Colored Range Queries and Document Retrieval

Travis Gagie[1⋆], Gonzalo Navarro[1⋆], and Simon J. Puglisi[2]

[1] Dept. of Computer Science, Univt. of Chile, {tgagie,gnavarro}@dcc.uchile.cl
[2] School of Computer Science and Information Technology
Royal Melbourne Institute of Technology, simon.puglisi@rmit.edu.au

Abstract. Colored range queries are a well-studied topic in computational geometry and database research that, in the past decade, have found exciting applications in information retrieval. In this paper we give improved time and space bounds for three important one-dimensional colored range queries — colored range listing, colored range top-k queries and colored range counting — and, thus, new bounds for various document retrieval problems on general collections of sequences. Specifically, we first describe a framework including almost all recent results on colored range listing and document listing, which suggests new combinations of data structures for these problems. For example, we give the fastest compressed data structures for colored range listing and document listing, and an efficient data structure for document listing whose size is bounded in terms of the high-order entropies of the library of documents. We then show how (approximate) colored top-k queries can be reduced to (approximate) range-mode queries on subsequences, yielding the first efficient data structure for this problem. Finally, we show how a modified wavelet tree can support colored range counting in logarithmic time and space that is succinct whenever the number of colors is superpolylogarithmic in the length of the sequence.

1 Introduction

A *range query* on a sequence $S[1, n]$ of elements in $[1, \sigma]$ takes as arguments two indices i and j and returns information about $S[i, j]$. This information could be, for example, the minimum or maximum value in $S[i, j]$ [12], the element with a specified rank in sorted order [15] (e.g., the median [7]), the mode [17], a complete list of the distinct elements [31], the frequencies of the elements [35], a list of the k most frequent elements for a given k [20], or the number of distinct elements [6]. In this paper, motivated by problems in document retrieval, we consider the latter three kinds of problems, which are often referred to as "colored" range queries: colored range listing (with or without color frequencies), colored range top-k queries, and colored range counting. These have been associated, respectively, to very relevant document retrieval queries on general texts [31, 35, 37, 20, 15, 12, 9]: listing the documents where a pattern appears (possibly computing

⋆ Partially funded by the Millennium Institute for Cell Dynamics and Biotechnology (ICDB), Grant ICM P05-001-F, Mideplan, Chile.

term frequencies), finding the most relevant documents to a query (under a $tf \times idf$ scheme, for example), and computing document frequencies. Such techniques have been shown to be competitive [9], even beating classical inverted indexes on natural-language texts.

In Section 2 we describe a framework that includes almost all recent results on colored range listing and the related problem of document listing. This framework suggests new combinations of data structures that yield interesting new bounds, including the fastest compressed data structures for colored range listing and an efficient data structure for document listing whose space occupancy is bounded in terms of the higher-order entropies of the library of documents. In Section 3 we describe what seems to be the first data structure to support efficient, general approximate colored range top-k queries. By "approximate" we mean that we are given an $\epsilon > 0$ with S and we guarantee that no element we do not list occurs more than $1 + \epsilon$ times more often in the range than any element we list. Finally, in Section 4 we describe a new solution to the colored range counting problem, reducing the space bound from $\mathcal{O}(n \log n)$ bits to $n \log \sigma + \mathcal{O}(n \log \log n)$ bits without changing the $\mathcal{O}(\log n)$ time bound. The improvements for colored range queries we present in Sections 3 and 4 are not competitive with the state of the art when mapped to the more specific problem of document retrieval. However, as we discuss in Section 5, data structures for general colored range queries can be applied to information retrieval scenarios that specialized document-retrieval data structures cannot.

2 Listing

Related work. The problem of colored range listing (CRL) is to preprocess a given sequence $S[1, n]$ over $[1, \sigma]$ such that later, given a range $S[i..j]$, we can quickly list all the distinct elements ("colors") in that range. Almost all recent data structures for CRL (and the related problem of document listing) are based on a key idea by Muthukrishnan [31] (see [23] for older work). He defined $C[1, n]$ to be the array in which $C[j]$ is the largest value $i < j$ such that $S[i] = S[j]$, or 0 if there is no such i, so that $S[\ell]$ is the first occurrence of a color in $S[i..j]$ if and only if $i \leq \ell \leq j$ and $C[\ell] < i$. He showed how, if we store C in an $\mathcal{O}(n \log n)$-bit data structure due to Gabow, Bentley and Tarjan [14] that supports $\mathcal{O}(1)$-time range-minimum queries (RMQs), we can quickly find all the values in $C[i..j]$ less than i and, thus, list all the colors in $S[i..j]$. To do this, we find the minimum value $C[\ell]$ in $C[i..j]$; if it is less than i, then we output $S[\ell]$ and recurse on $S[i..\ell - 1]$ and $S[\ell + 1..j]$. Altogether, Muthukrishnan's CRL data structure uses $\mathcal{O}(n \log n)$ bits and $\mathcal{O}(1)$ time per color reported.

Muthukrishnan gave his solution to the CRL problem as part of a solution to the problem of document listing (DL), in which we are given a library of documents and asked to preprocess them such that later, given a pattern, we can quickly list all the distinct documents containing that pattern (see [29] for older work). Let $T[1, n]$ be the concatenation of the D documents. Muthukrishnan defined the array $E[1, n]$ such that $E[i]$ is the document containing the starting

position of the lexicographically ith suffix in T. If we store a suffix tree [38, 1] for T then, given a pattern, we can quickly find the lexicographic ranks i and j of the first and last suffixes starting with the pattern. This is equivalent to finding the range $A[i..j]$ in the suffix array [27] A for T that lists the starting positions of all the suffixes of T that start with the pattern. Once we know i and j, we can implement a DL query as a CRL query on $E[i..j]$. Altogether, Muthukrishnan's DL data structure uses $\mathcal{O}(n \log n)$ bits and $\mathcal{O}(m + \mathsf{ndoc})$ time to list the ndoc documents containing a pattern of length m.

Sadakane [35] gave a slower but smaller version of Muthukrishnan's DL data structure, in which he replaced Gabow, Bentley and Tarjan's data structure by a $4n + o(n)$ bit index that, given a range $C[i..j]$, in $\mathcal{O}(1)$ time and without consulting C returns the position of the minimum value in that range (but not the value itself). He also replaced the suffix tree by a compressed suffix array (CSA) for T and showed how the CSA and a bit vector $V[1, n]$ can simulate access to E: 1s in V mark the positions in T where the documents start; then, for $1 \leq \ell \leq n$, $E[\ell] = \mathsf{rank}_1(V, \mathsf{CSA}[\ell])$, where $\mathsf{rank}_1(V, r)$ is the number of 1s in $V[1..r]$. It takes $D \log(n/D) + \mathcal{O}(D) + o(n)$ bits to store V such that a rank query takes $\mathcal{O}(1)$ time [33]. Sadakane did not store C at all so, when listing the distinct documents containing a pattern, he used a D-bit string to mark which documents he had already listed. He used a recursion similar to Muthukrishnan's, stopping whenever it finds a document already reported.

Altogether, Sadakane's DL data structure uses $|\mathsf{CSA}| + 4n + D \log(n/D) + \mathcal{O}(D) + o(n)$ bits and $\mathcal{O}(\mathsf{search}(m) + \mathsf{ndoc} \cdot \mathsf{lookup}(n))$ time, where $\mathsf{search}(m)$ is the time to find the range $\mathsf{CSA}[i..j]$ containing the starting positions of suffixes beginning with the pattern and $\mathsf{lookup}(n)$ is the time to compute $\mathsf{CSA}[\ell]$ for any ℓ. (There are a number of CSA implementations, allowing various space/time tradeoffs [32].) He used $|\mathsf{CSA}| + 4n + o(n)$ additional bits for data structures to compute the pattern's frequency in each document, increasing the time bound to $\mathcal{O}(\mathsf{search}(m) + \mathsf{ndoc}(\mathsf{lookup}(n) + \log\log \mathsf{ndoc}))$ (assuming $\mathsf{lookup}(n)$ is also the time to find $\mathsf{CSA}^{-1}[\ell]$, where CSA^{-1} is the inverse permutation).

Välimäki and Mäkinen [37] gave an alternative slower-but-smaller version of Muthukrishnan's CRL data structure, in which they used a $2n + o(n)$ bit, $\mathcal{O}(1)$ time RMQ succinct index due to Fischer and Heun [13] that requires access to C. Välimäki and Mäkinen showed how access to C can be implemented by rank and select queries on S; specifically, for $1 \leq \ell \leq n$, $C[\ell] = \mathsf{select}_{S[\ell]}(S, \mathsf{rank}_{S[\ell]}(S, \ell) - 1)$, where $\mathsf{select}_a(S, r)$ is the position of the rth occurrence of a in S. Välimäki and Mäkinen stored S in a multiary wavelet tree [10], which takes $nH_0(S) + o(n) \log \sigma$ bits and $\mathcal{O}(1 + \log \sigma / \log\log n)$ time; when σ is polylogarithmic in n, it takes $nH_0(S) + o(n)$ bits and $\mathcal{O}(1)$ time. The 0-th order empirical entropy $H_0(S) = \sum_a \frac{\mathsf{occ}(a,S)}{n} \log \frac{n}{\mathsf{occ}(a,S)}$, where $\mathsf{occ}(a, S)$ is the number of times element a occurs in S, is the Shannon entropy of the distribution of elements in S.

Altogether, their CRL data structure takes $nH_0(S) + 2n + o(n) \log \sigma$ bits and $\mathcal{O}(1 + \log \sigma / \log\log n)$ time per reported color. Combining this data structure with a CSA yields a DL data structure that takes $|\mathsf{CSA}| + n \log D + 2n + o(n) \log D$ bits and $\mathcal{O}(\mathsf{search}(m) + \mathsf{ndoc}(1 + \log D / \log\log n))$ time. They also showed how

to compute the pattern's frequency in a document d using two rank queries on E, $\mathsf{rank}_d(E, j) - \mathsf{rank}_d(E, i-1)$. Since multiary wavelet trees support rank queries in the same time as accesses, it follows that reporting the pattern's frequency in all the documents does not affect their time and space bounds. Finally, they noted that, using one select query per occurrence, they can list the positions of the pattern's occurrences in a specified document.

Gagie, Puglisi and Turpin [15] showed that a binary wavelet tree [18] can be used to compute range quantile queries on S in $\mathcal{O}(\log \sigma)$ time, and that these queries can be used to enumerate the distinct elements in $S[i..j]$, eliminating the need for RMQs. A binary wavelet tree for S takes $nH_0(S) + o(n) \log \sigma$ bits and supports access, rank and select in $\mathcal{O}(\log \sigma)$ time; therefore, by itself it is a CRL data structure that takes $\mathcal{O}(\log \sigma)$ time per reported element. Combining a wavelet tree for E with a CSA for T, we obtain a DL data structure that takes $|\mathsf{CSA}| + n \log D + o(n) \log D$ bits and $\mathcal{O}(\mathsf{search}(m) + \mathsf{ndoc} \log D)$ time.

Hon, Shah and Vitter [20] described a solution to DL similar to Sadakane's but removing the $\Theta(n)$-bit space term. They pack $\log^\epsilon n$ consecutive cells of C into a block and build the RMQ data structure on the block minima (so it takes $\mathcal{O}(n/\log^\epsilon n)$ bits of space), and tries to report (avoiding repetitions) all the documents in the block that holds the minimum. Their whole data structure takes $|\mathsf{CSA}| + D \log(n/D) + \mathcal{O}(D) + o(n)$ bits and answers queries in time $\mathcal{O}(\mathsf{search}(m) + \mathsf{ndoc} \log^\epsilon n \cdot \mathsf{lookup}(n))$, for any constant $\epsilon > 0$.

They can also return the number of times the pattern occurs in any document by using, like Sadakane, one CSA_d local to each document d. These add up to other $|\mathsf{CSA}|$ extra bits. To find out how many times document $d = E[\ell]$, $i \leq \ell \leq j$, appears in $E[i..j]$, it maps ℓ to position $p = \mathsf{CSA}[\ell] - \mathsf{select}_1(V, d) + 1$ within document d, and then to $\ell' = \mathsf{CSA}_d^{-1}[p]$. This is the first lexicographic occurrence of the pattern in CSA_d. The last occurrence is found by an exponential search and then binary search on $\mathsf{CSA}_d[\ell'..]$, for the largest c such that $\mathsf{CSA}^{-1}[\mathsf{CSA}_d[\ell' + c] + \mathsf{select}_1(V, d) - 1)] \leq j$. Then the answer, $c + 1$, is obtained in time $\mathcal{O}(\mathsf{lookup}(n) \log c) = \mathcal{O}(\mathsf{lookup}(n) \log n)$.

New tradeoffs. All the previous solutions have essentially the same ingredients: for CRL, access to S, distinct color enumeration on S (implemented via RMQs on C or range quantile queries on S) and, to count the number of times each color occurs, rank on S; for DL, a suffix tree or CSA for T, access to E, distinct document enumeration on E and, to report the pattern's frequency in each document, rank on E. Solutions for CRL can be used for DL with the addition of a CSA for T, setting $S = E$ and $\sigma = D$. Recall that Sadakane's [35] and Hon, Shah and Vitter's [20] solutions for DL implement access to E using a CSA and bit vector V on T, so they cannot be used for general CRL.

Our main contribution in this section is the observation that, using new data structures for access, color enumeration and rank, we can obtain interesting new bounds for both CRL and DL. This is formalized in the next theorem.

Theorem 1. *Suppose we are given a sequence $S[1, n]$ over $[1, \sigma]$ and we store any data structure supporting access on S in time t_{acc} and any structure sup-*

porting distinct enumeration in a range of S in time t_{enum} per element (and any structure supporting rank on S in time t_{rank} if computing frequencies is desired). Then later, given i and j, we can list the distinct elements in $S[i..j]$ in time $\mathcal{O}(t_{\mathsf{acc}} + t_{\mathsf{enum}})$ per reported element, plus $\mathcal{O}(t_{\mathsf{rank}})$ to list its frequency in $S[i..j]$.

Corollary 1. *Given a concatenation $T[1, n]$ of D documents, we can store either*

- *the CSA for T and data structures supporting access, enumeration and rank on the corresponding array $E[1, n]$ in times t_{acc}, t_{enum} and t_{rank}, or*
- *the CSA for T, a bit vector occupying $D \log(n/D) + \mathcal{O}(D) + o(n)$ bits, and data structures supporting enumeration and rank on E as above,*

such that, given a pattern of length m, we can list the distinct documents containing that pattern in time $\mathcal{O}(\mathsf{search}(m))$ plus $\mathcal{O}(t_{\mathsf{acc}} + t_{\mathsf{enum}} + t_{\mathsf{rank}})$ per reported document, where $t_{\mathsf{acc}} = \mathsf{lookup}(n)$ in the second case and t_{rank} is required only in order to list the frequencies of the documents.

A selection of these data structures is shown in Table 1. If we choose a set of rows covering support for access and enumeration (and rank) then we can answer CRL queries (and return the frequency of each color). The space bound is the sum of the space bounds and the time bound per reported color is $\mathcal{O}(t_{\mathsf{acc}} + t_{\mathsf{enum}} + t_{\mathsf{rank}})$, the latter term for computing frequencies. For example,

2+9: is Välimäki and Mäkinen's scheme [37].

1: is the scheme by Gagie, Puglisi, and Turpin [15].

3+9+10: combining Ferragina and Venturini's [11] data structure with Fischer's [12] succinct index for RMQ and Grossi, Orlandi and Raman's [19] succinct index for rank gives a solution for CRL that takes $nH_k(S) + 2n + o(n) \log \sigma + n\, o(\log \sigma)$ bits and $\mathcal{O}(1)$ time per reported color, matching the time of Muthukrishnan's $\mathcal{O}(n \log n)$-bit space solution [31]. The k-th order empirical entropy $H_k(S)$ measures the compressibility of S when we use contexts of length k; see [28] for details. The frequency of any color can be obtained in time $\mathcal{O}(\log \log \sigma)$.

6+9: is similar to the above but the $n\, o(\log \sigma)$ space term is avoided, as the structure by Barbay, Gagie, Navarro and Nekrich [4] computes rank as well. This becomes the *least-space* reported solution to CRL, listing in $\mathcal{O}(1)$ time.

(4 or 5)+9: combining Barbay et al.'s [4] access and rank data structure with Fischer's [12] succinct index for RMQ gives a solution for CRL that takes $nH_0(S) + 2n + o(n)(H_0(S) + 1)$ bits and $\mathcal{O}(\log \log \sigma)$ bits per reported color and its frequency (variant *4*), which is the *fastest* compressed solution when we want all the frequencies; or $\mathcal{O}(1)$ per reported color and $\mathcal{O}(\log \log \sigma \log \log \log \sigma)$ per reported frequency (variant *5*), which trades frequency reporting time for constant-time listing.

[35]+*9:* replacing Sadakane's [35] RMQ data structure with the one by Fischer [12] improves Sadakane's space bound by $2n$ bits.

[20]+*10:* replacing Hon, Shah and Vitter's [20] CSA_d structures by that of Grossi et al. [19] speeds up counting document frequencies (here $t_{\mathsf{acc}} = \mathsf{lookup}(n)$).

Table 1. Space and time bounds for some data structures supporting operations on $S[1,n]$ over $[1,\sigma]$. The $\mathcal{O}(\sigma \log n)$ extra bits of wavelet trees [18, 10] can be avoided [26] so we have not included it. The space bound in rows *3* and *6* holds for $k = o(\log_\sigma n)$. In rows *7* and *8*, g is the size (in bits) of a given context-free grammar generating S and only S and α is the inverse Ackermann function. The succinct index for RMQ in row *9* does not need access to the underlying data (i.e., C), but the succinct index for rank in row *10* does (i.e., S), hence the time of the latter depends on t_{acc}. Due to space constraints, here we write $\log^{[2]}$ and $\log^{[3]}$ for $\log\log$ and $\log\log\log$.

row	source	space (in bits)	t_{acc}	t_{enum}	t_{rank}
1	[18, 15]	$nH_0(S) + o(n)\log\sigma$	$\mathcal{O}(\log\sigma)$	$\mathcal{O}(\log\sigma)$	$\mathcal{O}(\log\sigma)$
2	[10, Cor. 3.3]	$nH_0(S) + o(n)\log\sigma$	$\mathcal{O}\left(1 + \frac{\log\sigma}{\log^{[2]} n}\right)$		$\mathcal{O}\left(1 + \frac{\log\sigma}{\log^{[2]} n}\right)$
3	[11]	$nH_k(S) + o(n)\log\sigma$	$\mathcal{O}(1)$		
4	[4, Thm. 1]	$nH_0(S) + o(n)(H_0(S)+1)$	$\mathcal{O}\left(\log^{[2]}\sigma\right)$		$\mathcal{O}\left(\log^{[2]}\sigma\right)$
5	[4, Thm. 1]	$nH_0(S) + o(n)(H_0(S)+1)$	$\mathcal{O}(1)$		$\mathcal{O}\left(\log^{[2]}\sigma \log^{[3]}\sigma\right)$
6	[4, Thm. 2]	$nH_k(S) + o(n)\log\sigma$	$\mathcal{O}(1)$		$\mathcal{O}\left((\log^{[2]}\sigma)^2 \log^{[3]}\sigma\right)$
7	[5, Thm. 1]	$\mathcal{O}(g\,\alpha(g))$	$\mathcal{O}(\log n)$		
8	[5, Thm. 1]	$\mathcal{O}(g)$	$\mathcal{O}\left(\log n \log^{[2]} n\right)$		
9	[12, Thm. 1]	$2n + o(n)$		$\mathcal{O}(1)$	
10	[19, Thm. 5(a)]	$n\,o(\log\sigma)$			$\mathcal{O}\left(t_{\mathsf{acc}} \log^{[2]}\sigma\right)$

The $|\mathsf{CSA}|$ space is exchanged by $n\,o(\log d)$ bits, which can be less or more. We can then also discard the D-bit string marking documents used by both solutions [35, 20] and replace it with rank queries on E.

(7 or 8)+9+10: combines Bille, Landau and Weimann's [5] grammar-based data structure for access, Fischer's [12] succinct index for RMQ, and Grossi et al.'s [19] succinct index for rank. González and Navarro [16] showed how to build a grammar generating an array that, together with some other small data structures, gives access to the suffix array (SA) A. Building Bille, Landau and Weimann's data structure for this grammar, we obtain a $\mathcal{O}(\log n)$-time data structure for DL whose size is bounded in terms of the high-order entropies of the library of documents. This is described next.

Theorem 2. *Given a concatenation $T[1,n]$ of D documents, we can store T in*

$$|\mathsf{CSA}| + 2n + o(n) + n\,o(\log D) + \mathcal{O}\left(\left(n\min(H_k(T),1) + D\right)\log\left(\frac{1}{\min(H_k(T),1) + D/n}\right)\alpha(n)\log n\right)$$

bits, for any $k \le \alpha\log_\tau n$, constant $0 < \alpha < 1$ and τ the size of the alphabet of T. Then given a pattern of length m, we can list the distinct documents containing that pattern in time $\mathcal{O}(\mathsf{search}(m))$ plus $\mathcal{O}(\log n)$ to list each document, plus $\mathcal{O}(\log n \log\log D)$ to give its frequency.

Proof. González and Navarro's algorithm takes advantage of the so-called *runs* of the SA, that is, areas $A[i..i+\ell]$ such that there is some other area $A[j..j+\ell]$ where $A[j+k] = A[i+k]+1$ for all $0 \le k \le \ell$. Let R be the number of runs with which the SA can be covered; it is known that $R \le \min(n, nH_k(T)+\sigma^k)$ for any k [25]. González and Navarro represent the SA differentially so that these areas become true repetitions, and use a grammar-based compression algorithm that represents A using at most $R\log(n/R)$ rules. We note that, in E, those SA runs become identical areas $E[i..i+\ell] = E[j..j+\ell]$ except for at most D cells where the document number can change when we advance one text position. It follows that, by applying the same compression algorithm [16] to E we obtain at most $(R+D)\log(n/(R+D))$ rules and hence the space given in the theorem. □

As a final note applying only to document collections, Sadakane's CSA [34] essentially represents a function Ψ such that $A[\Psi(i)] = A[i]+1$, which is stored in compressed form and any value computed in constant time. Thus one advances virtually in the text by successively applying Ψ. Now assume we sample E with a step r such that, for any i, $E[\Psi^j(i)]$ is sampled for some $0 \le j < r$. Then one computes any $E[i]$ value in time $\mathcal{O}(r)$ by following Ψ until hitting a sampled entry, whose value will be the same as $E[i]$ if we also sample every document end in the text collection. The space is $\mathcal{O}((n/r)\log r) + (n/r)\log D$ for a bitmap marking the sampled cells and an array with the sampled values, respectively. For example, using $r = \log D$ yields access to E (though not rank nor select on it) in the same time of a binary wavelet tree, within bit space $n+o(n)$. Depending on the relation between n and D, this can be an interesting alternative to using lookup and marking the document beginnings [35].

3 Top-k Queries

Improving the current-best solution for documents. Recently, Hon, Shah and Wu [21] described a data structure that stores a library T of D documents of total length n in $\mathcal{O}\big(n\log^2 n\big)$ bits such that later, given a pattern of length m and an integer $k \ge 1$, we can find the k documents that contain that pattern most frequently, in $\mathcal{O}(m + \log n \log\log n + k)$ time. We call this the document top-k problem (DTK). Hon, Shah and Vitter [20] gave solutions for DTK that store T in $\mathcal{O}(n\log n)$ bits and answer queries in $\mathcal{O}(m + k\log k)$ time, or in $2|\mathsf{CSA}| + o(n) + D\log(n/D) + \mathcal{O}(D)$ bits and $\mathcal{O}\big(\mathsf{search}(m) + k\log^{3+\epsilon} n \cdot \mathsf{lookup}(n)\big)$ time.

The last solution consists of a tree τ_k built for each k power of 2. For τ_k they divide E into blocks of size $z = k\log^{2+\epsilon} n$, and τ_k consists of the suffix tree nodes that are lowest common ancestors (lca) of end points of blocks, and transitively all the lcas of pairs of those nodes. At each node, τ_k stores the k most frequent documents within the whole blocks it contains, and their frequencies. Thus each τ_k requires $\mathcal{O}((n/z)k\log n) = \mathcal{O}\big(n/\log^{1+\epsilon} n\big)$ bits, and all the trees together add up to $\mathcal{O}(n/\log^{\epsilon} n)$ bits. At query time, to find the top-k documents in $E[i..j]$, they increase k to the next power of 2 and find the highest node of τ_k whose range $[i'..j']$ is contained in $[i..j]$. They show that $i'-i \le z$ and $j-j' \le z$ by

the *lca* properties. Then the query is answered by considering the k candidates given by τ_k and the $\mathcal{O}(z)$ further candidates found at positions of $E[i..i'-1]$ and $E[j'+1..j]$, for each of which they compute the frequency. The total time, considering priority queue operations, is $\mathcal{O}(\mathsf{search}(m) + z(t_{\mathsf{rank}} + \log k) + k \log k) = \mathcal{O}\big(\mathsf{search}(m) + k \log^{3+\epsilon} n \cdot \mathsf{lookup}(n)\big)$. This time bound can be improved to $\mathcal{O}\big(\mathsf{search}(m) + k \log D \log(D/k) \log^{1+\epsilon} n \cdot \mathsf{lookup}(n)\big)$ by noticing that (a) one needs only $\mathcal{O}(\log D)$ powers of 2 for k since $k \leq D$; (b) one can store the top-k elements in the τ_k trees and not their frequency. The k frequencies can be computed at query time without changing the time complexity since $k = o(z)$. Thus the k documents out of D can be stored in increasing order and as gamma-encoded differences, taking $\mathcal{O}(k \log(D/k))$ bits. Therefore we can use smaller blocks of size $z = k \log D \log(D/k) \log^{\epsilon} n$, which are processed faster, and still have $O(n/\log^{\epsilon} n) = o(n)$ space for the structure.

In addition, as shown in Section 2, by replacing the $|\mathsf{CSA}|$ bits of their solution for computing frequencies, by Grossi et al.'s [19] succinct index for rank, we achieve a new space bound of $|\mathsf{CSA}| + o(n) + D \log(n/D) + \mathcal{O}(D) + n\, o(\log D)$ bits, which can be better or worse than before, but the time is reduced to $\mathcal{O}(\mathsf{search}(m) + k \log D \log(D/k) \log^{\epsilon} n \cdot \mathsf{lookup}(n))$, for any ϵ (log-logarithmic terms disappear by adjusting ϵ).

An approximate solution to the general problem. We now give a solution to the approximate colored range top-k problem (CRTK), which asks us to preprocess a given sequence S such that later, given a range $S[i..j]$ and an integer $k \geq 1$, we can return an approximate list of the k elements ("colors") that occur most frequently in that range. We do not know of any previous efficient solutions to this problem, although finding the k most frequent or important items in various data sets and models is a well studied problem and there has been work on interesting special cases (see, e.g., [22, 24]).

Greve, Jørgensen, Larsen and Truelsen [17] recently gave a data structure that, for any $\epsilon > 0$, stores S in $\mathcal{O}((n/\epsilon) \log n)$ bits such that we can find an element such that no element is more than $1 + \epsilon$ times more frequent in $S[i, j]$, in $\mathcal{O}(\log(1/\epsilon))$ time. Thus, their data structure solves the approximate CRTK problem for $k = 1$, which is called the approximate range-mode problem. We can assume their data structure also returns the frequency of the approximate mode in $S[i..j]$, since adding a rank data structure for S allows us to compute this and does not change their space bound. We show how to use their data structure as a building block to store S in $\mathcal{O}((n/\epsilon)(H_0(S) + 1) \log n)$ bits such that, given an integer k, we can approximately list the k most common elements and their frequencies in $\mathcal{O}(k \log \sigma \log(1/\epsilon))$ time.

We first build a binary wavelet tree for S [18]. This is a balanced tree where each node represents a range of $[1, \sigma]$: the root represents the full range, the leaves the individual symbols, and the children of a node represent the left and right halves of the node's interval. For each node v, let S_v be the subsequence of S consisting of characters labelling the leaves in v's subtree. The original wavelet tree does not store S_v, but just a bitmap B_v of length $|S_v|$ telling whether each $S_v[i]$ went to the left or right child. Rank and select over those bitmaps allow

accessing any $S[i]$, as well as computing $\mathsf{rank}_a(S,i)$ and $\mathsf{select}_a(S,i)$, in time $\mathcal{O}(\log\sigma)$, and the overall space is $n\log\sigma(1+o(1))$. It can also track any range $S[i..j]$ down to any node [26].

Here we do store each subsequence S_v in an instance of Greve et al.'s approximate range-mode data structure. For now, assume $[i,j]=[1,n]$ and that Greve et al.'s data structure returns the exact mode, rather than an approximation. Notice that, if $a_1,\ldots,a_{k'}$ are the k' most frequent elements and v is an ancestor of the leaf labelled $a_{k'}$ but not of those labelled $a_1,\ldots,a_{k'-1}$, then $a_{k'}$ is the mode in S_v. Let V be the set of ancestors of $a_1,\ldots,a_{k'-1}$ and let V' be the set of nodes whose siblings are in V but who are not in V themselves; V' contains the root of the tree if V is empty. We can find $a_{k'}$ by finding the mode of S_v for each $v\in V'$, finding their frequencies in S, and taking the most frequent.

We keep the modes for each $v\in V'$ in a priority queue, ordered by their frequencies and with the corresponding nodes of the wavelet tree as auxiliary data. Notice $a_{k'}$ is the head of the queue, so we can find and output it in $\mathcal{O}(1)$ time; let v be the corresponding node, i.e., the node in V' such that the mode of S_v is $a_{k'}$. To update the queue, we delete $a_{k'}$, perform range-mode queries on the siblings of nodes on the path from v to the leaf labelled $a_{k'}$, and add the modes to the queue. There are always $\mathcal{O}(k\log\sigma)$ nodes in the queue (the tree is of height $\mathcal{O}(\log\sigma)$) so, if we use a priority queue allowing $\mathcal{O}(\log(k\log\sigma))=\mathcal{O}(\log\sigma)$ time deletion and $\mathcal{O}(1)$ time insertion [8], then we can find the k most frequent elements in S in $\mathcal{O}(k\log\sigma\log(1/\epsilon))$ time. We can deal with general i and j by using the wavelet tree to compute the appropriate range in each subsequence [26]. As for the approximation, it is clear that, whenever we output an element, none of the elements not output yet can be more than $1+\epsilon$ times more frequent.

If we use a Huffman-shaped wavelet tree, then calculation shows that our space usage is $\mathcal{O}((n/\epsilon)(H_0(S)+1)\log n)$ bits. However, since a Huffman tree can be very deep (height $n-1$ for a very skewed distribution), this would compromise our time bound. Therefore, we use a an $\mathcal{O}(\log\sigma)$-restricted Huffman tree [30], which yields both the space and time bounds we want.

Theorem 3. *Given a sequence $S[1,n]$ over an alphabet of size σ and a constant $\epsilon>0$, we can store S in $\mathcal{O}((n/\epsilon)(H_0(S)+1)\log n)$ bits such that, given i, j and k, we can list k distinct elements such that no element is more than $1+\epsilon$ times more frequent in $S[i..j]$ than any of the ones we list, in $\mathcal{O}(k\log\sigma\log(1/\epsilon))$ time.*

This $(1+\epsilon)$-approximation makes sense in information retrieval scenarios, where top-k queries are understood to be just approximations to the ideal answer.

Corollary 2. *Given a set of D documents of total length n and a constant $\epsilon>0$, we can store them in $\mathcal{O}((n/\epsilon)\log n\log D)$ bits such that, given a pattern of length m and k, we can list k distinct documents such that no document contains that pattern more than $1+\epsilon$ times as often as any of the ones we list, in a total of $\mathcal{O}(m+k\log D\log(1/\epsilon))$ time.*

Although Corollary 2 is weaker than Hon, Shah and Vitter's uncompressed result, our approach applies to the general colored range query problem, and may

be faster than what the upper bound suggests. For example, if the documents are webpages sorted lexicographically by URL, then it is more likely that interesting patterns will occur often in clusters of documents than widely spread out [36, 39]. In this case, leaves in a balanced wavelet tree for E that are labelled with the k distinct documents that contain the pattern most often, are likely to share many ancestors; if so, our data structure can speed up to $\mathcal{O}(m + k \log k \log(1/\epsilon))$.

The K-mining problem. Muthukrishnan [31] defined (document) K-mining (DKM) as the problem of finding all the documents in the library that contain a given pattern at least K times. He gave an $\mathcal{O}\big(n \log^2 n\big)$-bit data structure that, given K and a pattern of length m, answers queries in $\mathcal{O}(m)$ time plus $\mathcal{O}(1)$ time per reported document. Hon, Shah and Wu [21] noted that we can use binary search with a DTK data structure to solve DKM, with an $\mathcal{O}(\log n)$ slowdown for the queries. They then showed how we can use an $\mathcal{O}\big(n \log^2 n\big)$-bit data structure to find the largest k such that k documents contain the pattern K times, in $\mathcal{O}(\mathsf{search}(m) + \log n \log\log n)$ time. Hon, Shah and Vitter [20] gave an $\mathcal{O}(n \log n)$-bit data structure that answers K-mine queries in time $\mathcal{O}(m)$ plus $\mathcal{O}(1)$ per reported document. They also showed how to improve the space bound to $2|\mathsf{CSA}| + o(n) + D \log(n/D)$ bits at the cost of increasing the time $\mathcal{O}\big(\mathsf{search}(m) + k \log^{3+\epsilon} n \cdot \mathsf{lookup}(n)\big)$, which we can improve similarly as before. Neither of these solutions applies to general colored range queries, however.

Since our CRTK data structure outputs elements in (approximately) non-increasing order by frequency in the range, it also solves (approximately) the natural generalization of DKM: i.e., the colored range K-mine (CRKM) problem, which asks us to report all the elements that occur at least K times in $S[i..j]$. If we query our data structure until the next element it would report occurs fewer than $(1 + \epsilon)K$ times, then we use $\mathcal{O}(\log \sigma \log(1/\epsilon))$ time per reported element, but we may miss some elements that occur between K and $(1 + \epsilon)K$ times. Alternatively, if we query our data structure until the next element it would report occurs fewer than $K/(1 + \epsilon)$ times, then we find all the elements that occur at least K times, but we can bound our time only in terms of the number of elements that occur at least $K/(1 + \epsilon)$ times.

4 Counting

Given a wavelet tree for the array C we described in Section 2, and positions i and j, it is not difficult to count the number of values less than i in $C[i..j]$ [26], which is the number of distinct elements in $S[i..j]$ [31]. The wavelet tree for C takes $\mathcal{O}(n \log n)$ bits and does this counting in time proportional to its height, $\mathcal{O}(\log n)$. This already matches the best known solution, due to Bozanis, Kitsios, Makris and Tsakalidis [6]. In the rest of this section we show how to reduce the space bound to $n \log \sigma + \mathcal{O}(n \log\log n)$ bits.

Theorem 4. *We can represent a sequence $S[1, n]$ over alphabet $[1, \sigma]$ in $n \log \sigma + \mathcal{O}(n \log\log n)$ bits of space so as to count the number of distinct elements in any interval $S[i..j]$ in time $\mathcal{O}(\log n)$.*

Proof. Our structure represents $C[1,n]$ using a wavelet tree. We have already explained how to attain the given time bound. The remaining problem is that the wavelet tree for C requires $n\log n(1+o(1))$ bits. We reduce the space to $n\log\sigma+\mathcal{O}(n\log\log n)$ as follows. Note that each symbol $c\in[1,\sigma]$ that appears at positions $c_1<c_2<\ldots<c_{n_c}$, $S[c_1]=S[c_2]=\ldots=S[c_{n_c}]=c$, induces a *chain* in C of the form $C[c_1]=0$, $C[c_2]=c_1$, $C[c_3]=c_2$, ..., $C[c_{n_c}]=c_{n_c-1}$. Now consider the middle point $n/2$ of C. For any c, let us call m_c the last value such that $c_{m_c}<n/2$. Then for any c and any $k\le m_c$ it holds $C[c_k]<n/2$, and for any $k>m_c$ it holds $C[c_k]\ge n/2$. Thus $C[c_{m_c+1}]=c_{m_c}\ge n/2$ and $C[c_{m_c}]<n/2$, and $i=m_c$ is the only value satisfying this for c. Thus all the sequence values are $C[i]<n/2$ for $i<n/2$. For $i\ge n/2$ there are at most σ positions $i=m_c\ge n/2$ such that $C[m_c]<n/2$, and all the rest are $C[i]\ge n/2$. Thus there are at most σ positions in C where $C[i]<n/2$ and $C[i+1]\ge n/2$, and at most σ positions where $C[i]\ge n/2$ and $C[i+1]<n/2$. Since the root bitmap B_v satisfies $B_v[i]=0$ iff $C[i]<n/2$, there are at most σ transitions from 0 to 1 in B_v, and at most σ transitions from 1 to 0. Both children of v may contain the σ subsequences and thus each may contain up to σ transitions again. Thus, there are at most $2^d\sigma$ 0/1 and 1/0 transitions among all the bitmaps at depth d of the wavelet tree.

For $d\ge\log(n/\sigma)$ this upper bound is useless, so we may assume that bitmaps at depths $\log(n/\sigma)$ to $\log n-1$ are incompressible. These add up to $n(\log n-\log(n/\sigma))=n\log\sigma$ bits, plus $o(n\log\sigma)$ to provide *rank* and *select* capabilities to those bitmaps. For smaller d, we introduce a compression scheme. Consider the concatenation B_d of all the bitmaps at depth d. Then B_d contains at most $2^d\sigma$ runs of 0s and $2^d\sigma$ runs of 1s. We represent B_d using two sparse bitmaps. A bitmap $R_d[1,n]$ will mark with a 1 the beginning of each run of 0s or 1s. Let $o_1,o_2,\ldots$ the lengths of the runs of 1s. A second bitmap $O_d[1,rank_1(B_d,n)]$ will have a 1 at positions 1, $1+o_1$, $1+o_1+o_2$, ... Then $\mathsf{rank}_1(B_d,i)$ can be computed as follows. First we compute $x=\mathsf{rank}_1(R_d,i)$. Because $C[i]<i$, the first run of B_d is a 0-run, thus if x is odd then i is within a 0-run and otherwise within a 1-run. If x is odd, then we must count the 1s in the first $(x-1)/2$ 1-runs of B_d, that is, $\mathsf{rank}_1(B_d,i)=\mathsf{select}_1(O_d,(x+1)/2)-1$. If, instead, x is even, then we must count the 1s in the first $x/2-1$ 1-runs and add the 1s in the current run. This is $\mathsf{rank}_1(B,d_i)=\mathsf{select}_1(O_d,x/2)+i-select_1(R_d,x)$.

We represent R_d with Raman et al.'s technique [33]. If R_d has m 1s, then the representation takes $m\log\frac{n}{m}+\mathcal{O}(m)+o(n)$ bits. At level d we have $m\le 2^d\sigma$, thus R_d requires at most $2^d\sigma\log\frac{n}{2^d\sigma}+\mathcal{O}\big(2^d\sigma\big)+o(n)$ bits (that $o(n)$ is $\mathcal{O}(n\log\log n/\log n)$). Added over all the compressible levels we have $\sum_{d=0}^{\log(n/\sigma)-1}2^d\sigma\log\frac{n}{2^d\sigma}+\mathcal{O}\big(2^d\sigma\big)+o(n)=\mathcal{O}(n)+o(n\log(n/\sigma))$.

Analogously, the O_d bitmap takes $\mathcal{O}(n)+o(n\log(n/\sigma))$ bits. Added to the incompressible levels, we have $n\log\sigma+o(n\log n)$ bits of space, or more precisely, $n\log\sigma+\mathcal{O}(n\log\log n)$. The preprocessing time is the same as for a classical wavelet tree over alphabet $[0,n-1]$. □

On the other hand, the array C can also provide access to S as follows. Sample the tth occurrence of each color c, say at $S[i]=c$, in a bitmap $B[1,n]$, that is

$B[i] = 1$, and store the samples at $W[rank_1(B, i)] = c$. Then, we can find out any $S[j]$ without storing S by repeatedly asking whether $B[i] = 1$, $B[C[i]] = 1$, $B[C[C[i]]] = 1$, and so on until finding a sampled value, in time $\mathcal{O}(t \log n)$. The extra space is $n + o(n) + \mathcal{O}((n/t) \log \sigma)$, so we can set $t = \mathcal{O}(\log^{\epsilon} n)$ for any constant $\epsilon > 0$ to make it $n + o(n) \log \sigma$. Therefore, our representation *replaces* S, as it can compute any $S[i]$ in time $\mathcal{O}\big(\log^{1+\epsilon} n\big)$. Its space occupancy, $n \log \sigma + o(n) \log \sigma + \mathcal{O}(n \log \log n)$, makes the representation *succinct* (i.e., $|S|(1 + o(1))$ bits) whenever σ is more than polylogarithmic in n.

Theorem 4, applied over sequence $S = E$, lets us compute document frequencies for arbitrary patterns. Find the suffix array interval $\mathsf{CSA}[i..j]$ corresponding to the pattern, and then count the different values in $E[i..j]$. For this particular case, however, there is a better solution [35] using $2n + o(n)$ bits and constant time, yet it does not generalize to colored range counting. On the other hand, since our representation provides access to E in time $t_{\mathsf{acc}} = \mathcal{O}\big(\log^{1+\epsilon} n\big)$, it can be regarded within the framework of Section 2.

5 Further Applications to Information Retrieval

We have presented new and efficient solutions for three natural colored range queries: colored range listing, colored range top-k queries, and colored range counting. Our solutions for colored range listing lead to the fastest compressed data structures for that problem and for document listing; our (approximate) solution for colored range top-k queries is, as far as we know, the first efficient data structure for that problem; and our solution for colored range counting reduces the space bound from $\mathcal{O}(n \log n)$ bits to $n \log \sigma + \mathcal{O}(n \log \log n)$ bits while maintaining $\mathcal{O}(\log n)$ query time. Although our solutions for colored range top-k queries and colored range counting do not give improved bounds for the corresponding document retrieval problems, our more general data structures may find applications to other information retrieval scenarios beyond ranges induced by searching for exact patterns in suffix trees or arrays.

A simple example of natural queries not fitting in the restricted model are lexicographic range queries. Imagine we look for patterns lexicographically in the range [`"1969"`, `"2010"`] in documents; the result is a suffix array range that does not correspond to any suffix tree node. In this case, existing techniques for document retrieval based on suffix tree properties (such as for computing top-k queries [20] and for computing document frequencies [35]) will not work. The general techniques we have introduced in this article do.

Yet another scenario that is not captured by the suffix tree model is inverted indices for natural language text (as opposed to the general texts addressed in this paper) [3]. Consider that we store the list of documents where each vocabulary word appears, consecutively according to the order of the words in the vocabulary. If queries are simple words, then all the document retrieval problems we have considered are easily solved by storing the documents of each list ordered by decreasing term frequency. Yet, imagine we wish to provide *also* the same functionality on stemmed searching, upon user request at query time.

One solution is to group together the vocabulary words sharing the same stem so that, while individual word queries can be handled as usual, stemmed queries are handled by considering the concatenation of the lists of the words sharing the same stem. Then we can regard the concatenation of all inverted lists as the array E and use the general techniques developed in this paper to answer various document queries on stems: Document listing and counting algorithms apply verbatim, while those involving frequencies pose further challenges as each entry in the inverted lists is weighted by the term frequency of the word in the document. Other query operations, from case folding to thesauri expansion, can also be reduced to a proper grouping of lists.

Finally, there are information retrieval scenarios completely different from the text search framework. For example, colored range queries seem a natural tool for query mining [2], where logs of queries posed to search engines are recorded over periods of time, and then analyzed to discover trends in user behavior. By considering that each different query is a color, we can find the most popular queries or the number of distinct queries within any given time period, among many other potential queries of interest, which could in turn become new challenging colored range queries.

References

1. A. Apostolico. The myriad virtues of subword trees. In *Combinatorial Algorithms on Words*, NATO ISI Series, pp. 85–96. Springer-Verlag, 1985.
2. R. Baeza-Yates. Applications of web query mining. In *Proc. ECIR*, pp. 7–22, 2005.
3. R. Baeza-Yates and B. Ribeiro. *Modern Information Retrieval*. AW, 1999.
4. J. Barbay, T. Gagie, G. Navarro, and Y. Nekrich. Alphabet partitioning for compressed rank/select with applications. Technical Report 0911.4981, arXiv, 2010.
5. P. Bille, G. M. Landau, and O. Weimann. Random access to grammar compressed strings. Technical Report 1001.1565, arXiv, 2010.
6. P. Bozanis, N. Kitsios, C. Makris, and A. K. Tsakalidis. New upper bounds for generalized intersection searching problems. In *Proc. ICALP*, pp. 464–474, 1995.
7. G. S. Brodal, B. Gfeller, A. G. Jørgensen, and P. Sanders. Towards optimal range medians. *Theoretical Computer Science*. To appear.
8. S. Carlsson, J. I. Munro, and P. V. Poblete. An implicit binomial queue with constant insertion time. In *Proc. SWAT*, pp. 1–13, 1988.
9. J. S. Culpepper, G. Navarro, S. J. Puglisi, and A. Turpin. Top-k ranked document search in general text databases. In *Proc. ESA*, 2010. To appear.
10. P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. Compressed representations of sequences and full-text indexes. *ACM Transactions on Algorithms (TALG)*, 3(2):article 20, 2007.
11. P. Ferragina and R. Venturini. A simple storage scheme for strings achieving entropy bounds. *Theoretical Computer Science*, 371(1):115–121, 2007.
12. J. Fischer. Optimal succinctness for range minimum queries. In *Proc. LATIN*, pp. 158–169, 2010.
13. J. Fischer and V. Heun. A new succinct representation of RMQ-information and improvements in the enhanced suffix array. In *Proc. ESCAPE*, pp. 459–470, 2007.
14. H. N. Gabow, J. L. Bentely, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. STOC*, pp. 135–143, 1984.

15. T. Gagie, S. J. Puglisi, and A. Turpin. Range quantile queries: Another virtue of wavelet trees. In *Proc. SPIRE*, pp. 1–6, 2009.
16. R. González and G. Navarro. Compressed text indexes with fast locate. In *Proc. CPM*, pp. 216–227, 2007.
17. M. Greve, A. G. Jørgensen, K. D. Larsen, and J. Truelsen. Cell probe lower bounds and approximations for range mode. In *Proc. ICALP*, pp. 605–616, 2010.
18. R. Grossi, A. Gupta, and J. S. Vitter. High-order entropy-compressed text indexes. In *Proc. SODA*, pp. 636–645, 2003.
19. R. Grossi, A. Orlandi, and R. Raman. Optimal trade-offs for succinct string indexes. In *Proc. ICALP*, pp. 678–689, 2010.
20. W.-K. Hon, R. Shah, and J. Vitter. Space-efficient framework for top-k string retrieval problems. In *Proc. FOCS*, pp. 713–722, 2009.
21. W.-K. Hon, R. Shah, and S.-B. Wu. Efficient index for retrieving top-k most frequent documents. In *Proc. SPIRE*, pp. 182–193, 2009.
22. I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-K query processing techniques in relational database systems. *ACM Computing Surveys*, 40(4), 2008.
23. R. Janardan and M. A. Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry and Applications*, 3(1):39–69, 1993.
24. M. Karpinski and Y. Nekrich. Top-K color queries for document retrieval. Technical Report 1007.1361, arXiv, 2010.
25. V. Mäkinen and G. Navarro. Succinct suffix arrays based on run-length encoding. *Nordic Journal of Computing*, 12(1):40–66, 2005.
26. V. Mäkinen and G. Navarro. Rank and select revisited and extended. *Theoretical Computer Science*, 387(3):332–347, 2007.
27. U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.
28. G. Manzini. An analysis of the Burrows-Wheeler transform. *Journal of the ACM*, 48(3):407–430, 2001.
29. Y. Matias, S. Muthukrishnan, S. C. Sahinalp, and J. Ziv. Augmenting suffix trees, with applications. In *Proc. ESA*, pp. 67–78, 1998.
30. R. L. Milidiú and E. S. Laber. Bounding the inefficiency of length-restricted prefix codes. *Algorithmica*, 31(4):513–529, 2001.
31. S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proc. SODA*, pp. 657–666, 2002.
32. G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):article 2, 2007.
33. R. Raman, V. Raman, and S. Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proc. SODA*, pp. 233–242, 2002.
34. K. Sadakane. New text indexing functionalities of the compressed suffix arrays. *Journal of Algorithms*, 48(2):294–313, 2003.
35. K. Sadakane. Succinct data structures for flexible text retrieval systems. *Journal of Discrete Algorithms*, 5(1):12–22, 2007.
36. F. Silvestri. Sorting out the document identifier assignment problem. In *Proc. ECIR*, pp. 101–112, 2007.
37. N. Välimäki and V. Mäkinen. Space-efficient algorithms for document retrieval. In *Proc. CPM*, pp. 205–215, 2007.
38. P. Weiner. Linear pattern matching algorithm. In *Proc. IEEE Symp. on Switching and Automata Theory*, pp. 1–11, 1973.
39. H. Yan, S. Ding, and T. Suel. Inverted index compression and query processing with optimized document ordering. In *Proc. WWW*, pp. 401–410, 2009.