefly describe a compressed index based on the earlier framework by Hon et. al. [11] and the main result is summarized as follows:

Theorem 1. *There exists an index of size* $2|CSA| + D\log\frac{n}{D} + O(D) + o(n)$ *bits for top-k frequent document retrieval with* $O(t_{sa}\log^{2+\epsilon} n)$ *time per reported document.*

A set $S_{cand} \subseteq \mathcal{D}$ is called a *candidate set* of a query, if it contains all those documents that are reported as the answer to the query. Therefore, once the candidate set is given, the top-k query can be answered by first finding the $tf(P, d)$ score of each document $d \in S_{cand}$, sort them in its decreasing order of the scores, and report the first k documents. Using the structure in Section 2.3, this can be performed in $|S_{cand}| \times O(t_{sa}\log\log n + \log|S_{cand}|) = O(|S_{cand}|\, t_{sa}\log\log n)$ time [5].

[5] We assume $t_{sa} = \Omega(\log n)$ as we shall use the space-optimal CSA of [4]

Lemma 3 *A top-k query can be answered in $O(|S_{cand}|\, t_{sa} \log\log n)$ time once the candidate set S_{cand} is identified.* □

The query time is directly proportional to the size $|S_{cand}|$ of the candidate list. Hence, our objective is to find a candidate set with size as small as possible.

3.1 Index for top-z queries for a fixed z

Firstly, we derive an index for answering top-z queries, where z is fixed in advance. Therefore, the query is now only a suffix range $[L, R]$. The index consists of **(i)** a compressed suffix array CSA of T; **(ii)** a data structure for computing arbitrary term frequencies (Sec 2.3); and **(iii)** an auxiliary structures specific to z as follows. By choosing a grouping factor $g = z(\log n)^{2+\epsilon}$, we identify those nodes in GST which are marked and maintain GST_g in $O((n/g)\log g) = o(n/\log n)$ bits (refer to Lemma 2). Let $F(u, z)$ be the set of top-z frequent documents for node u being the locus. Then $F(., z)$ is maintained explicitly (in $z \log D$ bits) for every marked node. The space required is $O((n/g)z \log D) = o(n/\log n)$ bits. Therefore, the total index space is $2|CSA| + D \log \frac{n}{D} + O(D) + o(n)$ bits.

Query Answering: Suppose that the suffix range $[L, R]$ with u being the locus node is given. Let u^* be u's highest marked descendent. The pre-order rank of u^* and its suffix range $[L^*, R^*]$ can be computed in $O(\log n)$ time (refer to Lemma 2). Hon et. al [11] showed that the union of the following two sets is a *candidate set* of size at most $2g + z$.

- top-z frequent documents within $D_A[L^*..R^*]$;
- the documents $D_A[i]$ for $i \in [L, L^* - 1] \cup [R^* + 1, R]$;

All documents in the first set are pre-computed and stored, and hence can be retrieved in $O(z)$ time. Each $D_A[.]$ value can be decoded in $O(t_{sa})$ time, therefore retrieval of all documents in the second set takes $O(gt_{sa})$ time. In summary, we obtain a candidate set of $O(g+z)$ documents in $O(gt_{sa}+z)$ time. Combining with Lemma 3, the top-z documents can be answered in another $O((g + z)t_{sa} \log\log n)$ time. By substituting $g = z(\log n)^{2+\epsilon}$ and absorbing the $\log\log n$ factor in $\log^{\epsilon} n$, the resulting query time will be $O(zt_{sa}(\log n)^{2+\epsilon})$.

3.2 Index for top-k queries for any given k

In order to support top-k queries, we maintain the (at most) $\log D$ auxiliary structures for $z = 1, 2, 4, 8, \ldots$. Since an auxiliary structure for a specific z is $o(n/\log n)$ bits, the overall increase in total space is bounded by $o(n)$ bits. Now, a top-k query

can be answered by choosing $z = 2^{\lceil \log_2 k \rceil}$ and retrieve the top-z documents by querying on the substructures specific to z. Since $k = \Theta(z)$, the resulting query time will be $O(kt_{sa} \log^{2+\epsilon} n)$. This completes the proof of Theorem 1.

4 Faster Compressed Index

This section is dedicated to the description of our index with improved query time. The idea is to choose a smaller grouping factor, thereby reducing the size of candidate set. However, this will result in higher number of marked nodes, and the explicit storage of pre-computed answers (with $\log D$ bits per entry) at these marked nodes will lead to a non-succinct solution. Our key contribution is to show how these pre-computed lists can be encoded in $O(\log \log n)$ bits per entry (thereby achieving the desired space). Our main result is summarized as follows.

Theorem 2. *There exist an* $2|CSA| + D \log \frac{n}{D} + O(D) + o(n)$ *bits index for top-k frequent document retrieval with* $O(t_{sa} \log k \log^{\epsilon} n)$ *time per reported document.*

Similar to the index in Section 3, we have **(i)** a compressed suffix array CSA of T; **(ii)** a data structure for computing arbitrary term frequencies (Sec 2.3); and **(iii)** an auxiliary structures specific to z for $z = 1, 2, 4, 8, \ldots$. Note that **(i)** and **(ii)** are common for all auxiliary structures. Therefore, we turn our attention only to **(iii)** and show that its space is bounded by $o(n)$ bits.

4.1 Auxiliary structure for a fixed z

Here, we mark the nodes in GST based on two grouping factors g and h, where $g = z(\log n)^{2+\epsilon}$ and $h = z \log z \log^{\epsilon} n$. Then, we maintain the corresponding GST_g and GST_h in a total of $O((n/g) \log g + (n/h) \log h) = o(n/z)$ bits. In order to distinguish the marked nodes based of these two different grouping factors, we shall use the following terminologies: If a node is marked as per the grouping factor g, we shall call it as a marked node. Otherwise, if a node is marked as per the grouping factor h only, we shall call it as a prime node.

Let u be a node in GST with u' (resp., u^*) being its highest prime (marked) descendent (if it exists). Let $[L, R]$, $[L', R']$ and $[L^*, R^*]$ be the ranges of leaves within the subtree of u, u^* and u', respectively. The following inequalities hold:

1. $L \leq L' \leq L^* < R^* \leq R' \leq R$;
2. $L' - L < h$, and $R - R' < h$;
3. $L^* - L' < g$, and $R' - R^* < g$.

Then, if node u is the locus of a query, the union of the following sets is a candidate set (say S^h_{cand}) of size at most $2h + z$:

- top-z frequent documents within $D_A[L'...R']$ and
- the documents $D_A[i]$ for $i \in [L, L' - 1] \cup [R' + 1, R]$

Once S^h_{cand} is given, it takes only $O((h + z)t_{sa} \log \log n) = O(z\, t_{sa} \log z \log^{\epsilon} n)$ additional time[6] for answering a top-z query (using Lemma 3). Note that documents corresponding to $D_A[i]$ for $i \in [L, L' - 1] \cup [R' + 1, R]$ are computed on the fly in $O(h\, t_{sa})$ time and it will not affect the overall time complexity. The only remaining thing is to show how to obtain the top-z frequent documents within $D_A[L'..R']$. The idea is to store the *encoded form* of the top-z documents corresponding to every prime node. Using the result in the following lemma, the total query time can be bounded by $O(z\, t_{sa} \log z \log^{\epsilon} n)$.

Lemma 4 *Top-z documents corresponding to every prime node as the locus can be encoded in $O(n/(\log z \log^{\epsilon} n)) + o(n/\log n)$ bits, such that documents corresponding to any given prime node u' can be decoded in $O(z\, t_{sa} \log \log n)$ time.*

Proof. Consider the candidate set corresponding to a prime node u' as the locus, which includes

- the top-z frequent documents within $D_A[L^*..R^*]$, which can be obtained in $O(z)$ time since the top-z documents corresponding to every marked node are maintained explicitly in $O((n/g)z \log D) = o(n/\log n)$ bits.
- the documents $D_A[i]$ for $i \in [L', L^* - 1] \cup [R^* + 1, R']$. However we cannot afford to include all $O(g)$ documents into the candidate set. Instead, we select only at most z elements within this category which still form a desired candidate list of only $O(z)$ elements. Note that even though we have $O(g)$ documents in this category, only $\leq z$ can be in the output. For each output document d_j in this category, it can be associated with an integer $i \in [L', L^* - 1] \cup [R^* + 1, R']$, such that $D_A[i] = d_j$. If we replace each such i by its relative position in $[L', L^* - 1] \cup [R^* + 1, R']$, this problem can be rephrased as the encoding of z distinct integers drawn from a set $\{1, 2, 3, ..., 2g\}$ and it takes only $O(z \log(g/z)) = O(z \log \log n)$ bits (refer to Lemma 1). The space consumption is $O((n/h)z \log \log n) = O(n/(\log z \log^{\epsilon} n))$. Thus, given such a relative position, the corresponding document can be retrieved in $O(t_{sa})$ time. Now, if a documents $d_j$which is an output, but does not belong to this category, it will definitely be in the list of top-z frequent documents within $D_A[L^*..R^*]$, and hence will be retrieved from the previous category.

[6] The $\log \log n$ factor is absorbed by $\log^{\epsilon} n$.

Therefore, a candidate list of $O(z)$ elements corresponding to any given prime node u' as the locus can be computed in $O(z\, t_{sa})$ time using an $O(n/(\log z \log^{\epsilon} n))+ o(n/\log n)$ bits structure. Consequently, the top-z documents for u' can be computed in $O(z\, t_{sa} \log\log n)$ time using Lemma 3.

Putting everything altogether, we have the following lemma.

Lemma 5 *The auxiliary structure for a specific z takes $O(n/(\log z \log^{\epsilon} n)) + o(n/\log n) + o(n/z)$ bits and a top-z query can be answered in $O(z\, t_{sa} \log z \log^{\epsilon} n)$ time.* □

4.2 Handling top-k queries for any given k

In order to support top-k queries, we maintain the (at most) $\log D$ auxiliary structures for $z = 1, 2, 4, 8, \ldots$ as before. This requires a total of:

$$O(\log^{-\epsilon} n \sum_{z=1,2,4\ldots} n/\log z) + o(\sum_{z=1,2,4\ldots} n/\log n) + o(\sum_{z=1,2,4\ldots} n/z) = o(n)\ bits.$$

Now, a top-k query can be answered by choosing $z = 2^{\lceil \log_2 k \rceil}$ and retrieve the top-z documents by querying on the substructures specific to z. Since $k = \Theta(z)$, the resulting query time will be $O(k\, t_{sa} \log k \log^{\epsilon} n)$. This completes the proof of Theorem 2. □

5 Concluding Remarks

In this paper, we have shown how to index the documents in roughly $2 \times |CSA|$ bits of space, which achieves per-document reporting time for the top-k document retrieval problem in $O(\log k \log^{1+\epsilon} n)$ time. An interesting open question is whether we can further reduce the space to roughly $|CSA|$ bits, while maintaining the same reporting bound. Also, it will also be interesting to test the practical performance of our proposed index, and compare it with the series of results developed in the recent years.

References

1. D. Belazzougui and G. Navarro. Improved Compressed Indexes for Full-Text Document Retrieval. In *SPIRE*, pages 386 – 397 2011.
2. S. Culpepper, G. Navarro, S. Puglisi, and A. Turpin. Top-k Ranked Document Search in General Text Databases. In *ESA*, pages 194–205, 2010.
3. P. Ferragina and G. Manzini. Indexing Compressed Text. *Journal of the ACM*, 52(4):552–581, 2005.

4. P. Ferragina, G. Manzini, Veli Mäkinen, and G. Navarro, Compressed Representations of Sequences and Full-Text Indexes, *ACM Transactions on Algorithms*, 3(2), 2007.
5. T. Gagie, G. Navarro, and S. J. Puglisi. Colored Range Queries and Document Retrieval. In *SPIRE*, pages 67–81, 2010.
6. R. Grossi and J. S. Vitter. Compressed Suffix Arrays and Suffix Trees with Applications to Text Indexing and String Matching. *SIAM Journal on Computing*, 35(2):378–407, 2005.
7. W. K. Hon, M. Patil, R. Shah, and S.-B. Wu. Efficient Index for Retrieving Top-k Most Frequent Documents. *Journal of Discrete Algorithms*, 8(4):402–417, 2010.
8. W. K. Hon, R. Shah, S. V. Thankachan, and J. S. Vitter. String Retrieval for Multi-pattern Queries. In *SPIRE*, pages 55–66, 2010.
9. W. K. Hon, R. Shah, S. V. Thankachan, and J. S. Vitter. Document Listing for Queries with Excluded Pattern, In *CPM*, pages 185–195, 2012.
10. W. K. Hon, R. Shah, and S. V. Thankachan. Towards an Optimal Space-and-Query-Time Index for Top-k Document Retrieval, In *CPM*, pages 173–184, 2012.
11. W. K. Hon, R. Shah, and J. S. Vitter. Space-Efficient Framework for Top-k String Retrieval Problems. In *FOCS*, pages 713–722, 2009.
12. J. Jansson, K. Sadakane, and W. K. Sung. Ultra-succinct Representation of Ordered Trees. In *SODA*, pages 575–584, 2007.
13. M. Karpinski and Y. Nekrich. Top-k Color Queries for Document Retrieval. In *SODA*, pages 401–411, 2011.
14. G. Navarro and Y. Nekrich. Top-k document retrieval in optimal time and linear space In *SODA*, pages 1066-1077, 2012.
15. V. Mäkinen and G. Navarro. Compressed Full-Text Indexes. *ACM Computing Surveys*, 39(1), 2007.
16. U. Manber and G. Myers. Suffix Arrays: A New Method for On-Line String Searches. *SIAM Journal on Computing*, 22(5): 935–948, 1993.
17. Y. Matias, S. Muthukrishnan, S. C. Sahinalp, and J. Ziv. Augmenting Suffix Trees, with Applications. In *ESA*, pages 67–78, 1998.
18. S. Muthukrishnan. Efficient Algorithms for Document Retrieval Problems, In *SODA*, pages 657–666, 2002.
19. G. Navarro, S. J. Puglisi, and D. Valenzuela. Practical Compressed Document Retrieval. In *SEA*, pages 193–205, 2011.
20. G. Navarro and S. J. Puglisi. Dual-Sorted Inverted Lists. In *SPIRE*, pages 309–321, 2010.
21. M. Patil, S. V. Thankachan, R. Shah, W. K. Hon, J. S. Vitter, and S. Chandrasekaran. Inverted Indexes for Phrases and Strings. *SIGIR*, 2011.
22. R. Shah, C. Sheng, S. V. Thankachan, J. S. Vitter. On Optimal Top-K String Retrieval. In CoRR abs/1207.2632 (2012)
23. R. Raman, V. Raman, and S. Rao. Succinct Indexable Dictionaries with Applications to Encoding k-ary Trees, Prefix Sums and Multisets. *ACM Transactions on Algorithms*, 3(4), 2007.
24. K. Sadakane. Succinct Data Structures for Flexible Text Retrieval Systems. *Journal of Discrete Algorithms*, 5(1):12–22, 2007.
25. K. Sadakane and G. Navarro. Fully-Functional Succinct Trees. In *SODA*, pages 134–149, 2010.
26. N. Välimäki and V. Mäkinen. Space-Efficient Algorithms for Document Retrieval. In *CPM*, pages 205-215, 2007.
27. P. Weiner. Linear Pattern Matching Algorithms. In *SWAT*, 1973.
28. D. E. Willard. Log-logarithmic Worst-Case Range Queries Are Possible in Space $\Theta(N)$. *Information Processing Letters*, 17(2):81–84, 1983.