Document Listing for Queries with Excluded Pattern *

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. We consider the problem of indexing $\mathcal{D}$ such that, whenever two patterns P^+ and P^- comes as an online query, we can list all those documents containing P^+ *but not* P^-. Let t represent the number of such documents. An index proposed by Fischer et al. (LATIN, 2012) can answer this query in $O(|P^+| + |P^-| + t + \sqrt{n})$ time. However, its space requirement is $O(n^{3/2})$ bits. We propose the first linear-space index for this problem with a worst case query time of $O(|P^+| + |P^-| + \sqrt{n} \log\log n + \sqrt{nt} \log^{2.5} n)$.

1 Introduction and Related Work

Document retrieval is a fundamental problem in information retrieval, where the task is to index a collection of documents, such that whenever a pattern (or a set of patterns) comes as an online query, we can efficiently retrieve those documents which are relevant to the query. An occurrence of a query pattern in a document makes it relevant to the query. However, query with excluded patterns is a problem orthogonal to this. That is, the occurrence of an excluded pattern in a document makes it less relevant to the query. Such queries are fundamental and important in web-search applications. For example, the search results from Google for a pattern "jaguar" consists of many webpages related to "jaguar car", but one may be interested in jaguar as a big cat, not as a car. Whereas the search results for the query "jaguar -car" will be those documents which are related to "jaguar", but not to "car". Here the "-" symbol before the pattern"car" indicates that it is an excluded pattern.

More formally, we shall define the document listing problem for excluded pattern queries as follows: given a collection $\mathcal{D}$ of D documents $\{d_1, d_2, ..., d_D\}$ of total length n, and the query consisting of two patterns P^+ (called included pattern) and P^- (called excluded pattern), our task is to list the set of documents containing P^+ *but not* P^-. Traditionally,

* 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).

the documents are split into terms (or words) and then an inverted index is built over such terms. However, in the case of genome data or some Asian texts, there may be no natural word demarcation (we may call such documents as strings), so that the inverted index may provide only limited searching capabilities or may require too much space. To the best of our knowledge, the only known index which supports this kind of queries for string documents is by Fischer et al. [8], which takes $O(n^{3/2})$ bits of space and has $O(|P^+| + |P^-| + t + \sqrt{n})$ query time, where t is the number of documents containing P^+ *but not* P^-. We propose the first linear-space solution for this problem, and our main result is captured in the following theorem.

Theorem 1 *Given a collection of D string documents of total length n, there exists an $O(n)$-word data structure that supports listing documents with P^+ but not P^- in $O(|P^+| + |P^-| + \sqrt{n} \log\log n + \sqrt{nt} \log^{2.5} n)$ time, where P^+ and P^- are two online query patterns and t represents the number of such documents.*

On a related note, string document retrieval problem for queries with a single (included) pattern is a well studied problem [21, 26, 27, 20] with many interesting results. Another fundamental problem which has received a lot of attention recently is the top-k document retrieval [20, 22, 18, 1, 5, 9, 12, 23, 24, 15, 14]. Muthukrishnan [21] has studied the problem where the query consists of an excluded pattern alone, and has given an optimal-query-time solution. Document listing for queries with two included-patterns (P_1 and P_2) is another harder problem, and the following are the space-time tradeoffs of the known indexes (here t represents the number of documents containing both P_1 and P_2):

- $\tilde{O}(n^{3/2})$-space[†] and $O(|P_1| + |P_2| + \sqrt{n} + t)$ query time [7].
- $O(n \log n)$ words and $O(|P_1| + |P_2| + \sqrt{n(t+1)} \log^{2.5} n)$ query time [4].
- $O(n)$ words and $O(|P_1| + |P_2| + \sqrt{n(t+1)} \log^{1.5} n)$ query time [16].

Fischer et al. [8] showed that document listing problem for two-included-pattern queries is much harder than the one with single-included-pattern using reduction techniques via Geometric Burrows-Wheeler Transform (GBWT) [3].

2 Preliminaries

2.1 Suffix Trees and Suffix Arrays

Suffix Tree: Given a text $T[1...n]$, a substring $T[i...n]$ with $1 \leq i \leq n$ is called a suffix of T. The lexicographic arrangement of all n suffixes of T

[†] The notation $\tilde{O}$ ignores poly-logarithmic factors. Precisely, $\tilde{O}(f(n)) \equiv O(f(n) \log^{O(1)} n)$.

in a compact trie is called the *suffix tree* of T [28], where the ith leftmost leaf represents the ith lexicographically smallest suffix. Each edge in the suffix tree is labeled by a character string and for any node u, path(u) is the string formed by concatenating the edge labels from root to u. For any leaf v, path(v) is exactly the suffix corresponding to v. For a given pattern P, a node u is defined as the *locus node* of P if it is the closest node to the root such that P is a prefix of path(u); such a node can be determined in $O(|P|)$ time.

Suffix Array: Suffix array $SA[1...n]$ of a text T is an array such that $SA[i]$ stores the starting position of the ith lexicographically smallest suffix of T [19]. In SA the starting positions of all suffixes with a common prefix are always stored in contiguous range. The suffix range of a pattern P is defined as the maximal range $[\ell, r]$ such that for all $j \in [\ell, r]$, P is a common prefix of the suffix which starts at $SA[j]$.

Generalized Suffix Tree: Given a collection $\mathcal{D}$ of strings, the *generalized suffix tree* (GST) of $\mathcal{D}$ is a compact trie which stores all suffixes of all strings in $\mathcal{D}$. For the purpose of our index, we define an extra array D_A called *document array*, such that $D_A[i] = j$ if and only if the $i$the lexicographically smallest suffix is from document d_j.

2.2 Wavelet Tree

Let $A[1...n]$ be an array of length n, where each element $A[i]$ is a symbol drawn from a set Σ of size σ. The *wavelet tree* (WT) [11] for A is an ordered balanced binary tree on Σ, where each leaf is labeled with a symbol in Σ, and the leaves are sorted alphabetically from left to right. Each internal node W_k represents an alphabet set Σ_k, and is associated with a bit-vector B_k. In particular, the alphabet set of the root is Σ, and the alphabet set of a leaf is the singleton set containing its corresponding symbol. Each node partitions its alphabet set among the two children (almost) equally, such that all symbols represented by the left child are lexicographically (or numerically) smaller than those represented by the right child. For the node W_k, let A_k be a subsequence of A by retaining only those symbols that are in Σ_k. Then B_k is a bit-vector of length $|A_k|$, such that $B_k[i] = 0$ if and only if $A_k[i]$ is a symbol represented by the left child of W_k. Indeed, the subtree from W_k itself forms a wavelet tree of A_k. To reduce space requirement, the array A is not stored explicitly in the wavelet tree. Instead, we only store the bit-vectors B_k, each of which is augmented with Raman et al.'s scheme [25] to support constant-time bit-rank and bit-select operations. WT takes $n \log \sigma (1 + o(1))$ bits space and can answer the following queries in $O(\log \sigma)$ time.
$rank_c(i)$ = number of occurrences of $c \in \Sigma$ in $A[1...i]$

$select_c(i) = -1$ if $rank_c(n) < i$, else return j, where $A[j] = c$ and $rank_c(j) = i$.

Note that by using the $n \log \sigma + O(n \log \sigma / \log \log \sigma)$ bits index by [10], $rank_c$ and $select_c$ can be performed in $O(\log \log \sigma)$ time.

2.3 Weight-Balanced Wavelet Tree

Weight-balanced wavelet tree (WBT) is a modified version of WT proposed by Hon et al. [16]. Here the number of 0's and 1's in any bit-vector B_k is made almost equal, which ensures the following property.

Lemma 1 *Let W_k be a node in WBT at depth δ_k, and B_k denote its associated bit-vector. Let $n_k = |B_k|$. Then we have $n_k \leq 4n/2^{\delta_k}$.*

WBT on an array $A[1...n]$ takes $n(\log \sigma + 2)(1 + o(1))$ bits of space. The tree depth of WBT can be of $O(\log n)$, so that the worst case query time (for $rank_c(i)$ and $select_c(i)$ for any $c \in \Sigma$) is $O(\log n)$. See Appendix A and B for more details of WBT.

3 Data Structures for Document Counting

Here we describe an index which can count the number of documents containing P^+ *but not* P^-. We capture the result in the following theorem.

Theorem 2 *There exists an $O(n)$-word index that supports counting the number of documents with P^+ but not P^- in $O(|P^+| + |P^-| + \sqrt{n} \log \log n)$ time, where P^+ and P^- are two online query patterns.*

***Index Construction*:** The following shows the main components of the document counting index.

- GST/GSA, the generalized suffix tree/array of $\mathcal{D}$.
- Document array D_A, where $D_A[i] = j$ if the ith lexicographically smallest suffix belongs to document d_j.
- An $2n + o(n)$ bits structure, which can compute *document-frequency* $df(P)$ of a pattern P in $O(1)$ time from the suffix range of P [26].*
- COUNT matrix, to be defined below.

First, starting from left in GST, we combine every g (called group size, to be determined later) contiguous leaves together to form a group. Thus, the first group consists of $\ell_1, ..., \ell_g$, the next group consists of $\ell_{g+1}, ..., \ell_{2g}$, and so on, where ℓ_j denotes the jth leftmost leaf in GST. Consequently, we have a total of $O(n/g)$ groups, and for each group we mark the least common ancestor (LCA) of its first and its last leaves. Moreover, if two nodes

* $df(P)$ = the number of distinct documents in $\mathcal{D}$ which has at least one occurrence of P.

are marked, we mark their LCA as well. The total number of marked nodes by this scheme can be bounded by $O(n/g)$ [13]. Now suppose for any node u in GST, with its subtree containing the leaves $\ell_x, \ell_{x+1}, \ldots, \ell_y$, then the range $[x, y]$ is referred to as the *suffix range* corresponding to u.

Lemma 2 *[13] The suffix range $[L, R]$ of any pattern P can be split into a suffix range $[L', R']$ corresponding to some marked node u^*, and two other suffix ranges $[L, L'-1]$ and $[R'+1, R]$ with $L' - L < g$ and $R - R' < g$.*

Proof. By setting $L' = g\lceil L/g \rceil + 1$ and $R' = g\lfloor R/g \rfloor$, we have $L' - L < g$ and $R - R' < g$, and the LCA of $\ell_{L'}$ and $\ell_{R'}$ gives the desired marked node u^*. □

Essentially, the suffix range $[L, R]$ of a pattern P corresponds to the leaves $\ell_L, \ell_{L+1}, \ldots, \ell_R$ in the GST. This set of leaves can be partitioned into three groups: those which are under the subtree of u^* ($\ell_{L'}, \ell_{L'+1}, \ldots, \ell_{R'}$), and the remaining two with those on the left of $\ell_{L'}$ and those on the right of $\ell_{R'}$. We shall refer to the latter two groups of leaves ($\ell_L, \ell_{L+1}, \ldots, \ell_{L'-1}$ and $\ell_{R'+1}, \ell_{R'+2}, \ldots, \ell_R$) as *fringe leaves*, each such group contains fewer than g leaves.

Let d be a document in $\mathcal{D}$, and u and v be two nodes in GST. Then we define the following functions:

- $F(d, u, v) = 1$, if d contains the pattern $path(u)$ *but not* the pattern $path(v)$, else 0.
- $COUNT(u, v) = \sum_{d \in \mathcal{D}} F(d, u, v)$, which is the number of documents containing the pattern $path(u)$ *but not* the pattern $path(v)$.

Lemma 3 *The function $F(d, u, v)$ can be evaluated in $O(\psi)$ time, where ψ denotes the time for a $rank_d$ query on D_A.*

Proof. Using the tree encoding of GST, the suffix ranges $[L_u, R_u]$ and $[L_v, R_v]$ corresponding to u and v can be computed in constant time. Then, the number of occurrences of $path(u)$ in d, called *term-frequency* (denoted by $tf(path(u), d)$) can be computed as follows: $tf(path(u), d) = rank_d(R_u) - rank_d(L_u - 1)$. Similarly $tf(path(v), d) = rank_d(R_v) - rank_d(L_v - 1)$. If $tf(path(u), d) \geq 1$ and $tf(path(v), d) = 0$, then $F(d, u, v) = 1$, else 0. Therefore, the time for computing F can be bounded by $O(\psi)$, where ψ denotes the time for a $rank_d$ query on D_A. □

$COUNT$ matrix is simply an $O(n/g) \times O(n/g)$ matrix (of size $O(n^2 \log D/g^2)$ bits), which stores $COUNT(u^*, v^*)$ between all pairs of marked nodes u^* and v^* in GST.

***Query Answering*:** The first step is to obtain the locus nodes u and v (and the corresponding suffix ranges $[L_u, R_u]$ and $[L_v, R_v]$) of P^+ and P^-, respectively. Then, we compute the suffix ranges $[L'_u, R'_u]$ and $[L'_v, R'_v]$ (as described in Lemma 2), and the corresponding marked LCA nodes u^* and v^*. Our objective is to compute $COUNT(u, v)$, where as $COUNT(u^*, v^*)$ is precomputed and is stored in the *COUNT matrix*. We have the following lemma on these values.

Lemma 4 *Given $COUNT(u^*, v^*)$, the value $COUNT(u, v)$ can be computed in $O(g\psi)$ time, where g is the group size and ψ is the time for a $rank_d$ query on D_A.*

Proof : Let $S(u, v)$ represent the set of all documents containing the pattern *path*(u) *but not* the pattern *path*(v), hence $COUNT(u, v) = |S(u, v)|$. Note that for those documents d_j, with none of its suffix corresponding to a fringe leaf (i.e., $D_A[i] \neq d_j$ for all $i \in [L_u, L'_u - 1] \cup [R'_u + 1, R_u] \cup [L_v, L'_v - 1] \cup [R'_v + 1, R_v]$), $d_j \in S(u^*, v^*)$ if and only if $d_j \in S(u, v)$. From this observation, $COUNT(u, v)$ can be computed from $COUNT(u^*, v^*)$ by recomputing the membership of only those documents with suffixes corresponding to a fringe leaf, and the number of such documents is bounded by $4g$. Note that we may not be able to find the set $S(u, v)$ efficiently as we have not stored $S(u^*, v^*)$, however what we are interested is in $|S(v, v)|$, which can be computed from $|S(u^*, v^*)|$ as follows:

```
COUNT(u, v) ← COUNT(u*, v*)
for all distinct documents d corresponding to a fringe
leaf do
  if F(d, u, v) = 1 and F(d, u*, v*) = 0 then
    COUNT(u, v) ← COUNT(u, v) + 1
  else if F(d, u, v) = 0 and F(d, u*, v*) = 1 then
    COUNT(u, v) ← COUNT(u, v) − 1
  end if
end for
return COUNT(u, v)
```

The time for evaluating F is $O(\psi)$, and the number of such distinct documents is bounded by $4g$. This completes the proof of the lemma. □

Therefore document counting query can in general be answered in $O(|P^+| + |P^-| + g\psi)$ time. However, we need to handle the following two special cases as well.

1. When $R_u - L_u + 1 < 2g$, the marked node u^* may not exist. Then we shall retrieve all (at most g) distinct documents corresponding to the suffixes in $[L_u, R_u]$, and eliminate those documents which has a suffix in $[L_v, R_v]$ as well. This can be verified in $O(\psi)$ time per document, hence the total time can be bounded by $O(|P^+| + |P^-| + g\psi)$.

2. When $R_v - L_v + 1 < 2g$ the marked node v^* may not exist.. Therefore, we first retrieve all (at most g) distinct documents corresponding to the suffixes in $[L_v, R_v]$. These are the documents (say excluded documents) which do not contribute to $COUNT(u, v)$. Now, we compute the number of excluded documents which have an occurrence of P^+ as well using D_A in $O(g\psi)$ time. By subtracting this number from $df(P^+)$ (the number of distinct documents where P^+ occurs), we get $COUNT(u, v)$, and the total time can be bounded by $O(|P^+| + |P^-| + g\psi)$.

The space and time bounds in Theorem 2 can simultaneously be achieved by choosing $g = \sqrt{n}$, and by maintaining D_A using the data structure in [10], where $\psi = O(\log \log D) = O(\log \log n)$.

4 Data Structures for Document Listing

Our index supporting document listing consists of the following components:

- GST of $\mathcal{D}$.
- *Weight-balanced wavelet tree* (WBT) over document array D_A.
- Let W_k represent an internal node in WBT, $\mathcal{D}_k$ be the set of distinct documents represented by the leaf nodes in the sub-tree of W_k and $n_k = \sum_{d_j \in \mathcal{D}_k} |d_j|$. At every internal node W_k, we maintain the index (from Section 3) for answering document counting query for the corresponding document collection $\mathcal{D}_k$. However, to save space, we do not maintain the generalized suffix tree GST_k of $\mathcal{D}_k$; instead, we maintain only its tree encoding[4] along with marked nodes information and the $2n_k + o(n_k)$ bits data structure for finding *document-frequency*. Moveover we do not need to maintain separate document array for this collection, since the subtree of W_k in WBT is a *weight-balanced wavelet tree* (WBT_k) on $\mathcal{D}_k$. We choose the group size $g_k = \sqrt{n_k \log n}$ and since we are using WBT, the time for a $rank_d$ query on D_A is $\psi = O(\log n)$.

***Index space*:** The total index space can be computed as follows: GST takes $O(n \log n)$ bits, WBT takes $O(n \log D)$ bits. The bit vector B_k associated with the node W_k is of length n_k. Therefore the tree encoding (along with the marked nodes information and the data structure for computing $df(P)$) of GST_k takes $O(n_k)$ bits space. The $COUNT$ matrix

[4] Any n-node ordered tree can be represented in $2n + o(n)$ bits, such that if each node is labeled by its pre-order rank in the tree, any of the following operations can be supported in constant time [17]: *parent*(i), which returns the parent of node i; *lca*(i, j), which returns the lowest common ancestor of two nodes i and j; and *lmost-leaf*(i)/*rmost-leaf*(i), which returns the leftmost/rightmost leaf of node i.

associated with data structure in node W_k takes $O(n_k^2 \log D/g_k^2) = O(n_k)$ bits by choosing $g_k = \sqrt{n_k \log n}$. Note that $\sum_k |n_k|$ is the size of WBT (in bits). Thus the total space is $O(n \log n)$ bits $= O(n)$ words.

***Query Answering*:** Query answering is performed as follows: After computing the locus nodes of P^+ and P^- in GST, we perform a document counting query on $\mathcal{D}$. This is performed using the count structure associated with the root node in WBT. If the count is non-zero, we do a multi-way search in both child nodes, which correspond to searching two partitions of $\mathcal{D}$. This procedure is continued recursively until we reach a leaf node in the binary tree, thus the document corresponding to that leaf can be listed as an output. At any node, if the count is zero, we do not need to continue the recursive step further in its subtree.

Let $[L, R]$ be the suffix range of a pattern P in GST. Then, the suffix range of P in GST_k can be computed in $O(\psi)$ time by translating the range $[L, R]$ to the node W_k by navigating the WBT. Once we get the suffix range of a pattern, its locus node (and the corresponding marked node) in GST_k can be computed in constant time using the tree encoding [17]. Therefore, we need to perform the pattern searching only once (in GST), and the count queries at each internal node W_k of the WBT can be performed in $O(g_k \psi)$ time, instead of $O(|P^+| + |P^-| + g_k \psi)$ time. The overall query time consists of the following components and can be analyzed as follows:

- *Count Queries*: The count query at an internal node W_k takes $O(g_k \psi) = O(\sqrt{n_k \log n} \log n)$ time. Since WBT ensures that $n_k \leq 4n/2^{\delta_k}$, where δ_k is the depth of W_k, so the overall time for count queries will be bounded by:

$$
\begin{aligned}
&O\left(\sum_{W_k \in WBT_{visited}} \sqrt{n_k \log n} \log n\right) \\
&= O\left(\sqrt{n} \log^{3/2} n \sum_{W_k \in WBT_{visited}} 2^{-\delta_k/2}\right) \\
&= O\left(\sqrt{n} \log^{3/2} n \sqrt{\sum_{W_k \in WBT_{visited}} 1^2} \sqrt{\sum_{W_k \in WBT_{visited}} 2^{-\delta_k}}\right) \qquad (1) \\
&= O\left(\sqrt{n} \log^{3/2} n \sqrt{t \log n} \sqrt{\log(1 + \#\text{ of nodes in } WBT_{visited})}\right) \qquad (2) \\
&= O\left(\sqrt{nt} \log^{2.5} n\right),
\end{aligned}
$$

where Equation (1) is by Cauchy-Schwarz's inequality,[‡] while Equation (2) is by the following fact: In a binary tree T with a total

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

of z nodes, and the depth of a node $u \in T$ is given by δ_u, then $\sum_{u \in T} 2^{-\delta_u} \leq \log(1+z)$[§].

– *Initial pattern matching*: This is the time for searching P^+ and P^- in GST and computing the locus nodes u and v, respectively, which can be bounded by $O(|P^+| + |P^-|)$.
– *WBT tree traversal*: Let $t = COUNT(u, v)$ be the number of outputs. Now, consider a binary tree structure $WBT_{visited}$, which is a subtree of WBT with only those nodes visited when we answer the query. Since each internal node in $WBT_{visited}$ must be on the path from the root to some document in the output set, and since the height of WBT is $O(\log n)$, the number of internal nodes in $WBT_{visited}$ is bounded by $O(t \log n)$. As $WBT_{visited}$ is a binary tree, the total number of nodes (i.e., leaves and internal nodes) is bounded by $O(t \log n)$. Thus, the tree traversal time can be bounded by $O(t \log n)$, since it takes only constant time to traverse from a node to its child node.

Note that even if $t = 0$, we need to spend $O(\sqrt{n} \log^{3/2} n)$ time for count query at the root note of WBT. Putting all things together, we get a query time of $O(|P^+| + |P^-| + \sqrt{n} \log^{3/2} n + t \log n + \sqrt{nt} \log^{2.5} n) = O(|P^+| + |P^-| + \sqrt{n} \log^{3/2} n + \sqrt{nt} \log^{2.5} n)$. The $O(\sqrt{n} \log^{3/2} n)$ term can be improved to $O(\sqrt{n} \log \log n)$ by maintaining an additional $O(n)$-word data structure (described in Theorem 2) for performing the first count query, just in case $t = 0$. This completes the proof of Theorem 1.

5 Concluding Remarks

In this paper, we give the first linear space index for two-pattern queries with one included pattern and one excluded pattern. The technique used in this paper is similar to that in [16], where we define a different COUNT matrix for solving the two-pattern queries problem with positive patterns only. However, there are some subtle differences. In particular, the handling of the fringe leaves, and the analysis of the query time in document listing, are much trickier. For further work, we hope to extend the study to the top-k version of this problem, though we suspect that it may not be easily solved with the existing techniques in the literature for positive patterns. Finally, the authors wish to thank Travis Gagie for providing his manuscript [8] on the first solution to this problem.

[§] This fact can be proven by induction: When T contains a single node this is trivially valid ($\delta_{root} = 0$). And for a general tree, we can split T as the root, and two binary trees T_1 and T_2 of z_1 and z_2 nodes respectively, where $z = 1 + z_1 + z_2$. Then $\sum_{u \in T} 2^{-\delta_u} = 1 + \frac{1}{2}(\sum_{u \in T_1} 2^{-\delta_u} + \sum_{u \in T_2} 2^{-\delta_u}) \leq 1 + \frac{1}{2}(\log(1+z_1) + \log(1+z_2)) \leq \log(2\sqrt{(1+z_1)(1+z_2)}) \leq \log(1 + z_1 + 1 + z_2) = \log(1+z)$.

References

1. D. Belazzougui and G. Navarro. Improved Compressed Indexes for Full-Text Document Retrieval. In *SPIRE*, pages 386–397, 2011.
2. M. A. Bender and M. Farach-Colton. The LCA Problem Revisited. In *LATIN*, pages 88–94, 2000.
3. Y.-F. Chien, W.-K. Hon, R. Shah, and J. S. Vitter. Geometric Burrows-Wheeler transform: Linking range searching and text indexing. In *DCC*, pages 252–261, 2008.
4. H. Cohen and E. Porat. Fast Set Intersection and Two Patterns Matching. *Theor. Comput. Sci.*, 411(40-42): pages 3795–3800, 2010.
5. S. Culpepper, G. Navarro, S. Puglisi, and A. Turpin. Top-k Ranked Document Search in General Text Databases. In *ESA*, pages 194–205, 2010.
6. P. Ferragina, R. Giancarlo, and G. Manzini. The Myriad Virtues of Wavelet Trees. *Inf. and Comp.*, 207(8): 849–866, 2009.
7. P. Ferragina, N. Koudas, S. Muthukrishnan, and D. Srivastava. Two-dimensional substring indexing. *J. Comput. Syst. Sci.*, 66(4): 763–774, 2003.
8. J. Fischer, T. Gagie, T. Kopelowitz, M. Lewenstein, V. Mäkinen, L. Salmela and N. Välimäki. Forbidden Patterns. To appear in *LATIN*, 2012.
9. T. Gagie, G. Navarro, and S. J. Puglisi. Colored Range Queries and Document Retrieval. In *SPIRE*, pages 67–81, 2010.
10. A. Golynski, J. I. Munro and S. S. Rao. Rank/select operations on large alphabets: a tool for text indexing. In *SODA*, pages 368–373, 2006.
11. R. Grossi, A. Gupta, and J. S. Vitter. High-Order Entropy-Compressed Text Indexes. In *SODA*, pages 841–850, 2003.
12. 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.
13. W. K. Hon, R. Shah, and J. S. Vitter. Space-Efficient Framework for Top-k String Retrival Problems. In *FOCS*, pages 713–722, 2009.
14. W. K. Hon, R. Shah, and J. S. Vitter. Compression, Indexing, and Retrieval for Massive String Data. In *CPM*, pages 260–274, 2010.
15. W.-K. Hon, R. Shah, and S. Thankachan. Towards an optimal space-and-query-time index for top-k document retrieval. *CoRR*, arXiv:1108.0554, 2011.
16. W. K. Hon, R. Shah, S. V. Thankachan and J. S. Vitter. String Retrieval for Multi-pattern Queries. In *SPIRE*, pages 55–66, 2010.
17. J. Jansson, K. Sadakane, and W. K. Sung. Ultra-succinct Representation of Ordered Trees. In *SODA*, pages 575–584, 2007.
18. M. Karpinski and Y. Nekrich. Top-K Color Queries for Document Retrieval. In *SODA*, pages 401–411, 2011.
19. U. Manber and G. Myers Suffix Arrays: A New Method for On-Line String Searches. In *SICOMP*, 22(5):935–948, 1993.
20. Y. Matias and S. Muthukrishnan and S. C. Sahinalp and J. Ziv Augmenting Suffix Trees, with Applications. In *ESA*, pages 67–78, 1998.
21. S. Muthukrishnan. Efficient Algorithms for Document Retrieval Problems. In *SODA*, pages 657–666, 2002.
22. G. Navarro and Y. Nekrich Top-k document retrieval in optimal time and linear space. In *SODA*, pages 1066–1077, 2012.
23. G. Navarro and S. J. Puglisi. Dual-Sorted Inverted Lists. In *SPIRE*, pages 309–321, 2010.
24. M. Patil, S. V. Thankachan, R. Shah, W. K. Hon, J. S. Vitter, and S. Chandrasekaran. Inverted Indexes for Phrases and Strings. In *SIGIR*, pages 555-564, 2011.
25. R. Raman, V. Raman, and S. S. Rao. Succinct Indexable Dictionaries with Applications to Encoding k-ary Trees, Prefix Sums and Multisets. *TALG*, 3(4), 2007.

26. K. Sadakane. Succinct Data Structures for Flexible Text Retrieval Systems. *JDA*, 5(1):12–22, 2007.
27. N. Välimäki and V. Mäkinen. Space-Efficient Algorithms for Document Retrieval. In *CPM*, pages 205–215, 2007.
28. P. Weiner. Linear Pattern Matching Algorithms. In *Proc. Switching and Automata Theory*, pages 1–11, 1973.

A Proof of Lemma 1

Let W_ℓ and W_r denote the left and the right children of W_k, respectively. Let B_ℓ and B_r be their corresponding bit-vectors, and n_ℓ and n_r be their lengths. Thus $n_k = n_\ell + n_r$. Let L_k denote the number of occurrences of the least frequent symbol σ' (i.e., σ_1) represented by W_k, and similarly L_ℓ and L_r. By the partitioning property, we can easily show that $n_r - L_r \leq n_l$ and $n_\ell - L_\ell \leq n_r$. Also, $L_k \leq L_\ell$ (resp., L_r). Combining these, $2(n_r - L_r) \leq n_\ell + n_r - L_r \leq n_k - L_k$ (similar is true for $n_\ell - L_\ell$). Thus, we get that the quantity $n_k - L_k$ goes down by at least the factor of half as we go to a child node.

Thus, $n_k - L_k \leq n/2^{\delta_k}$. Now for any node which has at least two leaves in its subtree, $L_k \leq (1/2)n_k$ and thus $n_k \leq 2n/2^{\delta_k}$. Taking leaf nodes into account, we get $n_k \leq 4n/2^{\delta_k}$. This completes the proof of Lemma 1.

B Space of a WBT

Lemma 5 *The space of a weight-balanced wavelet tree of an array A of size n is $n(H_0(A) + 2)(1 + o(1))$ bits, where $H_0(A)$ is the 0th-order empirical entropy of A.*

Proof. Let the depth of a leaf corresponding to the symbol σ_i be δ_i. Then σ_i contributes f_i bits in each bit-vector corresponding to the nodes from root to this leaf (excluding the leaf). Hence the contribution of σ_i towards the total space is $f_i \cdot \delta_i$. By Lemma 1, $\delta_i \leq \log(4n/f_i)$. Therefore, the total size of a weight-balanced wavelet tree is at most $(1 + o(1)) \sum f_i \cdot (\log(n/f_i) + 2) = n(H_0(A) + 2)(1 + o(1))$ bits. This completes the proof of Lemma 5. $\square$