Top-k document retrieval in optimal space

Dekel Tsur*

Abstract

We present an index for top-k most frequent document retrieval whose space is $|\text{CSA}|+o(n)+D\log\frac{n}{D}+O(D)$ bits, and its query time is $O(\log k \log^{2+\epsilon} n)$ per reported document, where D is the number of documents, n is the sum of lengths of the documents, and $|\text{CSA}|$ is the space of the compressed suffix array for the documents. This improves over previous results for this problem, whose space complexities are $|\text{CSA}| + \omega(n)$ or $2|\text{CSA}| + \omega(1)$.

Keywords data structures, document retrieval, text indexing.

1 Introduction

In document retrieval problems, the goal is to construct an index for a set of documents (strings) that can answer queries on the documents, for example, "which documents contain a given query string P?" or "how many documents contain P?". Matias et al. [14] were the first to study document retrieval problems, and afterward these problems were widely studied, e.g. [4–6, 11–13, 15, 19, 21].

In this paper we consider the *top-k most frequent document retrieval problem.* The goal in this problem is to build an index for a set $\mathcal{D}$ of documents that supports the following queries: given a string P and an integer k, find the k documents in $\mathcal{D}$ in which P occurs the most number of times. The theoretical results on this problem are summarized in Table 1. The paper of Hon et al. [13] was the first to give a succinct index for this problem. Additional succinct indices were given in [3, 7, 10]. These succinct indices use a compressed suffix array (cf. [16]) in order to store the concatenation of the documents in $\mathcal{D}$, and a rank-select structure holding the lengths of the documents. These two structures use $|\text{CSA}| + o(n) + D\log\frac{n}{D} + O(D)$ bits, where D is the number of documents, n is the sum of lengths of the documents, and $|\text{CSA}|$ is the space of the compressed suffix array. In this paper, we show that only an additional $o(n)$ bits are required for the index. More precisely, we give an index

*Department of Computer Science, Ben-Gurion University of the Negev. Email: dekelts@cs.bgu.ac.il

Source	Space	Time per reported document
[9]	$O(n(\log n + \log^2 D))$	$O(1)$
[13]	$O(n \log n)$	$O(\log k)$
[17]	$O(n(\log \sigma + \log D + \log\log n))$	$O(1)$
[13]	$2\lvert\text{CSA}\rvert + o(n) + D \log \frac{n}{D} + O(D)$	$O(\log^{4+\epsilon} n)$
[7]	$2\lvert\text{CSA}\rvert + o(n) + D \log \frac{n}{D} + O(D)$	$O(\log^{4+\epsilon} n)$
[3]	$2\lvert\text{CSA}\rvert + o(n) + D \log \frac{n}{D} + O(D)$	$O(\log k \log^{2+\epsilon} n)$
[7]	$\lvert\text{CSA}\rvert + O(n \log D / \log\log D)$	$O(\log^{3+\epsilon} n)$
[3]	$\lvert\text{CSA}\rvert + O(n \log D / \log\log D)$	$O(\log k \log^{2+\epsilon} n)$
[10]	$\lvert\text{CSA}\rvert + 2n \log D + o(n \log D)$	$O(\log\log n)$
[10]	$\lvert\text{CSA}\rvert + n \log D + o(n \log D)$	$O((\log \sigma \log\log n)^{1+\epsilon})$
[7]	$\lvert\text{CSA}\rvert + n \log D + o(n)$	$O(\log^{2+\epsilon} n)$
[3]	$\lvert\text{CSA}\rvert + n \log D + o(n)$	$O(\log k \log^{1+\epsilon} n)$
[3]	$\lvert\text{CSA}\rvert + O(n \log\log\log D)$	$O(\log k \log^{2+\epsilon} n)$
This paper	$\lvert\text{CSA}\rvert + o(n) + D \log \frac{n}{D} + O(D)$	$O(\log k \log^{2+\epsilon} n)$

Table 1: Results for top-k document retrieval. D is the number of documents, n is the sum of lengths of the documents, and $|\text{CSA}|$ is the space of a compressed suffix array holding the concatenation of the documents. The time complexities are per reported document, so the overall query time is k times the given complexity, plus the time to searching the compressed suffix array. The time complexities are in simplified form, using the assumption that the time for accessing a value of the suffix array is $t_{\text{SA}} = O(\log^{1+\epsilon} n)$.

whose space is $|\text{CSA}| + o(n) + D \log \frac{n}{D} + O(D)$ bits, and whose query time is $O(t_{\text{search}} + k \log k (t_{\text{SA}} + \log\log k + \log\log\log n) \log n (\log\log n)^4)$, where t_{search} is the time for searching the compressed suffix array, and t_{SA} is the time for accessing a value of the suffix array. The space complexity of our index is better than previous results (see Table 1). The query time is poly-logarithmic in n as previous succinct solutions for the problem, with the exception of the index of Hon et al. [10] that uses substantially more memory than our index. Our result is based on an index of Belazzougui and Navarro [3]. The index of [3] stores pre-computed frequencies and minimal perfect hash functions in order to compute the frequencies in which the pattern P occurs in some candidate documents. We show that this task can be achieved without the use of minimal perfect hash functions, thus reducing the space of the index.

2 Preliminaries

An *interval* $[L, R]$ is a set of integers $\{L, L+1, \ldots, R\}$. For a string S, the *suffix array* SA_S of S is an array in which $\text{SA}_S[i] = j$ if $S[j..|S|]$ is the i-th smallest suffix of S in the lexicographical order of the suffixes. The *suffix range* of a string P with respect to S is the interval $[L, R]$ such that P is a prefix of $S[\text{SA}_S[i]..|S|]$ if and only if $i \in [L, R]$.

Our index uses several succinct data-structures. A *compressed suffix array* of a string S is a data-structure that supports the following operations: (1) computation of the suffix range of a string P w.r.t. S in time t_{search}, and (2) computation of $\text{SA}_S[i]$ in time t_{SA}. See [1,2,16] for a survey and recent results on compressed suffix arrays. A *rank-select* structure stores a binary vector (bitmap) B, and allows querying for the number of ones in B in positions $1, \ldots, i$, or reporting the position of the i-th one. A *succinct tree* structure stores a rooted ordered tree, and support queries on the structure of the tree, for example, finding the lowest common ancestor of two nodes.

Let E be an array of integers, which will be called *colors*. For an interval $I = [L, R]$ of E, define $\text{top}_{k,E}(I)$ to be the set of the k most frequent colors in I, where ties are broken according to the color values (if there is a tie between colors c and c', where $c < c'$, then c is chosen for the set). Define $\text{freq}_E(c, I)$ to the frequency of color c in I.

A *nested interval sequence* is a sequence of intervals $(I_0, \ldots, I_s)$ such that for every two consecutive intervals $I_t = [L_t, R_t]$ and I_{t+1}, either $I_{t+1} = [L_t - 1, R_t]$ or $I_{t+1} = [L_t, R_t + 1]$. If $I_{t+1} = [L_t - 1, R_t]$ for all t then the sequence is called *left-nested interval sequence*.

Let $\mathcal{I} = (I_0, \ldots, I_s)$ be a nested interval sequence. Define $\text{top}_{k,E,\mathcal{I}}(0) = \text{top}_{k,E}(I_0)$, and define $\text{top}_{k,E,\mathcal{I}}(t)$ to be the set of the k most frequent colors in the interval I_t of E, where ties are broken with preference to colors appearing in $\text{top}_{k,E,\mathcal{I}}(t-1)$. More precisely, let i be the single element of $I_t \setminus I_{t-1}$. If the color $c = E[i]$ is in $\text{top}_{k,E,\mathcal{I}}(t-1)$ then $\text{top}_{k,E,\mathcal{I}}(t) = \text{top}_{k,E,\mathcal{I}}(t-1)$. Otherwise, let c' be the color in $\text{top}_{k,E,\mathcal{I}}(t-1)$ with smallest frequency in the interval I_{t-1} of E, where ties are broken according the color values. If c occurs in the interval I_t of E more times than c' occurs in this interval, then $\text{top}_{k,E,\mathcal{I}}(t) = \text{top}_{k,E,\mathcal{I}}(t-1) \setminus \{c'\} \cup \{c\}$. Otherwise, $\text{top}_{k,E,\mathcal{I}}(t) = \text{top}_{k,E,\mathcal{I}}(t-1)$. We also define sequences $\text{col}_{k,E,\mathcal{I}}$ and $\text{ind}_{k,E,\mathcal{I}}$ as follows. Start with both of these sequences empty and $t = 0$, and repeatedly increment the value of t. Whenever $\text{top}_{k,E,\mathcal{I}}(t-1) \neq \text{top}_{k,E,\mathcal{I}}(t)$, let i be the single element of $I_t \setminus I_{t-1}$, and append $E[i]$ to $\text{col}_{k,E,\mathcal{I}}$ and t to $\text{ind}_{k,E,\mathcal{I}}$.

We use the following lemma of Belazzougui and Navarro [3].

Lemma 1. *Let $\mathcal{I}$ be a left-nested interval sequence with s intervals. Then, $|\text{col}_{k,E,\mathcal{I}}| = O(\sqrt{sk})$.*

Corollary 2. *Let $\mathcal{I}$ be a nested interval sequence with s intervals. Then, $|\text{col}_{k,E,\mathcal{I}}| = O(\sqrt{sk})$.*

Proof. Denote $\mathcal{I} = (I_0, \dots, I_{s-1})$ and $I_0 = [L, R]$. Build a sequence E' in which $E'[L, R] = E[L, R]$, and for every $t > 0$, $E'[L - t] = E[i_t]$, where i_t is the single element of $I_t \setminus I_{t-1}$. Let $\mathcal{I}' = (I'_0, \dots, I'_{s-1})$, where $I'_t = [L - t, R]$. By definition, $\mathrm{col}_{k,E',\mathcal{I}'} = \mathrm{col}_{k,E,\mathcal{I}}$, and the corollary follows from Lemma 1. ■

3 The index

Let S be the concatenation of the documents in $\mathcal{D}$, and let T be the suffix tree of S. The leaves of T, from left to right, will be denoted $u_1, u_2, \dots, u_n$. The *leaf range* of a node v in T is the interval $[L_v, R_v]$ such that u_i is a descendant of v if and only if $i \in [L_v, R_v]$. For an integer b, the *sampled suffix tree* T_b is obtained from T by taking the subtree of T induced by the leaves $u_b, u_{2b}, u_{3b}, \dots$ and their ancestors, and removing all nodes with only one child (for every removed node, its parent and its child are connected by an edge). The sampled suffix tree T_b has at most $2n/b$ nodes.

The *document array* of $\mathcal{D}$ is an array of colors $E[1..n]$ such that $E[i] = j$ if the character $\mathrm{SA}_S[i]$ of S is a character of d_j. Let B be a bitmap of length n satisfying $B[i] = 1$ if and only if the character $S[i]$ is the first character of some document of $\mathcal{D}$.

The index stores the following structures. (1) A compressed suffix array for S. (2) A succinct rank-select data-structure over the bitmap B. (3) A succinct representation of the tree T. (4) For every $k \leq \min(D, \frac{1}{2}n/\log^2 n(\log\log n)^4)$ which is a power of 2, a succinct representation of the tree T_{kl_k}, where $l_k = \log k \log n(\log\log n)^4$. For every node in a tree T_{kl_k} (except the root), additional information is stored as described below.

Let v be some node in tree T_{kl_k}, and v' be the parent of v in T_{kl_k}. Let $v_0 = v, v_1, \dots, v_s, v'$ be the path from v to v' in T. Let $I'_0, \dots, I'_s$ be the leaf ranges of $v_0, \dots, v_s$, and denote $I'_t = [L_t, R_t]$. Note that $I'_0 \subset I'_1 \subset \cdots \subset I'_s$, $L_0 - L_s < kl_k$, and $R_s - R_0 < kl_k$. Construct a nested interval sequence $\mathcal{I}_v$ of $(I'_1, \dots, I'_s)$ as follows. Start with a sequence containing the interval I'_0. Then, append to $\mathcal{I}_v$ the interval I'_1 with intermediate intervals:

$$[L_0, R_0 + 1], [L_0, R_0 + 2], \dots, [L_0, R_1], [L_0 - 1, R_1], [L_0 - 2, R_1], \dots, [L_1, R_1] = I'_1.$$

Continue this process with the intervals $I'_2, \dots, I'_s$. Clearly, the number of intervals in $\mathcal{I}_v$ is at most $2kl_k$.

We use the following definitions and observations from [3].

Observation 3. *For every $c \in \mathrm{col}_{k,E,\mathcal{I}_v}$, $f - kl_k + 1 \leq \mathrm{freq}_E(c, [L_v, R_v]) \leq f$, where f is the minimum frequency of a color of $\mathrm{top}_{k,E}([L_v, R_v])$ in the interval $[L_v, R_v]$ of E.*

As in [3], we divide the colors of $\mathrm{col}_{k,E,\mathcal{I}_v}$ into two types. A color c is a *type 1* color if the frequency $\mathrm{freq}_E(c, [L_v, R_v])$ is between $f - l_k + 1$ and f, and a *type 2* color if the frequency is between $f - kl_k + 1$ and $f - l_k$.

Observation 4. *There are $O(k\sqrt{l_k})$ colors of type 1, and $O(k)$ colors of type 2.*

The first part of Observation 4 follows from Lemma 1, and the second part follows from the fact that a type 2 color must appear at least l_k times in the region $I'_s \setminus I'_0$ of E.

For the node v, the index stores the following information. (1) An array A_v containing pairs $(c, \mathrm{freq}_E(c, [L_v, R_v]))$ for every color $c \in \mathrm{top}_{k,E}([L_v, R_v])$, sorted according to colors. (2) For $j = 1, 2$, an array $A_{v,j}$ containing the values $\mathrm{freq}_E(c, [L_v, R_v])$ for every $c \in \mathrm{col}_{k,E,\mathcal{I}_v}$ of type j, sorted according to the order of these colors in $\mathrm{col}_{k,E,\mathcal{I}_v}$. (3) The sequence $\mathrm{ind}_{k,E,\mathcal{I}_v}$. (4) An array B_v holding the types of the colors in $\mathrm{col}_{k,E,\mathcal{I}_v}$. Note that the colors of $\mathrm{col}_{k,E,\mathcal{I}_v}$ are not stored as it would take too much space.

Answering queries Given a query P, k, let k' be the smallest power of 2 which is larger than k. Let $b = k' l_{k'}$. The suffix range $[L, R]$ of P w.r.t. T is equal to the leaf range of some node w of T. Let v be the highest descendant of w which is also a node of T_b. To answer the query, we will use the information stored for the copy of v in T_b. This information gives the frequencies of relevant colors in the interval $[L_v, R_v]$ of E. We will then scan the region $[L, R] \setminus [L_v, R_v]$ of E and update the frequencies of these colors. In order to decode the stored information, we need to construct the sequence $\mathcal{I}_v$ at query time, and this is done using the succinct representation of T.

In more detail, answering a query is performed as follows. (1) Using the compressed suffix array, find the suffix range $[L, R]$ of P. (2) If $k > \frac{1}{2}n/\log^2 n(\log\log n)^4$ or $R - L \le b$, scan the interval $[L, R]$ and compute the frequencies of the colors in this interval. Return the k most frequent colors. (3) Using the succinct representation of T_b, find the lowest common ancestor of $u_{\lceil L/b \rceil}$ and $u_{\lfloor R/b \rfloor}$ in T_b, which will be denoted by v. (4) Using the succinct representation of T_b, find the leaf ranges of the ancestors of v in T, from the parent of v to v', where v' is the parent of v in T_b. (5) Build the sequence $\mathcal{I}_v$ from the leaf ranges of the ancestors of v. Denote $\mathcal{I}_v = (I_0, \ldots, I_s)$, and let s' be the index such that $I_{s'} = [L, R]$. (6) Create a list of *candidates* that consists of (c, f) pairs, where c is a color and f is an integer. The candidates list is initialized with the elements of A_v. (7) Initialize indices p, p_1, p_2 to 1. (8) For every t from 1 to s', perform the following steps. (a) Let i be the single element of $I_t \setminus I_{t-1}$. (b) Using the compressed suffix array and the rank-select structure, compute the color $c = E[i]$. (c) If $t = \mathrm{ind}_{k',E,\mathcal{I}_v}[p]$, let $j = B_v[p]$. Add the pair $(c, A_{v,j}[p_j] + 1)$ to the candidates list, and increase p and p_j by one. (d) Otherwise, add $(c, 1)$ to the candidates list. (9) Sort the candidates list according to the colors. (10) Scan the candidates list and for each color c that appears in the list, compute the sum of the second coordinate of the pairs whose first coordinate is c. (11) Return the k most frequent candidates.

Time complexity The time complexity is determined by steps 1, 8, and 9. Step 1 takes t_{search} time, step 8 takes $O(s' \cdot t_{\text{SA}})$ time, and Step 9 takes $O((k'+s')\log\log(k'+s'))$ time using the sorting algorithm of Han [8]. Recall that $s' \leq 2k'l_{k'} = O(k\log k \cdot \log n(\log\log n)^4)$. If step 2 is performed, the time complexity of this step is $O((R-L)t_{\text{SA}})$, and in both cases $R-L = O(k'l_{k'})$. The following lemma follows.

Lemma 5. *A query takes* $O(t_{\text{search}}+k\log k(t_{\text{SA}}+\log\log k+\log\log\log n)\log n(\log\log n)^4)$ *time.*

Space complexity Since B has exactly D ones, the rank-select structure on B requires $D\log\frac{n}{D}+O(D)+o(n)$ bits [18]. The succinct representation of T requires $2n+o(n)$ bits [20]. Storing the tree T_{kl_k} takes $O(n/(kl_k))$ bits. For each node in T_{kl_k}, the following space is used: $O(k\log n)$ bits for storing the colors of $\text{top}_{k,E}([L_v,R_v])$ and their frequencies, $O(k\sqrt{l_k}\log l_k + k\log(kl_k)) = O(k\sqrt{l_k}\log\log n + k\log n)$ bits for storing the frequencies of colors in $\text{col}_{k,E,\mathcal{I}_v}$ (due to Corollary 2 and Observation 3), $O(k\sqrt{l_k}\log\frac{kl_k}{k\sqrt{l_k}}) = O(k\sqrt{l_k}\log\log n)$ bits for storing $\text{ind}_{k,E,\mathcal{I}_v}$ (as $\text{ind}_{k,E,\mathcal{I}_v}$ is a monotone increasing sequence of length $O(k\sqrt{l_k})$ with elements from $\{1,\ldots,2kl_k\}$), and $O(k\sqrt{l_k})$ bits for storing B_v. The total space over all nodes in all the trees T_{kl_k} is

$$O\left(\sum_{k=1,2,4,\ldots}\frac{n}{kl_k}(k\log n + k\sqrt{l_k}\log\log n)\right) = o(n).$$

Lemma 6. *The index uses* $|\text{CSA}|+2n+o(n)+D\log\frac{n}{D}+O(D)$ *bits.*

Reducing the space complexity The succinct representation of T requires $2n+o(n)$ bits. To reduce this space, store a succinct representation of T_g instead of T, where $g=\log\log n$. We assume that g divides l_k for all k (we can define $l_k = g\lceil(\log k\log n(\log\log n)^4)/g\rceil$ in order to ensure that this assumption is met). The tree T_g does not allow us to compute the sequence $\mathcal{I}_v$ which is needed to decode the information stored for v when answering a query. Therefore, we need to replace $\mathcal{I}_v$ with a nested interval sequence that satisfies the following: (1) The sequence can be efficiently constructed using T_g. (2) Let v' be the parent of v in T_{kl_k}, and let $v_0=v,v_1,\ldots,v_s,v'$ be the path from v to v' in T_g (both v and v' are in T_g due to the assumption that g divides kl_k). Then, for every i, the leaf range of v_i appears in the sequence. The second requirement is needed to ensure that the colors whose frequencies are stored in v include the k most frequent colors in $[L_{v_i},R_{v_i}]$ for all i.

We cannot accomplish the goal above using one sequence. However, we can accomplish it by defining several nested interval sequences $\mathcal{I}_v^0,\ldots,\mathcal{I}_v^{g-1}$ (the second requirement is now that the leaf range of v_i appears in at least one sequence). Let $I'_0,\ldots,I'_s$ be the leaf ranges of $v_0,\ldots,v_s$, where $I'_t=[L_t,R_t]$. Construct the sequence $\mathcal{I}_v^i$ by starting with a sequence containing the interval I'_0. Then, add the following intervals:

$$[L_0, R_0+1], [L_0, R_0+2], \ldots, [L_0, R_0+i'], [L_0-1, R_0+i'], [L_0-2, R_0+i'], \\ \ldots, [L_1, R_0+i'], [L_1, R_0+i'+1], [L_1, R_0+i'+2], \ldots, [L_1, R_1],$$

where $i' = \min(i, R_1 - R_0)$. Continue this process with the intervals $I'_2, \ldots, I'_s$.

For each sequence $\mathcal{I}_v^i$, the index stores the same information as before (namely, the frequencies of colors in $\mathrm{col}_{k,E,\mathcal{I}_v^i}$, the type of the colors, and the sequence $\mathrm{ind}_{k,E,\mathcal{I}_v^i}$). Answering a query is similar to before. In step 5, the algorithm builds the sequence $\mathcal{I}_v^i$, where $i = R - R_u$ (note that $[L, R]$ is in $\mathcal{I}_v^i$). The query time complexity remains the same.

Compared with the previous structure, each node now stores information for g interval sequences. Therefore, the space for storing the sequence information is multiplied by $g = \log\log n$. The space for a single node is now $O\left((k\log k + k\sqrt{l_k}\log\log n)\log\log n\right)$, and the space over all nodes in all trees remains $o(n)$.

We have shown the following.

Theorem 7. *There is an index for top-k most frequent document retrieval that uses* $|\mathrm{CSA}| + o(n) + D\log\frac{n}{D} + O(D)$ *bits, and answer queries in time* $O(t_{\mathrm{search}} + k\log k(t_{\mathrm{SA}} + \log\log k + \log\log\log n)\log n(\log\log n)^4)$.

References

[1] J. Barbay, T. Gagie, G. Navarro, and Y. Nekrich. Alphabet partitioning for compressed rank/select and applications. In *Proc. 21st International Symposium on Algorithms and Computation (ISAAC)*, pages 315–326, 2010.

[2] D. Belazzougui and G. Navarro. Alphabet-independent compressed text indexing. In *Proc. 19th European Symposium on Algorithms (ESA)*, pages 748–759, 2011.

[3] D. Belazzougui and G. Navarro. Improved compressed indexes for full-text document retrieval. In *Proc. 18th Symposium on String Processing and Information Retrieval (SPIRE)*, pages 286–297, 2011.

[4] H. Cohen and E. Porat. Fast set intersection and two-patterns matching. *Theoretical Computer Science*, 411(40-42):3795–3800, 2010.

[5] P. Ferragina, N. Koudas, S. Muthukrishnan, and D. Srivastava. Two-dimensional substring indexing. *J. of Computer and System Sciences*, 66(4):763–774, 2003.

[6] J. Fischer, T. Gagie, T. Kopelowitz, M. Lewenstein, V. Mäkinen, L. Salmela, and N. Välimäki. Forbidden patterns. In *Proc. 10th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 327–337, 2012.

[7] T. Gagie, G. Navarro, and S. J. Puglisi. Colored range queries and document retrieval. In *Proc. 17th Symposium on String Processing and Information Retrieval (SPIRE)*, pages 67–81, 2010.

[8] Y. Han. Deterministic sorting in $O(n \log \log n)$ time and linear space. *J. of Algorithms*, 50(1):96–105, 2004.

[9] W.-K. Hon, M. Patil, R. Shah, and S.-B. Wu. Efficient index for retrieving top-k most frequent documents. *J. of Discrete Algorithms*, 8(4):402–417, 2010.

[10] W.-K. Hon, R. Shah, and S. V. Thankachan. Towards an optimal space-and-query-time index for top-k document retrieval. In *Proc. 23rd Symposium on Combinatorial Pattern Matching (CPM)*, pages 173–184, 2012.

[11] W.-K. Hon, R. Shah, S. V. Thankachan, and J. S. Vitter. String retrieval for multi-pattern queries. In *Proc. 17th Symposium on String Processing and Information Retrieval (SPIRE)*, pages 55–66, 2010.

[12] W.-K. Hon, R. Shah, S. V. Thankachan, and J. S. Vitter. Document listing for queries with excluded pattern. In *Proc. 23rd Symposium on Combinatorial Pattern Matching (CPM)*, pages 185–195, 2012.

[13] W.-K. Hon, R. Shah, and J. S. Vitter. Space-efficient framework for top-k string retrieval problems. In *Proc. 50th Symposium on Foundation of Computer Science (FOCS)*, pages 713–722, 2009.

[14] Y. Matias, S. Muthukrishnan, S. C. Sahinalp, and J. Ziv. Augmenting suffix trees, with applications. In *Proc. 6th European Symposium on Algorithms (ESA)*, pages 67–78, 1998.

[15] S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proc. 13th Symposium on Discrete Algorithms (SODA)*, pages 657–666, 2002.

[16] G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1), 2007.

[17] G. Navarro and Y. Nekrich. Top-k document retrieval in optimal time and linear space. In *Proc. 23rd Symposium on Discrete Algorithms (SODA)*, pages 1066–1077, 2012.

[18] R. Raman, V. Raman, and S. R. Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, 3(4), 2007.

[19] K. Sadakane. Succinct data structures for flexible text retrieval systems. *J. of Discrete Algorithms*, 5(1):12–22, 2007.

[20] K. Sadakane and G. Navarro. Fully-functional succinct trees. In *Proc. 21st Symposium on Discrete Algorithms (SODA)*, pages 134–149, 2010.

[21] N. Välimäki and V. Mäkinen. Space-efficient algorithms for document retrieval. In *Proc. 18th Symposium on Combinatorial Pattern Matching (CPM)*, pages 205–215, 2007.