Improved Compressed Indexes for Full-Text Document Retrieval [⋆]

Djamal Belazzougui[1] and Gonzalo Navarro[2]

[1] LIAFA, Univ. Paris Diderot - Paris 7, France. dbelaz@liafa.jussieu.fr
[2] Department of Computer Science, University of Chile. gnavarro@dcc.uchile.cl

Abstract. We give new space/time tradeoffs for compressed indexes that answer document retrieval queries on general sequences. On a collection of D documents of total length n, current approaches require at least $|\mathsf{CSA}| + O(n\frac{\lg D}{\lg\lg D})$ or $2|\mathsf{CSA}| + o(n)$ bits of space, where CSA is a full-text index. Using monotone minimum perfect hash functions, we give new algorithms for document listing with frequencies and top-k document retrieval using just $|\mathsf{CSA}| + O(n\lg\lg\lg D)$ bits. We also improve current solutions that use $2|\mathsf{CSA}| + o(n)$ bits, and consider other problems such as colored range listing, top-k most important documents, and computing arbitrary frequencies.

1 Introduction and Related Work

Full-text document retrieval is the problem of, given a collection of D *documents* (i.e., general sequences over alphabet $[1,\sigma]$), concatenated into a text $T[1,n]$, preprocess T so as to later answer various queries of significance in IR. The problem has received much attention recently [16, 22, 24, 11, 8, 7, 4, 12] as a natural evolution over plain full-text indexing (which just counts and locates all the individual occurrences of a pattern $P[1,m]$ in T) and for its applications in IR on Oriental languages, software repositories, and bioinformatic databases. As space is a serious problem in classical solutions [16, 11], most of the focus has been on extending compressed full-text indexes to answer various document retrieval queries. The most studied queries, among several others, are the following.

Document listing: List the distinct documents where P appears.
Document listing with frequencies: List the distinct documents where P appears, and the frequency (number of occurrences) of P in each.
Top-k retrieval: List the k documents where P appears most times.

A compressed full-text index [17] is used as the base data structure. This is usually a compressed suffix array of T (we call this structure CSA and its bit space $|\mathsf{CSA}|$). The CSA simulates the suffix array $A[1,n]$ [13], where $A[i]$ points to the ith lexicographically smallest suffix in T. The CSA finds the interval $A[sp,ep]$ of occurrences of P in time t_{search}, usually $O(m\lg\sigma)$ or less [9, 5]. It can

⋆ Partially funded by Fondecyt Grant 1-110066, Chile. First author also partially supported by the French ANR-2010-COSI-004 MAPPI Project.

also compute any cell $A[i]$, and even $A^{-1}[i]$, in time $O(t_{\mathsf{SA}})$, usually $O(\lg^{1+\varepsilon} n)$ for any constant $\varepsilon > 0$. These indexes represent the text *and* the suffix array within as little as $nH_h(T) + o(n \lg \sigma)$ bits, for any $h \leq \alpha \lg_\sigma n$ and constant $\alpha < 1$. Here $H_h(T)$ is the empirical h-th order entropy of T [14], a lower bound on the bits-per-symbol a statistical order-h compressor may achieve on T.

In the rest of the section we describe our contributions in context. We introduce at this point the concepts of binary *rank*, *select*, and of a monotone minimum perfect hash function (*mmphf*). Given a bitmap $B[1, n]$ with m bits set, operation $rank_b(B, i)$ counts the number of occurrences of bit $b \in \{0, 1\}$ in $B[1, i]$, whereas $select_b(B, j)$ is the position of the jth occurrence of bit b in B. There exists a representation for B using $\lg \binom{n}{m} + O(\lg \lg m) + o(n) = m \lg \frac{n}{m} + O(m) + o(n)$ bits [21], solving both operations in constant time. As B can be reconstructed using operation *rank*, this space is asymptotically optimal. A mmphf can be seen as a weaker structure on B, able to answer only $rank_1(B, i)$ whenever $B[i] = 1$ and giving an arbitrary value otherwise (the mmphf is unable to tell whether $B[i] = 1$ or 0). As it cannot reconstruct B, the mmphf can be represented within less space than the previous lower bound: within $O(m \lg \lg \frac{n}{m})$ bits it answers the limited *rank* query in constant time, and using $O(m \lg \lg \lg \frac{n}{m})$ bits it takes time $O(\lg \lg \frac{n}{m})$ [2].

1.1 Document Listing with Frequencies

The pioneering work in this area [16] defines a *document array* $E[1, n]$, where $E[i]$ tells the document to which suffix $A[i]$ belongs. As noted by Sadakane [22], a bitmap $B[1, n]$ marking the document boundaries in T is enough to find $E[i] = rank_1(B, A[i])$ in time $O(t_{\mathsf{SA}})$. The extra space for B is just $D \lg \frac{n}{D} + O(D) + o(n)$ bits [21]. This permits simulating Muthukrishnan's optimal document listing algorithm [16] within time $O(t_{\mathsf{SA}})$ per document reported, in addition to the time t_{search}. The total space is $|\mathsf{CSA}| + O(n)$, the latter coming from range minimum query (RMQ) data structures [6]. The space was made succinct by Hon et al. [11], by sparsifying the RMQ structures over array blocks of size $\lg^\varepsilon n$, so that the time raises to $O(t_{\mathsf{SA}} \lg^\varepsilon n)$ and the space decreases to $|\mathsf{CSA}| + o(n) + D \lg \frac{n}{D} + O(D)$.

We do not innovate on the plain document listing problem, but on the variant that computes frequencies. The solutions build over plain document listing and add extra data structures using two main approaches. A first one stores, in addition to the CSA of the whole collection, one CSA_d for each individual document d, for a total space of $2|\mathsf{CSA}| + O(n)$ [22] or $2|\mathsf{CSA}| + o(n) + D \lg \frac{n}{D} + O(D)$ [11]. This extra $|\mathsf{CSA}|$ space is used to compute document frequencies along the document listing. The times are as for document listing without frequencies.

A second approach [24, 8] represents the document array directly, in the form of a wavelet tree [9]. This data structure makes the document listing times independent of t_{SA} and enables algorithms that do not derive from Muthukrishnan's [8], listing each document in $O(\lg D)$ time. The space, however, is at least $n \lg D + o(n)$ (by using a recent encoding of the redundancy [20]).

Gagie et al. [7] abstracted this problem in terms of representing a sequence E providing support for accessing any element of E, enumerating each distinct element in a range of E, and computing sequence $rank_d(E, i)$ (the number of occurrences of document d in $E[1, i]$), so that each document can be listed within the sum of these three times. The abstraction enabled new space/time tradeoffs for document listing with frequencies, achieving times as good as $O(\lg \lg D)$.

An interesting observation of Gagie et al. was that one could use *succinct indexes* over a given sequence representation, for example in order to support the $rank_d$ operation on top of just the B bitmap. These "weaker" representations that need an auxiliary mechanism to compute the cells of E are able to reduce space. For example, they achieved $O(n \frac{\lg D}{\lg \lg D})$ bits with $O(t_{\mathsf{SA}} \lg \lg D)$ time by using a succinct index by Grossi et al. [10]. The very same lower bounds on sequence *rank* given by Grossi et al. show that this tradeoff is optimal.

Our first major contribution improves upon this apparent lower bound. We obtain a succinct index on top of the B bitmap that enables us to carry out document listing with frequencies within *less time and space* than the best previous succinct index. We achieve $O(n \lg \lg D)$ bits extra space and $O(t_{\mathsf{SA}})$ time, or $O(n \lg \lg \lg D)$ bits space and $O(t_{\mathsf{SA}} + \lg \lg D)$ time per reported document.

Our solution is based on mmphfs. As we can solve only a limited case of *rank*, we cannot follow Gagie et al.'s framework [7]. Instead, we simulate Sadakane's method [22] using mmphfs instead of a second CSA. Our space/time results are incomparable with those of Sadakane. Compared to the methods that represent directly the document array, we obtain the least space, while the time comparison depends on t_{SA} (e.g., there are full-text indexes where $t_{\mathsf{SA}} = O(\lg^{\varepsilon} n \lg^{1-\varepsilon} \sigma)$ for any $\varepsilon > 0$, yet they require $O((1 + \frac{1}{\varepsilon})nH_h(T))$ bits [9]).

Actually our solution is general enough to solve the *colored range listing* problem, that is, finding the distinct colors (and their frequencies) of any range in an array $E[1, n]$ of D possible colors. Our solution is the *first* in achieving *optimal time* (i.e., $O(1)$ time per color reported) within *succinct space* (i.e., $n \lg D + n\, o(\lg D)$ bits). Achieving this optimal time involves solving in linear time a particular sorting problem, which can be of independent interest.

Table 1 summarizes our results on this part.

1.2 Top-k Document Retrieval

The pioneering work of Hon et al. [11] uses a sampled suffix tree [1] of $o(n)$ extra bits to reduce this problem to that of accessing $E[i]$ and computing *arbitrary* frequencies (document listing with frequencies turns out to be a simpler problem). They achieve time $O(t_{\mathsf{search}} + k \lg^{4+\varepsilon} n)$ using $2|\mathsf{CSA}| + o(n)$ bits.

Our *second major contribution* is the reduction of their time to $O(t_{\mathsf{search}} + k \lg k \lg^{2+\varepsilon} n)$. First, we show that by choosing better the block sizes one can reduce one $\lg n$ to $\lg k$ (in practice k is much smaller than n, and this improvement applies to many previous solutions). The other $\lg n$ is removed thanks to an improved algorithm to compute arbitrary frequencies, that reduces the time from their $O(t_{\mathsf{SA}} \lg n)$ to $O(t_{\mathsf{SA}} \lg \lg n)$. While both ideas are simple, their impact on performance is large and general.

When representing the document array with support for *rank* operations, arbitrary document counting is easy. Gagie et al. [7], apart from improving the

Source	Extra space	Extra Time	Space (colors)	Time (colors)
[16]	$O(n \lg n)$	$O(1)$	$O(n \lg n)$	$O(1)$
[22]	$\lvert\mathsf{CSA}\rvert + O(n)$	$O(t_{\mathsf{SA}})$	n/a	n/a
[11]	$\lvert\mathsf{CSA}\rvert + o(n)$ $+D \lg \frac{n}{D} + O(D)$	$O(t_{\mathsf{SA}} \lg^{\varepsilon} n)$	n/a	n/a
[8, 4]	$n \lg D + o(n)$	$O(\lg \frac{D}{ndoc})$	$n \lg D + o(n)$	$O(\lg \frac{D}{ncol})$
[7]	$n \lg D + O(n)$	$O(\frac{\lg D}{\lg \lg n})$	$n \lg D + O(n)$	$O(\frac{\lg D}{\lg \lg n})$
[7]	$n \lg D + O(n)$	$O(\lg \lg D)$	$n \lg D + O(n)$	$O(\lg \lg D)$
[7]	$O(n \frac{\lg D}{\lg \lg D})$	$O(t_{\mathsf{SA}} \lg \lg D)$	$n \lg D + O(n \frac{\lg D}{\lg \lg D})$	$O(\lg \lg D)$
Ours	$O(n \lg \lg D)$	$O(t_{\mathsf{SA}})$	$n \lg D + O(n \lg \lg D)$	$O(1)$
Ours	$O(n \lg \lg \lg D)$	$O(t_{\mathsf{SA}} + \lg \lg D)$	$n \lg D + O(n \lg \lg \lg D)$	$O(\lg \lg D)$

Table 1. Current and new results on document listing with frequencies (left side) and colored range listing with frequencies (right side). On the left, the extra space is on top of the $|\mathsf{CSA}|$ bits of the full-text index. The time complexities are in addition to t_{search}, and per each of the *ndoc* elements returned. They are valid for any constant $\varepsilon > 0$. On the right we give total space, and total time per each of the *ncol* results reported.

time achieved by Hon et al., gave several new space/time tradeoffs by replacing the second $|\mathsf{CSA}|$-bit space by *rank*-capable representations of E.

Replacing the document array by a weak representation based on mmphfs is not straightforward, as mmphfs do not support general *rank*s. Our *third main contribution* is a technique that modifies Hon et al.'s sampled suffix tree [11] so as to achieve the least space among the methods that represent the document array, while increasing their time by an $O(\lg n)$ factor with respect to the most space-consuming variant. The solution owes in part to the observation that there are not too many candidates around a sampled suffix tree node to replace its precomputed top-k documents. This observation can be useful in other scenarios.

Table 2 summarizes the state of the art and our contribution to the top-k problem. As noted by Hon et al. [11], the bounds apply to the frequency mining problem (list all documents with frequency over f), by running top-k queries with $k = 2^j$ for consecutive j values. Our final contribution is to reduce the time to report the k *most important* documents (i.e., they have a fixed priority) where P appears, from $O(t_{\mathsf{search}} + k \lg^{3+\varepsilon} n)$ [11] to $O(t_{\mathsf{search}} + k \lg k \lg^{1+\varepsilon} n)$.

2 Range Color Listing with Frequencies

We solve the following abstract problem: preprocess an array $E[1, n]$ over D colors so as to answer queries of the form: given i and j, list all the *ncol* distinct colors in $E[i, j]$ and their number of occurrences. The connection with the document listing problem with frequencies is obvious.

Muthukrishnan [16] solved this problem without reporting frequencies. He builds an array $F[1, n]$ where $F[i] = \max\{j < i, E[j] = E[i]\}$. Then, using a data structure that answers RMQ queries on F ($rmq(i, j) = \text{arg}\ \min_{i \le r \le j} F[r]$)

Source	Extra space	Extra Time	Simplified time
[11]	$\lvert\mathsf{CSA}\rvert + o(n) + D \lg \frac{n}{D} + O(D)$	$O(t_{\mathsf{SA}} \lg^{3+\varepsilon} n)$	$O(\lg^{4+\varepsilon} n)$
[7]	$\lvert\mathsf{CSA}\rvert + o(n) + D \lg \frac{n}{D} + O(D)$	$O(t_{\mathsf{SA}} \lg D \lg(D/k) \lg^{1+\varepsilon} n)$	$O(\lg^{4+\varepsilon} n)$
Ours	$\lvert\mathsf{CSA}\rvert + o(n) + D \lg \frac{n}{D} + O(D)$	$O(t_{\mathsf{SA}} \lg k \lg(D/k) \lg^{\varepsilon} n)$	$O(\lg k \lg^{2+\varepsilon} n)$
[7]	$n \lg D + o(n)$	$O(\lg D \lg(D/k) \lg^{\varepsilon} n)$	$O(\lg^{2+\varepsilon} n)$
[7]	$O(n \frac{\lg D}{\lg \lg D})$	$O(t_{\mathsf{SA}} \lg D \lg(D/k) \lg^{\varepsilon} n)$	$O(\lg^{3+\varepsilon} n)$
Ours	$n \lg D + o(n)$	$O(\lg k \lg(D/k) \lg^{\varepsilon} n)$	$O(\lg k \lg^{1+\varepsilon} n)$
Ours	$O(n \frac{\lg D}{\lg \lg D})$	$O(t_{\mathsf{SA}} \lg k \lg(D/k) \lg^{\varepsilon} n)$	$O(\lg k \lg^{2+\varepsilon} n)$
Ours	$O(n \lg \lg \lg D)$	$O(t_{\mathsf{SA}} \lg k \lg^{1+\varepsilon} n)$	$O(\lg k \lg^{2+\varepsilon} n)$

Table 2. Current and new results on top-k retrieval, using the same conventions of Table 1. The last column assumes $t_{\mathsf{SA}} = O(\lg^{1+\varepsilon} n)$, as in optimal-space CSAs [5].

in constant time (e.g., Fischer's [6] takes $2n + o(n)$ bits and does not access F), he finds the leftmost occurrences of all distinct colors in $E[i, j]$ in time $O(ncol)$.

For computing frequencies, Sadakane [22] finds also the rightmost occurrences of the colors by building another RMQ structure on the array $\overline{F}$ built on the reverse sequence $\overline{E}$. The colors could be reported in different order when listing their rightmost or leftmost occurrences. He does not represent F nor $\overline{F}$, and as a consequence needs to mark the colors found in an array $V[1, D]$. The rest of Sadakane's solution is particular of document retrieval; we instead build on it to obtain an improved solution to the general problem.

Theorem 1. *We can augment a sequence of n colors in $[1, D]$ with a structure using $O(n \lg \lg D)$ bits, so that range color listing with frequencies can be solved in $O(1)$ time per color reported, or using $O(n \lg \lg \lg D)$ bits and $O(\lg \lg D)$ time.*

The theorem assumes $D = O(n)$; otherwise a mapping to the colors actually occurring in the sequence, using $O(n \lg \frac{D}{n}) + o(D)$ bits [21], must be added.

To achieve the result, for each color c we store in a mmphf f_c the positions i such that $E[i] = c$ (i.e., $f_c(i) = rank_c(E, i)$ if $E[i] = c$). Let n_c be the frequency of color c in E, then this structure occupies $\sum_c O(n_c \lg \lg \frac{n}{n_c})$ bits, which by the log-sum inequality is $O(n(\lg H_0(E) + 1)) = O(n \lg \lg D)$ bits. The two RMQ data structures will add just $O(n)$ bits. Then a query proceeds in four steps:

1. Use the RMQ on (virtual array) F to get the leftmost occurrences of the $ncol$ colors appearing in the interval. This step takes time $O(ncol)$.
2. Use the RMQ on (virtual array) $\overline{F}$ to get the rightmost occurrences of the $ncol$ colors appearing in the interval. This step also takes time $O(ncol)$.
3. Match the left and right occurrences of the $ncol$ colors. This can be done via sorting, but we show how to do it in time $O(ncol)$.
4. For each color with leftmost and rightmost occurrences l_i and r_i, report the color and its frequency $f_c(r_i) - f_c(l_i) + 1$ in constant time.

To avoid the sorting in step 3, we will slightly modify steps 1 and 2. We will store V and the following additional structures:

1. A vector $R[1, \frac{D}{\lg n}]$, where each cell occupies $\lg D$ bits; R uses at most D bits.
2. A dynamic vector Q storing triplets (c_i, l_i, r_i) and taking $O(ncol \lg n)$ bits.
3. A dynamic vector S storing leftmost positions (c_i, l_i), in $O(ncol \lg n)$ bits.
4. A counter C.

Initially the bits in V and R are set to zero[3], Q and S are empty, and C is set to 1. We then run step 1, setting the bits in V as we progress, and appending the unique colors and their leftmost positions (c_i, l_i) in array S.

We now traverse S and, for each color c_i, compute $g = \lfloor c_i / \lg n \rfloor$. Then, if $R[g] = 0$, we set $R[g] = C$ and $c = rank_1(V[g \lg n + 1, (g+1) \lg n], \lg n)$, which can be computed in constant time in the RAM model [15]. Then we append c copies of the dummy triplet $(\#, \#, \#)$ at the end of vector Q and finally update counter $C = C + c$. At the end of this process array Q will be of size $ncol$ and each distinct color in $E[i,j]$ will have an allocated position into Q.

We now retraverse S and write each pair (c_i, l_i) in the triplet $Q[R[g] + p]$, where $p = rank_1(V[g \lg n + 1, g \lg n + r], r)$, $g = \lfloor c_i / \lg n \rfloor$, and $r = c_i - g \lg n$. So V and R simulate pointers to array Q, where we have already the information on leftmost positions, and now are prepared to write the rightmost positions.

Now we run step 2, but instead of using V to check if we have already reported a color c_i, we compute g and p as before and check whether $Q[R[g] + p] = (c_i, l_i, \#)$. If the third component is a $\#$, then we had not seen the color before and can set the component to r_i. Otherwise we have already seen it.

Now Q has the input to step 4, and step 3 is avoided. Note our working space $O(ndoc \lg n)$ bits of the query is of the same order needed to store the output.

Let us now consider the case where our mmphfs use $O(n_c \lg \lg \lg \frac{n}{n_c})$ bits. By the log-sum inequality these add up to $O(n \lg \lg \lg D)$ bits. The time to query f_c is $O(\lg \lg \frac{n}{n_c})$. To achieve $O(\lg \lg D)$ worst case, we use constant-time mmphfs when $\frac{n}{n_c} > D \lg \lg D$. This implies that on those arrays we spend $O(n_c \lg \lg \frac{n}{n_c}) = O(\frac{n}{D \lg \lg D} \lg \lg D) = O(n/D)$ bits, as it is increasing with n_c and $n_c < \frac{n}{D \lg \lg D}$. Adding over all the possible colors c, we have at most $O(n)$ bits.

By applying the algorithm to document retrieval, where accesses to E are through the CSA, we have the following result.

Theorem 2. *We can augment a* CSA *on $T[1,n]$ containing D documents with a data structure using $O(n \lg \lg D)$ bits, so that document listing with frequencies can be solved in time $O(t_{\mathsf{SA}})$ per document reported, or one using $O(n \lg \lg \lg D)$ bits and time $O(t_{\mathsf{SA}} + \lg \lg D)$. The $\lg D$ in the space complexities can be replaced by $\lg(H) + 1$, where $H = \sum \frac{n_d}{n} \lg \frac{n}{n_d}$ and n_d is the length of document d.*

3 Faster Top-*k* Retrieval

In this section we considerably improve the time complexities of Hon et al.'s scheme [11] for top-k retrieval. Their solution partitions the suffix array into

[3] This is done at indexing time. After a query returns the $ncol$ results and sets those $ncol$ bits, we reset them to 0 one by one, leaving V and R ready for the next query.

chunks of $b = k\ell$ bits. A suffix tree [1] on T is built and all the suffix tree nodes that are lowest common ancestors (lca) of consecutive chunk endpoints are represented in a *sampled* suffix tree, which contains $O(n/b)$ nodes. At each sampled node they store the top-k solution of its subtree.

When a pattern is mapped to the suffix array interval $A[sp, ep]$, it is shown that there exists a sampled node covering an area $A[sp', ep']$, where both $sp' - sp$ and $ep - ep'$ are less than b. Therefore one can simply collect the k precomputed candidates and the (at most $2b$) distinct documents mentioned in these remaining intervals, compute their frequencies in $A[sp, ep]$, and take the k highest frequencies. By using y-fast tries [25] on the identifiers and on the frequencies, the process takes time $O(t_{\mathsf{op}} b)$, where $t_{\mathsf{op}} = t_{\mathsf{SA}} + t_{\mathsf{count}} + \lg\lg n$ and t_{count} is the time to count an arbitrary frequency (the $\lg\lg n$ will be absorbed by a $\lg^{\varepsilon} n$ later).

Since k is unknown at indexing time, this structure is built for all k powers of 2 (i.e., $\lg D$ sampled trees), and at query time the next power of 2 is used. By storing the top-k identifiers in increasing order [7] a node uses $O(k \lg(D/k))$ bits, and the total space is $O((n/b)k \lg D \lg(D/k)) = O((n/\ell) \lg D \lg(D/k))$ bits. This allows using $b = k\ell = k \lg D \lg(D/k) \lg^{\varepsilon} n$, which determines the query time.

Something that is not properly considered by Gagie et al. [7] is that if the trees are stored using pointers, then there is a component of $O((n/b) \lg n)$ bits for $k = 1$, and thus ℓ must be at least $\lg^{1+\varepsilon} n$.

To avoid this we store the sampled tree in succinct form [23] using just $2 + o(1)$ bits per node and supporting in $O(1)$ time many operations, including lca, $preorder$ (whose consecutive values are used to index an array storing the top-k candidate data on each node), and $preorder^{-1}$. For each pair of consecutive chunk endpoints p_i and p_{i+1} we store the preorder x_i of the sampled tree node $lca(p_i, p_{i+1})$. As $x_i \geq x_{i-1}$, values $x_i + i$ are increasing, and thus can be stored in a structure of $(n/b) \lg \frac{2n}{n/b} + O(n/b)$ bits that retrieves any x_i in constant time [19][4]. This space is $O((n/b) \lg b) = O(n \frac{\lg k + \lg\lg n}{k \lg D \lg(D/k) \lg^{\varepsilon} n}) = o(n)$. Now we can find in constant time the lowest sampled node covering chunk interval $[L, R]$ as $lca(preorder^{-1}(x_L), preorder^{-1}(x_{R-1}))$. We will omit $preorder^{-1}$ for simplicity.

3.1 Lowering the $\lg D$ Factor to $\lg k$

The fact that we wish to answer queries for any $k \leq D$ translates into a $\lg D$ factor in ℓ, and into the time complexities. If we set a limit k^* on the maximum k allowed at queries, this $\lg D$ becomes $\lg k^*$. We show now that, by carefully choosing ℓ, we can convert the time to $\lg k$.

Instead of choosing $\ell = \lg D \lg(D/k) \lg^{\varepsilon} n$ so that all the sampled suffix trees have the same size, we reduce it to the slightly increasing $\ell = \lg k \lg(D/k) \lg^{\varepsilon} n$. Then the space for a given k is $(n/b)k \lg(D/k) = (n/\ell) \lg(D/k) = \frac{n}{\lg k \lg^{\varepsilon} n}$. Added over all the $k = 2^j$ values this gives $\sum_{j=1}^{\lg D} \frac{n}{j \lg^{\varepsilon} n} = O(\frac{n \lg\lg D}{\lg^{\varepsilon} n}) = o(n)$.

Therefore we obtain times $O(t_{\mathsf{op}} b) = O(t_{\mathsf{op}} k \lg k \lg(D/k) \lg^{\varepsilon} n)$. Note this applies also to previous solutions [7], as shown in Table 2.

[4] Using a constant-time $rank/select$ implementation on their internal bitmap H [15].

3.2 Computing Arbitrary Frequencies

We additionally remove an $O(\lg n)$ factor from Hon et al.'s top-k retrieval query time [11], while using the same asymptotic space. The following theorem states the result building on the improved variant of Gagie et al. [7] and on Section 3.1.

Theorem 3. *Given a concatenation $T[1,n]$ of D documents, the top-k retrieval problem can be solved in time $O(t_{\mathsf{search}}+t_{\mathsf{SA}} k \lg k \lg(D/k) \lg^{\varepsilon} n)$ while using $2|\mathsf{CSA}|+o(n)+D\lg\frac{n}{D}+O(D)$ bits of space, where t_{search} is the time to find the suffix array interval of pattern P in the CSA of T, t_{SA} is the time to compute a position of the suffix array or its inverse, and $\varepsilon > 0$ is any constant.*

The theorem is obtained just by noting that time $t_{\mathsf{count}} = O(t_{\mathsf{SA}} \lg n)$ in Hon et al.'s algorithm comes from a binary search for the ep_d such that an interval $[sp_d, ep_d]$ inside a local CSA_d is mapped to a given interval $[sp, ep]$ in the global CSA. This binary search can be sped up by sampling every $\lg^2 n$ positions in CSA_d and storing their corresponding position in the global CSA. This sampled array stores $\lfloor n_d / \lg^2 n \rfloor$ entries and thus takes $O(n_d/\lg n)$ bits of space for each document d of length n_d. The overall space is thus $O(n/\lg n) = o(n)$.

We store that array of increasing values in a y-fast trie [25] so that a predecessor query takes $O(\lg\lg n)$ time. Then the binary search for ep can be done by first querying the y-fast trie in time $O(\lg\lg n)$, which will delimit an interval of size $\lg^2 n$, and then with a binary search within that interval in time $t_{\mathsf{count}} = O(t_{\mathsf{SA}} \lg\lg n)$. They also need to find sp_d given ep_d, which is similar. With the optimum-space CSA used by Hon et al. [11] this time is $O(\lg^{1+\varepsilon} n)$, and the overall time reduces from $O(\lg^{4+\varepsilon} n)$ per element returned, to $O(\lg k \lg^{2+\varepsilon} n)$.

4 Using Mmphfs for Top-*k* Retrieval

We now use mmphfs f_c as in Section 2, instead of the local CSA_d's. This would give $t_{\mathsf{count}} = \lg\lg D$ using $O(n \lg\lg\lg D)$ bits. Then the time would be $O((t_{\mathsf{SA}} + \lg\lg D + \lg\lg n)k\ell) = O(t_{\mathsf{SA}} k\ell)$, as the $\lg\lg n$ term is absorbed by the $\lg^{\varepsilon} n$ in ℓ.

The problem is that mmphfs do not give a way to compute arbitrary frequencies. We could only do so if the document appeared *both* in $A[sp, sp'-1]$ and $A[ep'+1, ep]$. In such a case we could easily find its leftmost (l_i) and rightmost (r_i) occurrence in $A[sp, ep]$ and compute the frequency as $f_c(r_i) - f_c(l_i) + 1$.

The candidates can be divided into four groups: (1) Appearing only inside $A[sp', ep']$; (2) appearing both in $A[sp, sp'-1]$ and $A[ep'+1, ep]$; (3) appearing only in $A[sp, sp'-1]$; and (4) appearing only in $A[ep'+1, ep]$.

The only interesting candidates of group (1) are those in the precomputed top-k list, for which we must store the frequencies, as we will have no other way to compute them. This raises the $\lg(D/k)$ time of Section 3 to $\lg n$. Candidates of group (2) are found by scanning both subintervals, finding the documents that appear in both, and their leftmost and rightmost positions. This is easily done in time $O(b \lg\lg n)$ with y-fast tries. Then we compute their frequencies using the corresponding mmphf. How to handle the other two groups is considered next.

4.1 Bounding the Number of Valid Candidates

We show that the number of documents that can make it to the top-k list if they appear only to the left (or, similarly, to the right) chunk of the precomputed interval, is $O(k\sqrt{\ell})$. This allows us to store all those potentially relevant documents within the nodes. By storing their frequency in $A[sp', ep']$, we can complete the frequency computation in $A[sp, ep']$ by just traversing the area $A[sp, sp'-1]$ and increasing the frequencies of the documents found (we omit this step on documents that have already been found in both tails, as explained).

In order for a document to be out of the top-k list, but able to make it to the list by scanning the chunk to the left of the sampled node, its frequency must be betwen $f-b+1$ and f, where f is the frequency of the kth most frequent candidate stored. Therefore its frequency can be stored using $O(\lg b) = O(\lg k + \lg\lg n)$ bits. Moreover each document with frequency under $f-\ell+1$ must appear at least ℓ times in the chunk in order to have a chance, thus there are at most $b/k = \ell$ such nodes. The rest need only $O(\lg \ell)$ bits. Therefore the total space per node will be $O(k\lg n + k\lg b + k\sqrt{\ell}\lg\ell)) = O(k\lg n + k\sqrt{\ell}\lg\lg n)$ (note we are *not* storing the document identifiers of these extra candidates), and the overall space for a given $k = 2^j$ will be $O((n/b)k(\lg n + \sqrt{\ell}\lg\lg n))$. For the sum of spaces over j to be $o(n)$ we need that $\ell = \lg k \lg^{1+\varepsilon} n$ for some $\varepsilon > 0$.

To know which documents are indeed candidates (i.e., can make it to the top-k list so we have stored their frequency inside the node) we set up a bitmap of length b marking the rightmost occurrence of such candidates, and their position in the array of frequencies is obtained with $rank_1$ on that bitmap (a second bitmap distinguishes $\lg b$-bit from $\lg\ell$-bit candidates). As it has at most $k\sqrt{\ell}$ bits set, the bitmap can be stored within $O(k\sqrt{\ell}\lg\sqrt{\ell}) = O(k\sqrt{\ell}\lg\lg n)$ bits. Thus we traverse $A[sp, sp'-1]$ right to left. When we find a 1 in this bitmap, this is the first time we see a relevant candidate. We compute its identity in $O(t_{\mathsf{SA}})$ time and find its $A[sp', ep']$ frequency using $rank_1$ as explained. Now we have the data to insert it (increasing its frequency by 1) into the y-fast trie. The next occurrences (when the bitmap has value 0) correspond to candidates that either have already been found (and thus are already inserted in the y-fast trie) or candidates that cannot make it to the top-k list (and thus are not present in the y-fast trie and we must not care about them).

The missing piece is to prove that there are sufficiently few candidates.

Lemma 1. *Let $top_k(s, e)$ be k most frequent colors in an array $E[s, e]$. Then there is a choice of $top_k(\cdot, \cdot)$ sets in case of frequency ties such that, for any b, $C(b) = |\cup_{r=0}^{b} top_k(s-r, e)| < k + \sqrt{2bk}$.*

Proof. Let us call $s_t < s$ the position where $k \cdot t$ new elements have made it in top_k at some point, i.e., $C(s - s_t) = C(0) + kt = k + kt$. Let us call f_r the kth highest frequency in $E[r, e]$. Since all elements not in $top_k(s, e)$ have frequency at most $f = f_s$ in $E[s, e]$, a new element must appear at least once in $E[r, s-1]$ to reach frequency $f+1$ and force us choose it for $top_k(r, e)$. Hence $s_1 \leq s - k$.

Now, as k distinct elements have entered in $top_k(s_1, e)$, it must hold that $f_{s_1} \geq f + 1$, as we have seen k distinct elements reaching frequency $f+1$. Thus the $(k+1)$th *distinct* element appearing in $top_k(r,e)$ must appear at least twice in $E[r,s-1]$, to jump from frequency at most f to at least $f+2$. Thus we need $2k$ occurrences of elements that are incompatible with the previous k occurrences in order to have k new distinct elements, thus $s_2 \leq s - 3k$.

Once these new k distinct elements enter in $top_k(s_2, e)$, it holds that $f_{s_2} \geq f+2$, and thus we need $3k$ incompatible occurrences for the next k occurrences, and so on. Iterating the argument, it holds $s_t \leq s - \frac{t(t+1)}{2}k$ for all $t \geq 1$.

Thus as long as $s_t \geq s - b$ we have $\frac{t(t+1)}{2}k \leq b$, and thus $t < \sqrt{2b/k}$. Hence the number of new elements entering into some $top_k(s-r,e)$ for $1 \leq b \leq r$ is $C(b) < k(t+1) < k + \sqrt{2bk}$. □

In our case $b = k\ell$ so the bound is $C(b) = O(k\sqrt{\ell})$. We have proved the main result. The time simplifies to $O(t_{\mathsf{search}} + k \lg k \lg^{2+\varepsilon} n)$ when $t_{\mathsf{SA}} = \lg^{1+\varepsilon} n$.

Theorem 4. *Given a concatenation $T[1,n]$ of D documents, the top-k retrieval problem can be solved in time $O(t_{\mathsf{search}} + t_{\mathsf{SA}} k \lg k \lg^{1+\varepsilon} n)$ using $O(n \lg\lg\lg D)$ extra bits, where t_{search} is the time to find the suffix array interval of pattern P in the* CSA *of T, t_{SA} is the time to compute a position of the suffix array or its inverse, and $\varepsilon > 0$ is any constant.*

5 Top-*k* Most Important Document Retrieval

A particular variant of top-k document retrieval, somewhat easier than the one that seeks for the highest frequencies, is one where the documents have a fixed *importance* or priority. An example would be the PageRank value of Web pages. A way to handle this problem is to sort the documents by importance, so that document i is the ith most important in the collection. Then the problem becomes that of finding the k smallest distinct values in $E[sp, ep]$. While methods based on range quantile queries on wavelet trees [8] naturally report the documents in sorted order and thus automatically solve this problem in $O(k \lg D)$ time by pruning the process after reporting k results, the situation is not that easy for the other approaches that use potentially less space.

A solution comes from the same top-k retrieval technique of Hon et al. [11]. This time one stores the k smallest document values within each sampled node, and traverses the tails of the interval looking for smaller document identifiers. No frequencies need to be computed, which allows for an $O(t_{\mathsf{SA}} k \lg k \lg(D/k) \lg^{\varepsilon} n)$ time solution, e.g., $O(k \lg k \lg^{2+\varepsilon} n)$. This seems unimportant now that we have reduced the complexity of the more difficult top-k retrieval problem to the same level. Yet, we show that this particular problem can be solved faster, removing the $\lg(D/k)$ factor. When $t_{\mathsf{SA}} = \lg^{1+\varepsilon} n$, this gives time $O(t_{\mathsf{search}} + k \lg k \lg^{1+\varepsilon} n)$.

Theorem 5. *Given a concatenation $T[1,n]$ of D documents, the top-k most important retrieval problem can be solved in time $O(t_{\mathsf{search}} + t_{\mathsf{SA}} k \lg k \lg^{\varepsilon} n)$ while using $|\mathsf{CSA}| + o(n) + D \lg \frac{n}{D} + O(D)$ bits of space, where t_{search} is the time to find the suffix array interval of pattern P in the* CSA *of T, t_{SA} is the time to compute a position of the suffix array or its inverse, and $\varepsilon > 0$ is any constant.*

The result of Hon et al. [11] is achieved by using chunks of $b = k\ell$ positions for $\ell = \lg^{2+\varepsilon} n$ (for the more refined complexity we use $\ell = \lg k \lg(D/k) \lg^{\varepsilon} n$). Our idea is to further divide those chunks into $\lg(D/k)$ buckets of size $b' = k \lg k \lg^{\varepsilon} n$. For each chunk we build a small local sampled suffix tree. A query will then span at most one global node, two local nodes, and two tail buckets.

Consider the endpoints $p_1 \dots p_r$ of the buckets inside a given chunk, and call $v = lca(p_1, p_r)$ the lowest sampled global suffix tree node that covers the chunk. Just as for the global scheme, find in the suffix tree the lca nodes of each pair of consecutive endpoints, $lca(p_i, p_{i+1})$. All those lca nodes are below v or are v.

There are overall $O(n/b')$ local sampled nodes. Moreover, if some node $u = lca(p_i, p_{i+1})$ covers the whole chunk $[p_1, p_r]$, then it must be an ancestor of $v = lca(p_1, p_r)$, but since it is also a descendant of v, we have $u = v$. That is, the local sampled suffix tree nodes (that are not already global sampled suffix tree nodes) cannot cover a chunk and hence span less than $2b$ positions.

Instead of storing the top-k document identifiers using $O(k \lg(D/k))$ bits, for these local sampled nodes we will store the *positions* of some occurrence of those identifiers within the local sampled node, sorted by increasing position. The identifier must be obtained with an access to that position, which will not change the complexity. Since local positions span less than $2b$, they require $O(k \lg(b/k)) = O(k \lg \ell) = O(k \lg \lg n)$ bits. The tree topology itself will require $2 + o(1)$ bits per node, as for the global tree. The total space for a given $k = 2^j$ is $O((n/b')k \lg \lg n) = O(n \frac{\lg \lg n}{\lg k \lg^{\varepsilon} n})$, which added over all $k = 2^j$ values gives $o(n)$ bits overall. We also must store a local node identifier $y_i = preorder(lca(p_i, p_{i+1}))$ for each bucket, which requires $O((n/b') \lg b) = O(n \frac{\lg k + \lg \lg n}{k \lg k \lg^{\varepsilon} n}) = O(\frac{n \lg \lg n}{k \lg^{\varepsilon} n})$, which added over all $k = 2^j$ values gives $o(n)$ bits as well.

To query, we determine the interval $A[sp, ep]$ of P and the covered chunk $[L, R]$, the covered bucket $[l_1, r_1 = Lb'/b]$ to the left of chunk L, and the covered bucket $[l_2 = Rb'/b, r_2]$ to the right of chunk R. Then we find the global sampled node $v = lca(x_L, x_{R-1})$, and the local sampled nodes $u_1 = lca(y_{l_1}, y_{r_1 - 1})$ and $u_2 = lca(y_{l_2}, y_{r_2 - 1})$. If u_1 or u_2 are equal to v we discard them. Now we take the at most $3k$ candidates from v, u_1 and u_2, and also consider the elements in $E[sp, r_1 b' - 1]$ and $E[b' l_2 + 1, ep]$. The time is $O(t_{\mathsf{SA}}(k + b'))$ to extract all the candidate identifiers, plus $O(k \lg \lg n)$ to maintain a heap of the smallest k values seen in the process using a y-fast trie [25]. The time adds up to $O(t_{\mathsf{SA}} k \lg k \lg^{\varepsilon} n)$.

6 Final Remarks

A natural next step is to implement these solutions. Many of our improvements are easy to implement, and practical implementations of mmphfs exist [3]. A recent empirical work [18] shows that the individual CSA_d's pose much space overhead, at least if implemented naively. Instead, they compress wavelet trees to 7-17 bpc (bits per text character), compared to the 4.5-6.0 bpc of the global CSA. Over their same collections, our mmphf implementation takes 3-5 bpc and gives sub-microsecond times. This shows that the alternative of using mmphfs is very appealing compared to both using CSA_d's or wavelet trees.

References

1. A. Apostolico. The myriad virtues of subword trees. In *Combinatorial Algorithms on Words*, NATO ISI Series, pages 85–96. Springer-Verlag, 1985.
2. D. Belazzougui, P. Boldi, R. Pagh, and S. Vigna. Monotone minimal perfect hashing: searching a sorted table with $o(1)$ accesses. In *SODA*, pages 785–794, 2009.
3. D. Belazzougui, P. Boldi, R. Pagh, and S. Vigna. Theory and practise of monotone minimal perfect hashing. In *ALENEX*, 2009.
4. S. Culpepper, G. Navarro, S. Puglisi, and A. Turpin. Top-k ranked document search in general text databases. In *ESA*, pages 194–205 (part II), 2010.
5. P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. Compressed representations of sequences and full-text indexes. *ACM Trans. Alg.*, 3(2):art. 20, 2007.
6. J. Fischer. Optimal succinctness for range minimum queries. In *LATIN*, pages 158–169, 2010.
7. T. Gagie, G. Navarro, and S. J. Puglisi. Colored range queries and document retrieval. In *SPIRE*, pages 67–81, 2010.
8. T. Gagie, S. J. Puglisi, and A. Turpin. Range quantile queries: Another virtue of wavelet trees. In *SPIRE*, pages 1–6, 2009.
9. R. Grossi, A. Gupta, and J. Vitter. High-order entropy-compressed text indexes. In *SODA*, pages 841–850, 2003.
10. R. Grossi, A. Orlandi, and R. Raman. Optimal trade-offs for succinct string indexes. In *ICALP*, pages 678–689, 2010.
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. M. Karpinski and Y. Nekrich. Top-k color queries for document retrieval. In *SODA*, pages 401–411, 2011.
13. U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. *SIAM J. Comp.*, 22(5):935–948, 1993.
14. G. Manzini. An analysis of the Burrows-Wheeler transform. *J. ACM*, 48(3):407–430, 2001.
15. I. Munro. Tables. In *FSTTCS*, pages 37–42, 1996.
16. S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *SODA*, pages 657–666, 2002.
17. G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Comp. Surv.*, 39(1):art. 2, 2007.
18. G. Navarro, S. Puglisi, and D. Valenzuela. Practical compressed document retrieval. In *SEA*, pages 193–205, 2011.
19. D. Okanohara and K. Sadakane. Practical entropy-compressed rank/ select dictionary. In *ALENEX*, 2007.
20. M. Pătraşcu. Succincter. In *FOCS*, pages 305–313, 2008.
21. R. Raman, V. Raman, and S. Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *SODA*, pages 233–242, 2002.
22. K. Sadakane. Succinct data structures for flexible text retrieval systems. *J. Discr. Alg.*, 5(1):12–22, 2007.
23. K. Sadakane and G. Navarro. Fully-functional succinct trees. In *SODA*, pages 134–149, 2010.
24. N. Välimäki and V. Mäkinen. Space-efficient algorithms for document retrieval. In *CPM*, pages 205–215, 2007.
25. D. E. Willard. Log-logarithmic worst-case range queries are possible in space $\theta(n)$. *Inf. Process. Lett.*, 17(2):81–84, 1983.