Intermediate Report

**Paper title:**  Top-*K* Color Queries for Document Retrieval

**Authors:** Marek Karpinski & Yakov Nekrich

**Conference/Year:** SODA ‘11.

**1. Problem description & Nature of the results (Time & Space Complexity) :** Thepaper proposes an efficient algorithm for the *top K-color problem.* The problem statement is as follows: Given an array A where every element is assigned with a color c and an associated priority p(c), a query with a range [a,b] and a value K expects the algorithm to report K colors with the highest priorities among all colors that occur in A[a,...b], sorted in decreasing order of their priorities. As a practical application of the *top K-color problem,* the authors explore solutions to several flavors of document retrieval techniques. However, the main contribution of this paper is the optimal time and space complexities of the involved data structures and the core algorithm in solving the *top K-color query* problem.

The data structures designed in this paper uses a new explicit technique for recursive, exponentially decreasing size subarrays combined with a new method for caching pre-computed answers. Specifically, the authors show that an array A can be stored in **O(N log )** bits data structure so that for any two indices a < b and for any integer K, K distinct colors with highest priorities among all colors occurring in A[a..b] can be reported in **O(K)** time. The authors first design a data structure with O(N1/f + K), f > 1, query time and then transform the latter into the proposed data structure with optimal query time. This transformation critically depends on an efficient method for obtaining solutions for pre-computed intervals, and recursively defined data structures, a.k.a. trees, with exponentially decreasing number of elements.

**2. Key Ideas/Techniques :**

The authors start with an augmented yet simple tree data structure(DS), and thereafter consequently improve their methods of traversal and retrieval to generate optimal space and query time.

- **Wavelet tree** [O(Nlog2N) Space Data structure] (Section 3)

The first DS the authors use is a **wavelet tree.** The design of a wavelet tree is as follows. At each node, we split the color array C(Cv) into two equal parts such that every color in one part(C0) has a smaller priority than any color in the second(C1). Along with Cv, the node v also stores Av, the array A within the range [av,bv]. A bit array Bv is also stored at node v, such that Bv[i] = 1 iff Av[i] belongs to C1, 0 otherwise. Let us call the left child with color C0 as node w, and the right child with color C1 as u. Now, given Av and Bv, we can compute the bounds of nodes u as follows. We compute au()(**subscript**) as the number of 1’s in Av[1,av]. If (ask prateep)**Bv[av]** = 0, we increment au. Similarly, bu is the number of 1’s in Av[1,bv]. Values of aw and bw are also computed in a similar fashion, by counting numbers of 0’s. A modified copy of array Av is stored at each node u(Au) and w(Aw), such that [TODO]. The search algorithm is as follows: Initialize av = a and bv = b at the root node v. Also compute Bv.

- Compute [au,bu] of right child from Bv and Av.

- Count the number of colors (say mu) at node u in range A[au,bu]. //i need to check again

- If mu >= K, report the top K colors in range [au,bu].

- If mu < K, report mu colors. Decrease K by mu, and then compute [aw,bw] for the left node w.

- Set v = w, and recursively search the left subtree.

Thus, every time we are splitting the whole color domain into two. Hence, total number of visited nodes is O(logN) . The total number of bits stored at each node v is O(logN \* logN) = O(log2N), where Cv requires O(logN) bits and Av requires N bits. Thus, **total bits to store the wavelet tree is** **O(Nlog2N)**. Also, at each node v, the time complexity for color counting is O(logN) and color reporting is O(Kv), where Kv is the number of colors in Cv. Thus, at each node, the total query time is O(logN+Kv). Therefore, **total time complexity for each query in such a tree is O(log2N + K).**

- **A Linear Space DS** (Section 4)

//reporting and counting only in selected nodes of the tree

The first optimization that the authors propose is that instead of looking at nodes at each level of the tree, we can only look at *“important nodes”*. A node v is *“important”* if it is situated on levels whose height are a constat factor(f) of logN, that is those levels whose height are {i/f logN, i = [0,f-1]} . In this version, we only look at an important node and its ***highest important descendants***, the latter signifies those nodes which are children of v and there is no important node in the path from them to v. The maximum number of highest important descendants of a node can be N1/f, (say why). Now, instead of storing the entire array at each node, the algorithm computes an array Ev, such that each index of Ev stores the color-count of each of the highest important descendant of vi. Ev requires O(Nvi logNv) bits of space at each node vi and each query on E requires O(logN) time. Computing E over all visited nodes requires O(NvlogNv) bits of space.

The algorithm proceeds as follows. First, we initialize the root node v with av = a, bv = b, i = t.

- If we know [a,b] at node v, then we can find [avi,bvi] at each of its important descendent nodes

vi ( 1<= i <= t) from Ev. Each element of Ev( |Ev| = t) contains a one dimensional range

counting DS of all colors Ci such that A[vi] is colored with Ci. For each node vi, we count the

number of colors in A[avi,bvi]. Let, this number be mi.

Also, we maintain a sum ri = sum{i=1..t} mj.

- If ri < K, we visit vi and report all mi colors in A[avi,bvi] and proceed to vi-1in decreasing

order.

- If ri >= K, we have to look at the next level of important nodes. Therefore K = K -

ri+1 and we recursively look at level [(i+1)/f logN].

The total number of visited nodes in this case is O(fN1/f) = O(N1/f). At each node, we perform at most one color counting and one reporting query. Thus, the total query time for the tree, using f levels, is O(N1/f logN + K) time. Note however this gives us an unsorted list of top-K colors. Now, if K < N1/f, the list can be sorted in O(N1/f logN) time. If K >= N1/f, the sorting will take O(K) time (using radix sort). Thus, **the overall time complexity is O(N1/f logN+K)**. Considering the space complexity, we can see that for all nodes at same level of the tree we store Ev, and an auxiliary DS for range counting and reporting colors. This takes O(NlogN) bits of space, where N is the length of array A. Finally, since we are only looking at constant levels of the tree, the **total used space remains the same**, **O(NlogN)**.

- A Data structure with O(K) time (Section 5)

- O(N log ) Bit Data structure (Section 6) → Explain in part 1.

- External memory Data structure

- Applications to doc retrieval

**Online Queries:**

**3. Things we don’t understand currently :**

**4. Related Work :**

# *Matias et al. [3] described the first data structure for this problems;their data structure answers document listing queries in O(|P| log s + docc) time, where |P| is the length of P and docc is the number of reported documents(probably write s is set of documents)*

# *Muthukrishnan [4] describes an O(N logN) bits data structure that answers document listing query in optimal O(|P| + docc) time. The data structures of [6, 7] further improve the space usage by storing the documents in compressed form; it takes O(N) time [6] or O(log s) time to report each document.*

# *The solution of Gagie et al. [8], based on the wavelet tree, uses N log s bits but also needs suboptimal (|P| + docc log s) bits to answer a query.*

// check if we want to include the k-mine and repeats problem

In *Hon et al.'s[2]*[paper we only report the relevant documents with respect to a pattern P, instead of all the documents. Their data structure uses linear space (i.e., O(N logN) bits) and can report K most relevant documents in O(|P| + K logK) time. They also describe a compressed data structure that supports queries in O(|P| + Kpolylog(N)) time. (**explain about polylog and how it is space saving**)

*Improvements - Gonzalo Navarroy, Yakov Nekrichz, Top-k document retrieval in optimal time and linear space, Proc. 23rd ACM-SIAM SODA 2012, 1066-1077.*

*the above papaer shows how to reduce the space of the data structure from O(n log n) to O(n(log +log D+log log n)) bits, where is the alphabet size and D is the total number of documents.*

*A recent O(n) space data structure [our paper] enables us to answer top-k queries in O(jPj + k) time when the relevance measure is docrank(d). However, that result cannot be extended to other more important relevance measures.*

*Hon et al.'s results [2] are an important achievement, but their time is not yet optimal.*

*new paper -> A problem not addressed by Muthukrishnan, and arguably the most important one for information retrieval, is the top-k document retrieval problem: report k most highly ranked documents for a query pattern P in decreasing order of their ranks. The ranking is measured with respect to the so-called relevance of a string P for a document d.*

**5. References :**

*[1] P. K. Agarwal, S. Govindarajan, S. Muthukrishnan Range Searching in Categorical Data: Colored Range Searching on Grid, Proc. 10th ESA 2002, 17-28.*

*[2] W.-K. Hon, R. Shah, J. S. Vitter, Space-Efficient Framework for Top-k String Retrieval Problems, Proc. 50th IEEE FOCS 2009, 713-722.*

*[3] Y. Matias, S. Muthukrishnan, S. C. Sahinalp, and J. Ziv, Augmenting Suffix Trees, with Applications, Proc. 6th ESA 1998, 67-78.*

*[4] S. Muthukrishnan, Efficient Algorithms for Document Retrieval Problems, Proc. 13th ACM-SIAM SODA 2002, 657-666.*

*[5] R. Grossi, A. Gupta, and J. S. Vitter, High-order Entropy-compressed Text Indices, Proc. 14th ACM-SIAM SODA 2003, 841-850.* [Wavelet tree]

[6] K. Sadakane, Succinct Data Structures for Flexible Text Retrieval Systems, Journal of Discrete Algorithms, 5(1), 12-22 (2007).

[7] N. V¨alim¨aki and V. M¨akinen, Space-Efficient Algorithms for Document Retrieval, Proc. 18th CPM 2007, 205-215.

[8] T. Gagie, S. J. Puglisi, A. Turpin, Range Quantile Queries: Another Virtue of Wavelet Trees, Proc. 16th SPIRE 2009, 1-6.