Bottom-k Document Retrieval

Gonzalo Navarro[1] and Sharma V. Thankachan[2]

[1] Dept. of Computer Science, Univ. of Chile, gnavarro@dcc.uchile.cl
[2] Georgia Institute of Technology, USA, sharma.thankachan@gmail.com

Abstract. We consider the problem of retrieving the k documents from a collection of strings where a given pattern P appears *least* often. This has potential applications in data mining, bioinformatics, security, and big data. We show that adapting the classical linear-space solutions for this problem is trivial, but the compressed-space solutions are not easy to extend. We design a new solution for this problem that matches the best-known result when using $2|\mathsf{CSA}| + o(n)$ bits, where CSA is a Compressed Suffix Array. Our structure answers queries in the time needed by the CSA to find the suffix array interval of the pattern plus $O(k \lg k \lg^{\epsilon} n)$ accesses to suffix array cells, for any constant $\epsilon > 0$.

1 Introduction

The problem of *top-k document retrieval* in general strings is that of preprocessing a collection of strings (called "documents") so that, given a search pattern $P[1,m]$ (a short string) and a threshold k, we retrieve the k documents where P appears most often. It is a direct extension of widely used concepts in Information Retrieval (IR) [4, 1] to strings, where it finds applications in IR on East Asian languages (which lack clear word separators), bioinformatics, multimedia IR, software development, and others [19]. Much research has been carried out on this topic in recent years. A foundational result, by Hon et al. [16], achieved a linear-space index, that is, using $O(n \lg n)$ bits on a string collection of total length n, and $O(m + k \lg k)$ time, where m is the length of P. This was later improved to the optimal $O(m + k)$ time [21]. Optimality under more relaxed conditions has also been obtained [28].

The use of linear space ($O(n \lg n)$ bits), however, is not fully satisfactory. In practice, these indexes use several times the space needed to represent the raw collection, which renders them impractical when indexing large string collections. There has been a good deal of research on top-k document retrieval using reduced space. Compressed Suffix Arrays (CSAs) are data structures that can find all the individual occurrences of P in the collection, using space asymptotically equal not only to $n \lg \sigma$ bits (where σ is the alphabet size of the strings), but to the size of the compressed text collection [20]. Moreover, those CSAs can retrieve any substring of any document and hence replace the collection: they can be regarded as compressors that support queries. We call their space $|\mathsf{CSA}|$, which can be thought of as the minimum space in which the text collection can be represented. The first solution for the top-k document retrieval problem using

CSAs was in the same paper by Hon et al. [16], who use $2|\mathsf{CSA}| + o(n)$ bits and solve queries in time $O(t_{\mathsf{search}}(m) + k\, t_{\mathsf{SA}} \lg^{3+\epsilon} n)$, for any constant $\epsilon > 0$, where $t_{\mathsf{search}}(m)$ is the time needed by the CSA to count the number of times P occurs in the collection and t_{SA} is the time to retrieve the position of one such occurrence. After several improvements [11, 2], Hon et al. [15] achieved the best time to date within that space, $O(t_{\mathsf{search}}(m) + k\, t_{\mathsf{SA}} \lg k \lg^{\epsilon} n)$. Tsur [29] reduced the space to the asymptotically optimal $|\mathsf{CSA}| + o(n)$ bits, yet with higher time, $O(t_{\mathsf{search}}(m) + k\, t_{\mathsf{SA}} \lg k \lg^{1+\epsilon} n)$. Finally, Navarro and Thankachan [22] reduced the time to $O(t_{\mathsf{search}}(m) + k\, t_{\mathsf{SA}} \lg^2 k \lg^{\epsilon} n)$, the best to date within optimal space.

In this paper we focus on a variant of this problem, which we dub *bottom-k document retrieval*, where we seek the k documents where P appears *least* often. This is useful to solve a smoothed ranked version of the query P_1 *and not* P_2 [10], to intersect documents where P_1 appears frequently with those where P_2 appears infrequently (such smoothing is used in search engines when the exact Boolean meaning of the query does not return sufficiently many results).

Finding documents where a pattern appears infrequently also arises as a subproblem of, for example, infrequent itemset mining [13, 12, 5], with applications in statistical disclosure risk assessment (where rare patterns in anonymized census data can lead to statistical disclosure), bioinformatics (where rare patterns in microarray data may suggest genetic disorders), or fraud detection (where rare patterns in financial or tax data may suggest unusual activity associated with fraudulent behavior). It is also useful for studying negative association rules in data mining [32, 7], for finding adverse drug effects in medicine [17], for intrusion detection in computer systems [25], for detecting anomalies in system logs [3], etc. Our problem also arises as a subproblem of finding rare motifs (patterns with don't cares of various kinds) [9], which has potential applications to bioinformatic problems such as identifying absent words in genomic sequences [14] or proteins involved in parasitism [30].

Adapting the current linear-space solutions [16, 21, 28] to this scenario is trivial, as they are able to handle weighting functions more general than frequencies: It is sufficient to use the negative term frequencies. Instead, adapting the compressed indexes, which work only for frequencies, turns out to be challenging. We build on the ideas of Hon et al. [16, 15] to develop a compressed bottom-k document retrieval index that uses $2|\mathsf{CSA}|+o(n)$ bits and answers queries in time $O(t_{\mathsf{search}}(m) + k\, t_{\mathsf{SA}} \lg k \lg^{\epsilon} n)$, matching the best known time for top-k retrieval using the same space.

2 Compressed Top-k Retrieval Indexes

Consider a collection of D strings $\{T_1, T_2, \ldots, T_D\}$ over an alphabet $[1, \sigma]$, called *documents*, concatenated into a text $T[1, n] = T_1\$T_2\$ \ldots T_D\$$, where \$ is a special symbol smaller than all the others in $[1, \sigma]$ (e.g., $\$ = 0$). Consider the suffix tree [31] of T, the suffix array [18] $A[1, n]$ of T, and a corresponding Compressed Suffix Array [20] CSA. A CSA must be able to (1) given a pattern $P[1, m]$, find the area $A[sp, ep]$ of suffixes starting with P, in time we call $t_{\mathsf{search}}(m)$, and (2)

given a position i, compute $A[i]$, in time we call t_{SA}. We describe the path to the solution which we will build on for the *top-k retrieval problem*: given a pattern $P[1,m]$, return the k documents where P appears most often.

Given a suffix tree node v, we denote by $path(v)$ the string obtained by concatenating labels of all edges on the path from the root to v. The *locus* of a string P is the highest node v such that P is a prefix of $path(v)$. Each suffix tree leaf (or suffix array cell) can be associated with the document T_d where the corresponding suffix starts. We call $\mathsf{tf}(v,d)$ the number of leaves associated with document d that descend from suffix tree node v (i.e., the number of times $path(v)$ appears in document d). Then the top-k retrieval problem can be solved by first finding the locus v of pattern P, and then retrieving the k documents d with highest $\mathsf{tf}(v,d)$ values. Note that the problem could be solved by attaching the answer to any suffix tree node, but the space would be $O(kn \lg n)$ bits, and work only up to the chosen k value. Now we describe the solutions we build on to obtain our result.

Hon, Shah and Vitter. Hon et al.'s [16] structure is built (in principle) for a fixed k value. We choose a grouping factor $b = k \lg^{2+\epsilon} n$ and *mark* every bth leaf in the suffix tree (we use a slightly simplified description of their method [24]). Then we mark the lowest common ancestor (LCA) of every consecutive pair of marked leaves. The tree of marked nodes is called τ_k and has $O(n/b)$ nodes. For every marked suffix tree node v, we store the k pairs $(d, \mathsf{tf}(v,d))$ with highest $\mathsf{tf}(v,d)$. Hon et al. prove that any locus node v contains one maximal marked node u so that there are at most $2b$ leaves covered by v but not by u (we will denote that leaf set by $v \setminus u$). Therefore they traverse the leaves in $v \setminus u$ using the CSA, and for each one they (1) compute the corresponding document d, (2) compute the frequency $\mathsf{tf}(v,d)$, (3) add d to the top-k list (or correct its frequency from $\mathsf{tf}(u,d)$ to $\mathsf{tf}(v,d)$ if d was already stored in the precomputed top-k list of u).

To carry out (1) on the ith suffix tree leaf, they first compute $A[i]$ in $O(t_{\mathsf{SA}})$ time, and then convert it into a document number by storing a bitmap $B[1,n]$ that marks with a 1 the document's beginnings in T [27]. So the document is $d = \mathsf{rank}(B, A[i])$, where $\mathsf{rank}(B,j)$ counts the number of 1s in $B[1,j]$. Since B has D 1s, it can be represented using $D \lg(n/D) + O(D) + o(n)$ bits, which is $o(n)$ if $D = o(n)$, and answer rank queries in constant time [26]. To carry out (2) they need an additional $|\mathsf{CSA}|$ bits (see Sadakane [27]), and time $O(t_{\mathsf{SA}} \lg n)$. The node $u \in \tau_k$ is found using the CSA plus a constant-time LCA on τ_k for the leftmost and rightmost marked leaves in $[sp, ep]$, whereas the leaves covered by v are simply $[sp, ep]$. Thus the total query time is $O(t_{\mathsf{search}}(m) + b\, t_{\mathsf{SA}} \lg n) = O(t_{\mathsf{search}}(m) + k\, t_{\mathsf{SA}} \lg^{3+\epsilon} n)$.

As storing the top-k list needs $O(k \lg n)$ bits, the space for τ_k is $O((n/b)k \lg n) = O(n/\lg^{1+\epsilon} n)$ bits. One τ_k tree is stored for each k power of 2, so that at query time we increase k to the next power of 2 and solve the query within the same time complexity. Summed over all the powers of 2, the space becomes $O(n/\lg^{\epsilon} n) = o(n)$ bits. Therefore the total space is $2|\mathsf{CSA}| + o(n)$ bits.

Several subsequent technical improvements further reduced the time. Gagie et al. [11] noted that the powers of 2 for k must reach only D, and that one does not need to store the frequencies of the top-k precomputed answers (they can be computed at query time without increasing the time complexity). Only the identifiers of the documents need be stored, in increasing order. This allowed them to use a smaller $b = k \lg D \lg(D/k) \lg^\epsilon n$, which impacts query time. Further, Belazzougui et al. [2] showed that sublinearity is still obtained with the smaller block size $b = k \lg k \lg(D/k) \lg^\epsilon n$, and also reduced the $O(t_{\mathsf{SA}} \lg n)$ time of the binary search to $O(t_{\mathsf{SA}} \lg \lg n)$ by using a sampled predecessor search structure. The final query time obtained was $O(t_{\mathsf{search}}(m) + k\, t_{\mathsf{SA}} \lg k \lg(D/k) \lg^\epsilon n)$.

Hon, Shah, Thankachan and Vitter. Hon et al. [15] obtained the fastest solution to date using $2|\mathsf{CSA}| + o(n)$ bits of space. They consider two block values, $c < b$. For block value b they build the τ_k trees as before. For block value c they build another set of marked trees ρ_k. These trees are finer-grained than the τ_k trees. Now, given the locus node v, there exists a maximal node $w \in \rho_k$ contained in v, and a maximal node $u \in \tau_k$ contained in w. The key idea is to build a list of top-k to top-$2k$ candidates by joining the precomputed results of w and u, and then correct this result by traversing $O(c)$ suffix tree leaves.

Since we have a maximal node $u \in \tau_k$ contained in any node $w \in \rho_k$, we can encode the top-k list of w only for the documents that are not already in the top-k list of u. Note that a document must appear at least once in $w \setminus u$ if it is in the top-k list of w but not in that of u. Thus the additional top-k candidates of w can be encoded using $O(k \lg(b/k))$ bits, by storing as before one of their positions in $w \setminus u$, and encoding the sorted positions differentially.

The space for a τ_k tree is $O((n/b)k \lg n) = O(n/\lg^{1+\epsilon} n)$ bits using $b = k \lg^{2+\epsilon} n$, which added over all the powers of 2 for k gives $O(n/\lg^\epsilon n) = o(n)$ bits, as before. For the ρ_k trees they require $O((n/c)k \lg(b/k))$ bits, which using $c = k \lg k \lg^\epsilon n$ gives $O(n \lg \lg n/(\lg k \lg^\epsilon n))$ bits. Added over the powers of 2 for k this gives $O(n \lg \lg n/\lg^\epsilon n) \sum_{i=1}^{\lg D} 1/i = O(n \lg \lg n \lg \lg D/\lg^\epsilon n) = o(n)$ bits.

The time is dominated by that of traversing $O(c)$ cells. Using some speedups [2] over the basic technique [16], the time is $O(t_{\mathsf{SA}} \lg \lg n)$ per cell, for a total of $O(t_{\mathsf{search}}(m) + k\, t_{\mathsf{SA}} \lg k \lg^\epsilon n)$ for any constant $\epsilon > 0$.

3 Bottom-k Document Retrieval

We build upon the scheme of Hon et al. [16] and subsequent improvements [11, 2]. We choose a grouping factor b and mark every bth leaf in the suffix tree. Then we mark the lowest common ancestor (LCA) of every consecutive pair of marked leaves. For each marked node u, we will store a particular set of bottom-k results.

Definition 1. *Let u be a marked node. Let u^* be its lowest marked ancestor, and let u' be the child of u^* that is an ancestor of u (or u itself). Then the* fringe *of u,* $\mathsf{fringe}(u)$*, is the set of documents corresponding to the set of leaves $u' \setminus u$.*

Hon et al. [16] prove that $\mathsf{fringe}(u)$ contains $O(b)$ nodes (or, similarly, $|v \setminus u| \leq 2b$, where u is the highest marked node that descends from a node v). The bottom-k answer we will store associated with u is defined as follows (cf. [6, 8]).

Definition 2. *The* bottom-k answer set *of u, $\mathsf{bottom}(u)$, is a set of k documents that appear the least number of times in the leaves of u, among those that, additionally, do not appear in the fringe of u.*

Now we prove that the answers for any locus node v between u and u' (i.e., u is the highest marked descendant of v) are in $\mathsf{bottom}(u)$ or in $\mathsf{fringe}(u)$. Note that there are several correct answer sets for a given query; we say each such answer is *valid*.

Lemma 1. *There is a valid answer to the bottom-k query for locus node v that contains only nodes in $\mathsf{bottom}(u) \cup \mathsf{fringe}(u)$.*

Proof. We build an answer $bot(v)$ by choosing k documents with lowest positive frequency in v, from the set $\mathsf{bottom}(u) \cup \mathsf{fringe}(u)$. If this is not a valid answer, then there exists a document $d \notin bot(v)$ that appears below v between 1 and $f - 1$ times, where f is the maximum frequency occuring below node v of a document in $bot(v)$. Then d cannot appear in $\mathsf{bottom}(u) \cup \mathsf{fringe}(u)$, as otherwise it would have been chosen to form $bot(v)$. But this implies that d was not chosen to form $\mathsf{bottom}(u)$ even when it did not appear in $\mathsf{fringe}(u)$, therefore there are k other documents with frequency at most $f - 1$ below u. (As they do not appear in the fringe of u, the frequencies below v are the same as below u for d and for $\mathsf{bottom}(u)$.) Therefore, there are k documents ($\mathsf{bottom}(u)$) with frequency between 1 and $f-1$ below v. Then the k least frequencies in $\mathsf{bottom}(u) \cup \mathsf{fringe}(u)$ cannot be higher than $f - 1$. A contradiction. □

Therefore, we can answer a bottom-k query with locus node v as follows. Figure 1 illustrates the scenario.

1. Find the highest marked descendant u of v.
2. Initialize the answer set with $\mathsf{bottom}(u)$.
3. Move upwards from u, one by one, until reaching another marked node, u^*.
4. Compute $[s_{u'}, e_{u'}]$, where u' is the last traversed node before reaching u^*.
5. For each leaf in $[s_{u'}, e_{u'}] \setminus [s_u, e_u]$, compute the frequency of the document below v (i.e., in $[s_v, e_v]$), and update the answer if necessary.

The way to store the candidate set, to traverse the tree, and to compute the frequencies, is exactly as in previous work [16, 11, 2]. This yields a first result with $2|\mathsf{CSA}| + o(n)$ bits of space and query time $O(t_{\mathsf{search}}(m) + k\, t_{\mathsf{SA}} \lg k \lg(D/k) \lg^{\epsilon} n)$. In the next section we improve on this technique to obtain our final result.

4 A Dual Marking Scheme

Now we improve the result of Section 3 by combining it with the dual marking scheme of Hon et al. [15]. We will have a tree τ_k formed by the nodes marked

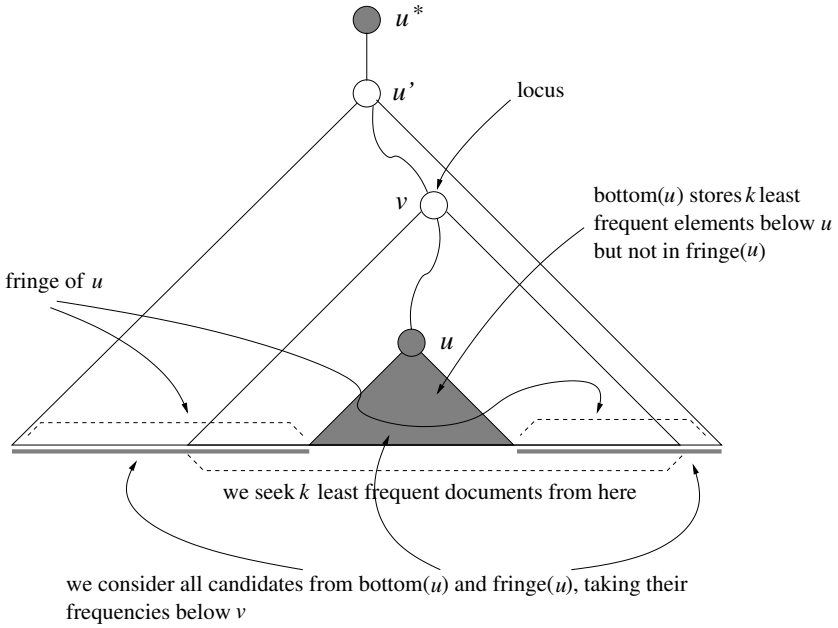

Fig. 1. The relationships between the nodes and leaf sets that participate in the query process.

with sampling step b, and a denser tree ρ_k formed by the nodes marked with sampling step $c < b$. The concept of fringe for nodes $w \in \rho_k$ will remain as in Definition 1, whereas for the nodes $u \in \tau_k$ we will use a slightly different concept.

Definition 3. *Let $u \in \tau_k$, which is marked with sampling step b. Let $w \in \rho_k$ be its lowest ancestor marked with sampling step c. Then the* τ-fringe *of u, τ-*$\mathsf{fringe}(u)$*, is the set of documents corresponding to the set of leaves of $w \setminus u$.*

Definition 4. *The* τ-bottom-k answer set *of $u \in \tau_k$, τ-*$\mathsf{bottom}(u)$*, is a set of k documents that appear the least number of times in the leaves of u, among those that, additionally, do not appear in the τ-fringe of u.*

For nodes $u \in \tau_k$, we will store τ-$\mathsf{bottom}(u)$, in the same way as in Section 3. For nodes $w \in \rho_k$, we will store $\mathsf{bottom}(w)$. This second set will be stored in the following way, where u is the highest descendant of w that is in τ_k:

- We will store k bits indicating which documents of τ-$\mathsf{bottom}(u)$ belong to $\mathsf{bottom}(w)$. In the case of ties we give preference to documents in τ-$\mathsf{bottom}(u)$.
- The elements in $\mathsf{bottom}(w) \setminus \tau$-$\mathsf{bottom}(u)$ are stored using a position of $w \setminus u$ to identify them, in increasing position order and using gap encoding.

In this way, the storage of $\mathsf{bottom}(w)$ requires $O(k \lg(b/k)) = O(k \lg \lg n)$ bits. Note that we are making the following assumption, which we prove next.

Lemma 2. *Any element in* $\mathsf{bottom}(w) \setminus \tau\text{-}\mathsf{bottom}(u)$ *must appear at least once in* $w \setminus u$.

Proof. Assume there exists $d \in \mathsf{bottom}(w) \setminus \tau\text{-}\mathsf{bottom}(u)$ for which there is no occurrence in $w \setminus u$. Then the frequency f of d below w is the same as below u. But since $d \notin \tau\text{-}\mathsf{bottom}(u)$ and $d \notin \tau\text{-}\mathsf{fringe}(u)$, there must exist k documents ($\tau\text{-}\mathsf{bottom}(u)$, precisely) with frequency at most f below u (and below w, as they do not appear in $\tau\text{-}\mathsf{fringe}(u)$). Since in the case of ties we give preference to $\tau\text{-}\mathsf{bottom}(u)$, d cannot be chosen from $\mathsf{bottom}(w) \setminus \tau\text{-}\mathsf{bottom}(u)$. □

The query process is exactly as in Section 3, but using the sampled tree ρ_k. The tree τ_k is only used internally in order to reconstruct $\mathsf{bottom}(w)$. The total amount of work is proportional to the sampling step, $O(c\, t_{\mathsf{SA}} \lg \lg n)$. Similarly to Hon et al. [15], we define $c = k \lg k \lg^{\epsilon} n$ to obtain our final result.

Theorem 1. *The bottom-k document retrieval problem, on a collection of D strings with concatenated length n, for a pattern of length m, can be solved using* $2|\mathsf{CSA}| + D \lg(n/D) + O(D) + o(n)$ *bits and in* $O(t_{\mathsf{search}}(m) + k\, t_{\mathsf{SA}} \lg k \lg^{\epsilon} n)$ *time, for any constant $\epsilon > 0$. Here* CSA *is a compressed suffix array over the collection,* $t_{\mathsf{search}}(m)$ *is the time the* CSA *takes to find the suffix array interval of the pattern, and* t_{SA} *is the time it takes to retrieve any suffix array cell. The space is* $2|\mathsf{CSA}| + o(n)$ *bits if* $D = o(n)$.

5 A Negative Result

There exist solutions [29, 22] for the top-k problem that use the asymptotically optimal $|\mathsf{CSA}| + o(n)$ bits, with time $O(t_{\mathsf{search}}(m) + k\, t_{\mathsf{SA}} \lg^2 k \lg^{\epsilon} n)$. These solutions are based on precomputing all the documents that belong to the top-k answer for some locus node v that lies between u and u' (using the terminology of Definition 1), and storing them associated with u. This is possible because there are only $O(\sqrt{bk})$ candidates along a fringe of size $O(b)$ [2, 29].

It is tempting to try to adapt this same technique to the bottom-k document retrieval problem. However, when we consider bottom-k answers, the number of candidates can be of size $\Theta(b)$, as we show in the following example.

Assume the sequence of documents in the fringe of u is $d_2 d_1 d_3 d_2 d_4 d_3 d_4 \ldots d_i d_{i-1} d_{i+1} d_i \ldots d_b d_{b-1} d_b$, and assume that each suffix of this sequence corresponds to the leaves below a locus node v between u and u' (in addition v contains the leaves of u). Then, the bottom-1 answer for the whole fringe is d_1, the only one with frequency 1. The bottom-1 answer after removing $d_2 d_1$ is d_2, the only one with frequency 1. In general, the bottom-1 answer for the suffix starting in $d_i d_{i-1} d_{i+1} d_i \ldots$ is d_{i-1}, and for the suffix starting in $d_{i+1} d_i \ldots$ is d_i. Thus, we have b different candidates to answers for a fringe of size $2b - 1$.

6 Conclusions

We have addressed a new problem called bottom-k document retrieval, which asks to find k documents where a pattern appears least frequently. We have

matched the best result for the much studied top-k (most frequent) retrieval problem, in the case of using $2|\mathsf{CSA}| + o(n)$ bits of space. We have also shown that a relatively direct extension of the current ideas [29, 22] for achieving asymptotically optimal space, $|\mathsf{CSA}| + o(n)$ bits, does not work for bottom-k queries.

We have not considered the case where one can use *compact space*, that is, $|\mathsf{CSA}| + n \lg D + o(n \lg D)$ bits. The best result for the top-k problem in this case is $O(t_{\mathsf{search}}(p) + k \lg^* n)$ [23]. The kind of result we have obtained in this paper suggests that a similar result can probably be reached in compact space.

Acknowledgements. Funded in part by Fondecyt Grant 1-140796.

References

1. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 2nd edition, 2011.
2. D. Belazzougui, G. Navarro, and D. Valenzuela. Improved compressed indexes for full-text document retrieval. *J. Discr. Alg.*, 18:3–13, 2013.
3. L. Burns, J.L. Hellerstein, S. Ma, C.S. Perng, D.A. Rabenhorst, and D. Taylor. A systematic approach to discovering correlation rules for event management. In *Proc. 7th IFIP/IEEE Intl. Symp. on Integrated Network Management*, 2001.
4. S. Büttcher, C. Clarke, and G. Cormack. *Information Retrieval: Implementing and Evaluating Search Engines*. MIT Press, 2010.
5. L. Cagliero and P. Garza. Infrequent weighted itemset mining using frequent pattern growth. *IEEE Trans. Knowl. Data Eng.*, 99(PrePrints), 2013.
6. T. M. Chan, S. Durocher, M. Skala, and B. Wilkinson. Linear-space data structures for range minority query in arrays. In *Proc. SWAT*, pages 295–306, 2012.
7. X. Dong. Mining interesting infrequent and frequent itemsets based on minimum correlation strength. In *Proc. AICI*, LNAI 7002, pages 437–443, 2011.
8. S. Durocher, R. Shah, M. Skala, and S.V. Thankachan. Linear-space data structures for range frequency queries on arrays and trees. In *Proc. MFCS*, pages 325–336, 2013.
9. T. El-Falah, T. Lecroq, and M. Elloumi. Extraction of infrequent simple motifs from a finite set of sequences using a lattice structure. *Recent Pat. DNA Gene Seq.*, 7(2):123–127, 2013.
10. J. Fischer, T. Gagie, T. Kopelowitz, M. Lewenstein, V. Mäkinen, L. Salmela, and N. Välimäki. Forbidden patterns. In *Proc. 10th LATIN*, LNCS 7256, pages 327–337, 2012.
11. T. Gagie, J. Kärkkäinen, G. Navarro, and S.J. Puglisi. Colored range queries and document retrieval. *Theoret. Comput. Sci.*, 483:36–50, 2013.
12. A. Gupta, A. Mittal, and A. Bhattacharya. Minimally infrequent itemset mining using pattern-growth paradigm and residual trees. *CoRR*, 1207.4958, 2012.
13. D. Haglin and A. Manning. On minimal infrequent itemset mining. In *Proc. DMIN*, pages 141–147, 2007.
14. J. Herold, S. Kurtz, and R. Giegerich. Efficient computation of absent words in genomic sequences. *BMC Bioinformatics*, 9:167, 2008.
15. W.-K. Hon, R. Shah, S. Thankachan, and J. Vitter. Faster compressed top-k document retrieval. In *Proc. 23rd DCC*, pages 341–350, 2013.
16. W.-K. Hon, R. Shah, and J. Vitter. Space-efficient framework for top-k string retrieval problems. In *Proc. 50th FOCS*, pages 713–722, 2009.

17. Y. Ji, H. Ying, J. Tran, P. Dews, A. Mansour, and R.M. Massanari. A method for mining infrequent causal associations and its application in finding adverse drug reaction signal pairs. *IEEE Trans. Knowl. Data Eng.*, 25(4):721–733, 2013.
18. U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. *SIAM J. Comp.*, 22(5):935–948, 1993.
19. G. Navarro. Spaces, trees and colors: The algorithmic landscape of document retrieval on sequences. *ACM Comp. Surv.*, 46(4):art. 52, 2014.
20. G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Comp. Surv.*, 39(1):art. 2, 2007.
21. G. Navarro and Y. Nekrich. Top-k document retrieval in optimal time and linear space. In *Proc. 23rd SODA*, pages 1066–1078, 2012.
22. G. Navarro and S. Thankachan. Faster top-k document retrieval in optimal space. In *Proc. 20th SPIRE*, LNCS 8214, pages 255–262, 2013.
23. G. Navarro and S. Thankachan. Top-k document retrieval in compact space and near-optimal time. In *Proc. 24th ISAAC*, LNCS 8283, pages 394–404, 2013.
24. G. Navarro and D. Valenzuela. Space-efficient top-k document retrieval. In *Proc. 11th SEA*, pages 307–319, 2012.
25. A. Rahman, C.I. Ezeife, and A.K. Aggarwal. WiFi Miner: An online apriori-infrequent based wireless intrusion system. In *Proc. Knowledge Discovery from Sensor Data*, LNAI 5840, pages 76–93, 2010.
26. R. Raman, V. Raman, and S. Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Trans. Alg.*, 3(4):art. 43, 2007.
27. K. Sadakane. Succinct data structures for flexible text retrieval systems. *J. Discr. Alg.*, 5:12–22, 2007.
28. R. Shah, C. Sheng, S. V. Thankachan, and J. Vitter. Top-k document retrieval in external memory. In *Proc. 21st ESA*, LNCS 8125, pages 803–814, 2013.
29. D. Tsur. Top-k document retrieval in optimal space. *Inf. Proc. Lett.*, 113(12):440–443, 2013.
30. C. Vens, E. Danchin, and M.N. Rosso. Identifying proteins involved in parasitism by discovering degenerated motifs. In *Proc. 4th International Workshop on Machine Learning in Systems Biology*, pages 81–84, 2010.
31. P. Weiner. Linear pattern matching algorithm. In *Proc. 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.
32. X. Wu, C. Zhang, and S. Zhang. Efficient mining of both positive and negative association rules. *ACM Trans. Inf. Syst.*, 22(3):381–405, 2004.