Top-k Document Retrieval in External Memory*

Rahul Shah[1], Cheng Sheng[2], Sharma V. Thankachan[1] and Jeffrey Scott Vitter[3]

[1] Louisiana State University, USA. {rahul,thanks}@csc.lsu.edu
[2] The Chinese University of Hong Kong, China. csheng@cse.cuhk.edu.hk
[3] The University of Kansas, USA. jsv@ku.edu

Abstract. Let $\mathcal{D}$ be a given set of (string) documents of total length n. The top-k document retrieval problem is to index $\mathcal{D}$ such that when a pattern P of length p, and a parameter k come as a query, the index returns those k documents which are most relevant to P. Hon et al. [22] proposed a linear space framework to solve this problem in $O(p+k \log k)$ time. This query time was improved to $O(p+k)$ by Navarro and Nekrich [33]. These results are powerful enough to support arbitrary relevance functions like frequency, proximity, PageRank, etc. Despite of continued progress on this problem in terms of theoretical, practical and compression aspects, any non-trivial bounds in external memory model have so far been elusive. In this paper, we propose the first external memory index supporting top-k document retrieval queries (outputs unsorted) in optimal $O(p/B+\log_B n+k/B)$ I/Os, where B is the block size. The index space is almost linear $O(n \log^* n)$ words, where $\log^* n$ is the iterated logarithm of n. We also improve the existing internal memory results. Specifically, we propose a linear space index for retrieving top-k documents in $O(k)$ time, once the locus of the pattern match is given.

1 Introduction and Related Work

The inverted index is the most fundamental data structure in the field of information retrieval [43]. It is the backbone of every known search engine today. For each word in any document collection, the inverted index maintains a list of all documents in that collection which contain the word. Despite its power to answer various types of queries, the inverted index becomes inefficient, for example, when queries are phrases instead of words. This inefficiency results from inadequate use of word orderings in query phrases [37]. Similar problems also occur in applications when word boundaries do not exist or cannot be identified deterministically in the documents, like genome sequences in bioinformatics and text in many East-Asian languages. These applications call for data structures to answer queries in a more general form, that is, (string) pattern matching. Specifically, they demand the ability to identify efficiently all the documents that contain a specific pattern as a substring. The usual inverted-index approach might require the maintenance of document lists for all possible substrings of the documents.

* Work supported by National Science Foundation (NSF) Grants CCF–1017623 (R. Shah and J. S. Vitter) and CCF–1218904 (R. Shah).

This can take quadratic space and hence is neither theoretically interesting nor sensible from a practical viewpoint.

The first frameworks for answering document retrieval queries were proposed by Matias et al. [30] and Muthukrishnan [31]. Their data structures solve the *document listing problem*, where the task is to index a collection $\mathcal{D}$ of D documents, such that whenever a pattern P of length p comes as a query, report all those documents containing P exactly once. Muthukrishnan also initiated the study of relevance metric-based document retrieval [31], which was then formalized by Hon et al. [22] as follows:

Problem 1 (Top-k document retrieval problem) *Let $w(P, d)$ be the score function capturing the relevance of a pattern P with respect to a document d. Given a document collection $\mathcal{D}$= $\{d_1, d_2, .., d_D\}$ of D documents, build an index answering the following query: given P and k, find k documents with the highest $w(P, .)$ values in its sorted (or unsorted) order.*

Here, instead of reporting all the documents that match a query pattern, the problem is to output the k documents most relevant to the query in sorted order of relevance score. Relevance metrics considered in the problem can be either pattern-independent (e.g., PageRank) or -dependent. In the latter case one can take into account information like the frequency of the pattern occurrences (or *term-frequency* of popular tf-idf measure, which takes the number of occurrences of P in a document d as $w(P, d)$) and even the locations of the occurrences (e.g.,*min-dist* [22] which takes proximity of two closest occurrences of pattern as the score). In general, we assume that other than a static weight which is fixed for each document d, $w(P, d)$ is dependent only on the set of occurrences of P in d. The framework of Hon et al. [22] takes linear space and answers the query in $O(p + k \log k)$ time. This was then improved by Navarro and Nekrich [33] to achieve $O(p + k)$ query cost. Both [22] and [33] reduced this problem to a 4-sided orthogonal range query in 3d, which is defined as follows: the data consists of a set S of 3-dimensional points and the query consists of four parameters x', x'', y' and z', and output is the set of all those points $(x_i, y_i, z_i) \in S$ such that $x_i \in [x', x'']$, $y_i \leq y'$ and $z_i \geq z'$. While general 4-sided orthogonal range searching is proved hard [9], the desired bounds can nevertheless be achieved by identifying a special property that one dimension of the reduced subproblem can only have p distinct values. Even though there has been series of work on top-k string, including in theory as well as practical IR [5, 10, 11, 13, 15, 17–25, 33–35, 37, 38, 42] communities, most implementations (as well as theoretical results) have focused on RAM based compressed and/or efficient indexes (See [32] for an excellent survey). In this paper, we introduce an alternative framework for solving this problem and obtain the first non-trivial external memory [3] solution as follows:

Theorem 1 *In the external memory model, there exists an $O(nh)$-word structure that solves the top-k (unsorted) document retrieval problem in $O(p/B + \log_B n + \log^{(h)} n + k/B)$ I/Os for any $h \leq \log^* n$, where $\log^{(h)} n = \log \log^{(h-1)} n$, $\log^{(1)} n = \log n$ and B is the block size.*

For $h = \log^* n$, $\log^{(h)} n$ is a constant, and hence we have the following result.

Corollary 1 *There exists an $O(n \log^* n)$-word structure for answering top-k (unsorted) document retrieval problem in optimal $O(p/B + \log_B n + k/B)$ I/Os.*

Our framework can also be used for improving the existing internal memory results. Specifically, we shall derive a linear space index for retrieving top-k documents in $O(k)$ time, once the locus of the pattern match is given. This result improves the previous work [22, 33] by eliminating the additive term p. In applications like cross-document pattern matching [26], the locus can be computed in much faster $O(\log \log p)$ time than $O(p)$. In many pattern matching applications, for example in suffix-prefix overlap [41], maximal substring matches [27], and autocompletion search (like in Google Instant^{TM}) multiple loci are searched with amortized constant time for each locus. In such situations, having extra $O(p)$ as in [22, 33] leads to inefficient solutions, where as our index can support faster queries. The result is summarized as follows:

Theorem 2 *There exists an $O(n)$ word space data structure in word RAM model for solving (sorted) top-k document retrieval problem in $O(k)$ time, once the locus of the pattern match is given.*

A related but somewhat orthogonal line of research has been to get top-k queries on general array based ranges. In this, we are given array A of colors with each color is assigned a score, and for a range query (i, j), we have to output k highest scored colors in this range (with each color reported at most once). If the scoring criteria is based on frequency, for example say score of a color is its number of occurrences in $A[i..j]$, then lower-bounds on range-mode problem [8, 16] would imply no efficient (linear space and polylog time) data structures can exist. There are variants considered where each entry in the array has a fixed score or each color (document) has a fixed score, independent of number of occurrences. Recent [29] surprising result of achieving optimal I/Os with $O(n \log^* n)$ space has been for 3-sided categorical range reporting where each entry has another attribute called score, and the query specifies range as well as score threshold. We are supposed to output all colors whose at least one entry within the range satisfies the score criteria. There are easier variants where each entry of the same color gets the same score attribute like PageRank which have been shown to have efficient external memory results [36]. There are even simpler variants, where only top-k scores are to be reported [2, 28, 39] without considering colors or unique colors are to be reported without considering scores (as in *document listing*). Both these variants lead to 3-sided queries which are easier to solve in external memory. In internal memory, there exists optimal space and time data structures for outputting these scores in the sorted order [7].

2 Preliminary: Top-k Framework

This section briefly explains the linear space framework for top-k document retrieval based on the work of Hon et al. [22], and Navarro and Nekrich [33]. The generalized suffix tree (GST) of a document collection $\mathcal{D} = \{d_1, d_2, d_3, \ldots, d_D\}$ is

the combined compact trie (a.k.a. Patricia trie) of all the non-empty suffixes of all the documents. Use n to denote the total length of all the documents, which is also the number of the leaves in GST. For each node u in GST, consider the path from the root node to u. Let $depth(u)$ be the number of nodes on the path, and $prefix(u)$ be the string obtained by concatenating all the edge labels of the path. For a pattern P that appears in at least one document, the *locus* of P, denoted as u_P, is the node closest to the root satisfying that P is a prefix of $prefix(u_P)$. By numbering all the nodes in GST in the pre-order traversal manner, the part of GST relevant to P (i.e., the subtree rooted at u_P) can be represented as a range.

Nodes are marked with documents. A leaf node ℓ is marked with a document $d \in \mathcal{D}$ if the suffix represented by ℓ belongs to d. An internal node u is marked with d if it is the lowest common ancestor of two leaves marked with d. Notice that a node can be marked with multiple documents. For each node u and each of its marked documents d, define a *link* to be a quadruple $(origin, target, doc, score)$, where $origin = u$, $target$ is the lowest proper ancestor[4] of u marked with d, $doc = d$ and $score = w\big(prefix(u), d\big)$. Two crucial properties of the links identified in [22] are listed below.

Lemma 1 *For each document d that contains a pattern P, there is a unique link whose origin is in the subtree of u_P and whose target is a proper ancestor of u_P. The score of the link is exactly the score of d with respect to P.*

Lemma 2 *The total number of links is $O(n)$.*

Based on Lemma 1, the top-k document retrieval problem can be reduced to the problem of finding the top-k links (according to its score) stabbed by u_P, where *link stabbing* is defined as follows:

Definition 1 (Link Stabbing) *We say that a link is* stabbed *by node u if it is originated in the subtree of u and targets at a proper ancestor of u.*

If we order the nodes in GST as per the *pre-order traversal* order, these constraints translate into finding all the links (i) the numbers of whose origins fall in the number range of the subtree of u_P, and (ii) the numbers of whose targets are less than the number of u_P. Regarding constraint (i) as a two-sided range constraint on x-dimension, and regarding constraint (ii) as a one-sided range constraint on y-dimension, the problem asks for the top-k weighted points that fall in a three-sided window in 2d space, where weight of a point is the score of the corresponding link [33].

3 External Memory Structures

This section is dedicated for proving Theorem 1. The initial phase of pattern search can be performed in $O(p/B + \log_B n)$ I/O's using a string B-tree [12]. Once the suffix range of P is identified, we take the lowest common ancestor of

[4] Define a dummy node as the parent of the root node, marked with all the documents.

the left-most and right-most leaves in the suffix range of GST to identify the locus node u_P. Hence, the first phase (i.e., finding the locus node u_P of P) takes optimal I/O's and now we focus only on the second phase (i.e., reporting the top-k links stabbed by u_P). Instead of solving the top-k version, we first solve a threshold version in Sec 3.1 where the objective is to retrieve those links stabbed by u_P with *score* at least a given threshold τ. Then in Sec 3.2, we propose a separate structure that converts the original top-k-form query into a threshold-form query so that the structure in Sec 3.1 can now be used to answer the original problem. Finally, we obtain Theorem 1 via bootstrapping on a special structure for handling top-k queries in lesser number of I/Os for small values of k. We shall assume all scores are distinct and are within $[1, O(n)]$. Otherwise, the ties can be broken arbitrarily and reduce the values into rank-space.

3.1 Breaking Down Into Sub-Problems

Instead of solving the top-k version, we first solve a threshold version, where the objective is to retrieve those links stabbed by u_P with *score* at least a given threshold τ. We show that the problem can be decomposed into simpler subproblems, which consists of a 3d dominance reporting and $O(\log(n/B))$ 3-sided range reporting in 2d, both can be solved efficiently using known structures. The main result is captured in Lemma 3 defined below. From now onwards, the origin, target and score of a link L_i are represented by o_i, t_i and w_i respectively.

Lemma 3 *There exists an $O(n)$ space data structure for answering the following query: given a query node u_P and a threshold τ, all links stabbed by u_P with score $\geq \tau$ can be reported in $O(\log^2(n/B) + z/B)$ I/Os, where z is the number of outputs.*

Rank and Components For any node u in GST, we use u to denote its pre-order rank as well. Let $size(u)$ denotes the number of leaves in the subtree of u, then we define its *rank* as:

$$rank(u) = \lfloor \log \lceil \frac{size(u)}{B} \rceil \rfloor$$

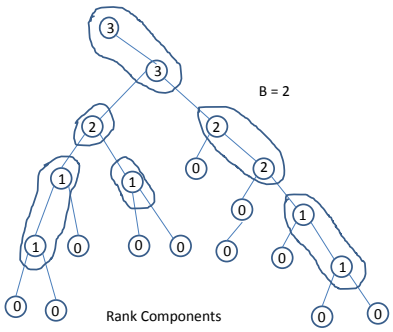

Fig. 1. Rank Components

Note that $rank(.) \in [0, \lfloor \log \lceil \frac{n}{B} \rceil \rfloor]$. A contiguous subtree consisting of nodes with the same rank is defined as a *component*, and the *rank* of a component is same as the rank of nodes within it (see figure 1). Therefore, a *component* with $rank = 0$ is a bottom level subtree of size (number of leaves) at most B. From the definition, it can be seen that a node and at most one of its children can have the same *rank*. Therefore, a component with $rank \geq 1$ consists of nodes in a path which goes top-down in the tree.

The number of links originating within the subtree of any node u is at most $2size(u) - 1$. Therefore, the number of links originating within a component with $rank = 0$ is $O(B)$. These $O(B)$ links corresponding to each component with $rank = 0$ can be maintained separately as a list, taking total $O(n)$ words space. Now, given a locus node u_P, if $rank(u_P) = 0$, the number of links originating within the subtree of u_P is also $O(B)$ and all of them can be processed in $O(1)$ I/O's by simply scanning the list of links corresponding to the component to which u_P belongs to. The query processing is more sophisticated when $rank(u_P) \geq 1$. For handling this case, we classify the links into the following 2 types based on the $rank$ of its target with respect to the $rank$ of query node u_P:

1. *equi-ranked links*: links with $rank(target) = rank(u_P)$
2. *high-ranked links*: links with $rank(target) > rank(u_P)$

Next we show that the problem of retrieving outputs among equi-ranked links can be reduced to a 3d dominance query, and the problem of retrieving outputs among high-ranked links can be reduced to at most $\lfloor \log \lceil \frac{n}{B} \rceil \rfloor$ 3-sided range queries in 2d.

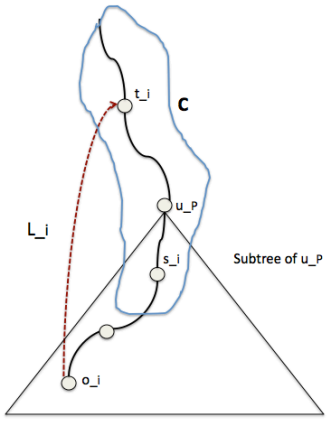

Fig. 2. Pseudo Origin

Processing equi-ranked links Let C be a component and S_C be set of all links L_i, such that its target t_i is a node in C. Also, for any link $L_i \in S_C$, let pseudo_origin s_i be the (pre-order rank of) lowest ancestor of its origin o_i within C (see Figure 2). Then a link $L_i \in S_C$ originates in the subtree of any node u within C if and only if $s_i \geq u$. Now if the locus u_P is a node in C, then among all equi-ranked links, we need to consider only those links $L_i \in S_C$, because the origin o_j of any other equi-ranked link $L_j \notin S_C$, will not be in the subtree of u_P. Based on the above observations, all equi-ranked output links are those $L_i \in S_C$ with $t_i < u_P \leq s_i$ and $w_i \geq \tau$. To solve this in external memory, we treat each link $L_i \in S_C$ as a 3d point (t_i, s_i, w_i) and maintain a 3d dominance query structure over it. Now the outputs with respect to u_P and τ are those links corresponding to the points within $(-\infty, u_P) \times [u_P, \infty) \times [\tau, \infty)$. Such a structure for S_C can be maintained in linear $O(|S_C|)$ words of space and can answer the query in $O(\log_B |S_C| + z_{eq}/B)$ I/O's using the result by Afshani [1], where $|S_C|$ is the number of points (corresponding to links in S_C) and z_{eq} be the output size. Thus overall these structures occupies $O(n)$-word space.

Lemma 4 *Given a query node u_P and a threshold τ, all the equi-ranked links stabbed by u_P with score $\geq \tau$ can be retrieved in $O(\log_B n + z_{eq}/B)$ I/Os using an $O(n)$ word space data structure, where z_{eq} is the output size.* □

Processing high-ranked links The following is an important observation.

Observation 1 *Any link L_i with its origin o_i within the subtree of a node u is stabbed by u if $rank(t_i) > rank(u)$, where t_i is the target of L_i.*

This implies, while looking for the outputs among the high-ranked links, the condition of t_i being a proper ancestor of u_P can be ignored as it is taken care of automatically if $o_i \in [u_P, u'_P]$, where u'_P be the (pre-order rank of) rightmost leaf in the subtree rooted at u_P. Let G_r be the set of all links with rank equals r for $1 \leq r \leq \lfloor \log \lceil \frac{n}{B} \rceil \rfloor$. Since there are only $O(\log(n/B))$ sets, we shall maintain separate structures for links in each G_r by considering only *origin* and *score* values. We treat each link $L_i \in G_r$ as a $2d$ point (o_i, w_i), and maintain a 3-sided range query structure over them for $r = 1, 2, .., \lfloor \log \lceil \frac{n}{B} \rceil \rfloor$. All high-ranked output links can be obtained by retrieving those links in $L_i \in G_r$ with the corresponding point $(o_i, w_i) \in [u_P, u'_P] \times [\tau, \infty]$ for $r = rank(u_P)+1, .., \lfloor \log \lceil \frac{n}{B} \rceil \rfloor$. By using the linear space data structure in [4], the space and I/O bounds for a particular r is given by $O(|G_r|)$ words and $O(\log_B |G_r| + z_r/B)$, where z_r is the number of output links in G_r. Since a link can be a part of at most one G_r, the total space consumption is $O(n)$ words and the total query I/Os is $O(\log_B n \log(n/B) + z_{hi}/B) = O(\log^2(n/B) + z_{hi}/B)$, where z_{hi} represents the number of high-ranked output links.

Lemma 5 *Given a query node u_P and a threshold τ, all the high-ranked links stabbed by u_P with score $\geq \tau$ can be retrieved in $O(\log^2(n/B) + z_{hi}/B)$ I/Os using an $O(n)$ word space data structure, where z_{hi} is the output size.* □

By combining Lemma 4 and Lemma 5, we obtain Lemma 3.

3.2 Converting Top-k to Threshold via Logarithmic Sketch

Here we derive a linear space data structure, such that given a query node u and a parameter k, a threshold τ can be computed in constant I/Os, such that the number of links z stabbed by u with score $\geq \tau$ is bounded by, $k \leq z \leq 2k + O(\log n)$. Hence query I/Os in Lemma 3 can be modified as $O(\log^2(n/B) + z/B) = O(\log^2(n/B) + k/B)$. From the retrieved z outputs, the actual top-k answers can be computed by selection [6, 40] and filtering in another $O(z/B) = O(k/B + \log_B n)$ I/O's. We summarize our result in the following lemma.

Lemma 6 *There exist an $O(n)$ word data structure for answering the following query in $O(\log^2(n/B) + k/B)$ I/O's: given a query point u and an integer k, report the top-k links stabbed by u.* □

The details of top-k to threshold conversion are given below.

Marked nodes and Prime Nodes in GST We identify certain nodes in the *GST* as marked nodes and prime nodes with respect to a parameter g called the *grouping factor*. The procedure starts by combining every g consecutive leaves (from left to right) together as a group, and marking the lowest common ancestor (LCA) of first and last leaf in each group. Further, we mark the LCA of all pairs of marked nodes recursively. Additionally, we ensure that the root is always

marked. At the end of this procedure, the number of marked nodes in GST will be $O(n/g)$ [22]. Prime nodes are those which are the children of marked nodes [5]. Corresponding to any marked node u^* (except root), there is a *unique prime* node u', which is its closest prime ancestor. In case u^*'s parent is marked then $u' = u^*$. For every prime node u' with atleast one marked node in its subtree, the corresponding closest marked descendant u^* is unique. If u' is marked then the closest marked descendant u^* is same as u'.

Hon et al. [22] showed that, given any node u with u^* being its highest marked descendent (if it exists), the number of leaves in the subtree of u, but not in the subtree of u^* (which we call as fringe leaves) is at most $2g$. This means for a given threshold τ, if z is the number of outputs corresponding to u^* as the locus node, then the number of outputs corresponding to u as the locus is within $z \pm 2g$. This is because of the fact that the number of documents d with $w\big(prefix(u), d\big) \neq w\big(prefix(u^*), d\big)$ cannot be more than the number of fringe leaves. Therefore, we maintain the following information at every marked node u^*: the score of $q-$th highest scored link stabbed by u^* for $q = 1, 2, 4, 8, ...$ By choosing $g = \log n$, the total space can be bounded by $O((n/g)\log n) = O(n)$ words, and can retrieve any particular entry in $O(1)$ time.

Using the above values, the threshold τ corresponding to any given u and k can be computed as follows: first find the highest marked node u^* in the subtree of u ($u^* = u$ if u is marked). Now identify i such that $2^{i-1} < k + 2g \leq 2^i$ and choose τ as the score of 2^i-th highest scored link stabbed by u^*. This ensures that $k \leq z < 2k + O(g) = 2k + O(\log n)$.

3.3 Special Structures for Bounded k

In this section, we derive a faster data structures for the case when k is upper bounded by a parameter g. The main idea is to identify smaller sets of $O(g)$ links, such that top-g links stabbed by any node u are contained in one of such sets. Thus by constructing the structure described in Lemma 6 over the links in each such sets, the top-k queries for any $k \leq g$ can be answered faster as follows:

Lemma 7 *There exists a $O(n)$ word data structure for answering top-k queries for $k \leq g$ in $O(\log^2(g/B) + k/B)$ I/O's.*

Recall the definitions of marked nodes and prime nodes from Sec 3.2. Let u' be a prime node and u^* (if it exists) be the unique highest marked descendent of u' by choosing a grouping factor g (which will be fixed later). All the links originated from the subtree of u' are categorized into the following (Figure 3).

- *near-links:* The links which are stabbed by u^*, but not by u'.
- *far-link:* The links which are stabbed by both u^* and u'.
- *small-link:* The links which are stabbed neither by u^*, nor by u'.
- *fringe-links:* The links originated not from the subtree of u^*.

Lemma 8 *The number of* fringe-links *and the number of* near-links *of any prime node u' is $O(g)$.*

[5] Note that the number of prime nodes can be $\Theta(n)$ in the worst case.

Proof. The number of leaves in $subtree(u')\backslash subtree(u^*)$ is at most $2g$ [22]. Therefore, the number of *fringe-links* can be bounded by $O(g)$. For every document d whose link originates from $subtree(u^*)$ going out of it ends up as a *near-link* if and only if d exists at one of the leaves of $subtree(u')\backslash subtree(u^*)$. Thus, this can also be bounded by $O(g)$. In the case where u^* does not exist for u', only fringe-links exist. More over the subtree size of u' is $O(g)$ there can be no more than $O(g)$ of these links. □

Consider the following set, consisting of $O(g)$ links with respect to u': all *fringe-links*, *near-links* and g highest scored *far-links*. We maintain these *links* at u' (as a data structure to be explained later). For any node u, whose closest prime ancestor (including itself) is u', the above mentioned set is called *candidate links* of u. From each u, we maintain the pointer to its closest prime ancestor where the set of *candidate links* is stored.

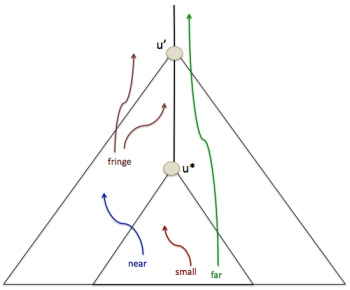

Fig. 3. Categorization of Links

Lemma 9 *The* candidate links *of any node u contains top-g highest scored links stabbed by u.*

Proof. Let u' be the closest prime ancestor of u. If no marked descendant of u' exist, then all the links are stored as candidate links. Otherwise, *small-links* can never be candidates as they never cross u. Now, if u lies on the path from u' to u^* then all *far-links* will satisfy both origin and target conditions. Else, *far-links* do not qualify. Hence, any link which is not among top-g (highest scored) of these far-links, can never be the candidate. □

Taking a clue from Lemma 8 and 9, for every prime node u', we shall maintain a data structure as in Lemma 6 by considering only the links stored at u', and top-k queries can be answered faster when $k \leq g$. For this we shall define a candidate tree $CT(u')$ of node u' (except the root) to be a modified version of subtree of u' in GST augmented with candidate links stored at u'. Firstly, for every candidate link which is targeted above u', we change the target to v, which will be a dummy parent of u' in $CT(u')$. Now $CT(u')$ consists of those nodes which are either origin or target (after modification) of some candidate link of u'. Moreover, all the nodes in $subtree(u')\backslash subtree(u^*)$ are included as well. Since only the subset of nodes is selected from $subtree(u')$, our tree is basically a Steiner tree connecting these nodes. Moreover, the tree is edge-compacted so that no degree-1 node remains. Thus, the size of the tree as well as the number of associated links is $O(g)$. Next we do a rank-space reduction of pre-order rank (w.r.t to GST) of the nodes in $CT(u')$ as well as the scores of candidate links.

The candidate tree (no degree-1 nodes) as well as the associated candidate links satisfies all the properties which we have exploited while deriving the structure in Lemma 6. Hence such a structure for $CT(u')$ can be maintained in $O(min(g, size(u'))$ words space and the top-k links in $CT(u')$ stabbed by any node u, with u' being its lowest prime ancestor can be retrieved in $O(\log^2(g/B) + k/B)$ I/O's. The total space consumption of structures corresponding all prime nodes can be bounded by $O(n)$ words as follows: the number of prime nodes with at least a marked node in its subtree is $O(n/g)$, as each such prime node can be associated with a unique marked node. Thus the associated structures takes $O(n/g \times g) = O(n)$ words space. The candidate set of a prime node u' with no marked nodes in its subtree consists of $O(size(u'))$ links, moreover a link cannot be in the candidate set of two such prime nodes. Thus the total space is $O(n)$ words in this case as well. Note that for $g = O(B)$, we need not store any structure on $CT(u')$, because such a candidate tree fits entirely in constant number of blocks which can be processed in $O(1)$ I/Os. This completes the proof of Lemma 7.

3.4 I/O-Optimal Data Structure via Bootstrapping

The bounds in Theorem 1 can be achieved by maintaining multiple structures as in Lemma 7. Clearly the structure in Lemma 6 is optimal for $k \geq B\log^2(n/B)$. However, for handling the case when $k < B\log^2(n/B)$, we shall choose the grouping factor $g_i = B(\log^{(i)}(n/B))^2$, for $i = 1, 2, 3, .., h \leq \log^* n$ and maintain h separate structures as in Lemma 7, occupying $O(nh)$ space. Thus top-k query for any $k \geq g_h$ can be answered by querying on the structure corresponding to the grouping factor g_j, where $g_j \geq k > g_{j+1}$ in $O(\log^2(g_j/B) + k/B) = O(g_{j+1}/B + k/B) = O(k/B)$ I/Os. For $k < g_h$, we shall query on the structure corresponding to the grouping factor g_h, and the I/Os are bounded by $O(\log^2(g_h/B) + k/B) = O(\log^{(h)} n + k/B)$. This completes the proof of Theorem 1.

4 Adapting to Internal Memory

Our external memory framework can be adapted to internal memory by choosing $B = \Theta(1)$, and by replacing the external memory substructures by the corresponding internal memory counterparts. Retrieving the outputs among high-ranked links is reduced to $O(\log n)$ 3-sided range reporting queries. By using an interval tree like approach, the problem of retrieving outputs among equi-ranked links also can be reduced to $O(\log n)$ 3-sided range reporting queries. By using the linear-space sorted range reporting structure by Brodal et al. [7] for 3-sided range reporting, the outputs can be obtained in the sorted order of score. Further, these sorted outputs from $O(\log n)$ different places can be merged using an atomic heap [14], which is capable of performing all heap operations in $O(1)$ time, provided the number of elements in the heap is $O(\log^{O(1)} n)$ as in our case. At the beginning of each of these $O(\log n)$ queries, we may need to perform a binary search for finding the boundaries, thus resulting in a total query time of $O(\log^2 n + k)$, which is $O(k)$ for $k \geq \log^2 n$. The space can be bounded by $O(n)$ words. For the case when $k < \log^2 n$, we obtain a linear space and $O(\log^2 \log n + k)$ query time structure by using the ideas from Sec 3.3 (here

we choose grouping factor $g = \log^2 n$). Again, this structure can answer queries in $O(k)$ time for $k \geq \log^2 \log n$. We do not continue this bootstrapping further. Instead, we make use of the following observation: the candidate set of a node consists of only $O(g)$ links, hence a pointer to any particular link within the candidate set of any node can be maintained in $O(\log g) = O(\log \log n)$ bits. Thus, at every node u in GST, we shall maintain the top-$(\log^2 \log n)$ links stabbed by u in the decreasing order of score as a pointer to its location within the candidate set of u. This occupies $O(n \log^3 \log n)$ bits or $o(n)$ words and top-k queries for any $k \leq \log^2 \log n$ can be answered in $O(k)$ time by chasing the first k pointers and retrieving the documents associated with the corresponding links. This completes the proof of Theorem 2.

References

1. P. Afshani. On dominance reporting in 3d. In *ESA*, pages 41–51, 2008.
2. P. Afshani, G. S. Brodal, and N. Zeh. Ordered and unordered top-k range reporting in large data sets. In *SODA*, pages 390–400, 2011.
3. A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, 1988.
4. L. Arge, V. Samoladas, and J. S. Vitter. On two-dimensional indexability and optimal range search indexing. In *PODS*, pages 346–357, 1999.
5. D. Belazzougui, G. Navarro, and D. Valenzuela. Improved compressed indexes for full-text document retrieval. volume 18, pages 3–13, 2013.
6. M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973.
7. G. S. Brodal, M. G. R. Fagerberg, and A. López-Ortiz. Online sorted range reporting. In *ISAAC*, pages 173–182, 2009.
8. T. M. Chan, S. Durocher, K. G. Larsen, J. Morrison, and B. T. Wilkinson. Linear-space data structures for range mode query in arrays. In *STACS*, pages 290–301, 2012.
9. B. Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *J. ACM*, 37(2):200–212, 1990.
10. J. S. Culpepper, G. Navarro, S. J. Puglisi, and A. Turpin. Top-k ranked document search in general text databases. In *ESA (2)*, pages 194–205, 2010.
11. J. S. Culpepper, M. Petri, and F. Scholer. Efficient in-memory top-k document retrieval. In *SIGIR*, 2012.
12. P. Ferragina and R. Grossi. The string b-tree: A new data structure for string search in external memory and its applications. *J. ACM*, 46(2):236–280, 1999.
13. J. Fischer, T. Gagie, T. Kopelowitz, M. Lewenstein, V. Mäkinen, L. Salmela, and N. Välimäki. Forbidden patterns. In *LATIN*, pages 327–337, 2012.
14. M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48(3):533–551, 1994.
15. T. Gagie, G. Navarro, and S. J. Puglisi. New algorithms on wavelet trees and applications to information retrieval. *Theor. Comput. Sci.*, 426:25–41, 2012.
16. M. Greve, A. G. Jørgensen, K. D. Larsen, and J. Truelsen. Cell probe lower bounds and approximations for range mode. In *ICALP (1)*, pages 605–616, 2010.
17. W.-K. Hon, M. Patil, R. Shah, S. V. Thankachan, and J. S. Vitter. Indexes for document retrieval with relevance. In *Munro Festschrift*, pages 351–362, 2013.
18. W.-K. Hon, R. Shah, and S. V. Thankachan. Towards an optimal space-and-query-time index for top-k document retrieval. In *CPM*, pages 173–184, 2012.

19. W.-K. Hon, R. Shah, S. V. Thankachan, and J. S. Vitter. String retrieval for multi-pattern queries. In *SPIRE*, pages 55–66, 2010.
20. W.-K. Hon, R. Shah, S. V. Thankachan, and J. S. Vitter. Document listing for queries with excluded pattern. In *CPM*, pages 185–195, 2012.
21. W.-K. Hon, R. Shah, S. V. Thankachan, and J. S. Vitter. Faster compressed top-k document retrieval. In *DCC*, 2013.
22. W.-K. Hon, R. Shah, and J. S. Vitter. Space-efficient framework for top-k string retrieval problems. FOCS '09, pages 713–722, 2009.
23. W.-K. Hon, R. Shah, and J. S. Vitter. Compression, indexing, and retrieval for massive string data. In *CPM*, pages 260–274, 2010.
24. M. Karpinski and Y. Nekrich. Top-k color queries for document retrieval. In *SODA*, pages 401–411, 2011.
25. R. Konow and G. Navarro. Faster compact top-k document retrieval. In *DCC*, 2013.
26. G. Kucherov, Y. Nekrich, and T. A. Starikovskaya. Cross-document pattern matching. In *CPM*, pages 196–207, 2012.
27. M. O. Külekci, J. S. Vitter, and B. Xu. Efficient maximal repeat finding using the burrows-wheeler transform and wavelet tree. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 9(2):421–429, 2012.
28. K. G. Larsen and R. Pagh. I/o-efficient data structures for colored range and prefix reporting. In *SODA*, pages 583–592, 2012.
29. K. G. Larsen and F. van Walderveen. Near-optimal range reporting structures for categorical data. In *SODA*, pages 265–276, 2013.
30. Y. Matias, S. Muthukrishnan, S. C. Sahinalp, and J. Ziv. Augmenting suffix trees, with applications. ESA '98, pages 67–78, London, UK, UK, 1998. Springer-Verlag.
31. S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *SODA*, pages 657–666, 2002.
32. G. Navarro. Spaces, trees and colors: The algorithmic landscape of document retrieval on sequences. *CoRR*, abs/1304.6023, 2013.
33. G. Navarro and Y. Nekrich. Top-k document retrieval in optimal time and linear space. In *SODA*, pages 1066–1077, 2012.
34. G. Navarro and S. J. Puglisi. Dual-sorted inverted lists. In *SPIRE*, pages 309–321, 2010.
35. G. Navarro and D. Valenzuela. Space-efficient top-k document retrieval. In *SEA*, pages 307–319, 2012.
36. Y. Nekrich. Space-efficient range reporting for categorical data. In *PODS*, pages 113–120, 2012.
37. M. Patil, S. V. Thankachan, R. Shah, W.-K. Hon, J. S. Vitter, and S. Chandrasekaran. Inverted indexes for phrases and strings. SIGIR, pages 555–564, 2011.
38. K. Sadakane. Succinct data structures for flexible text retrieval systems. *J. Discrete Algorithms*, 5(1):12–22, 2007.
39. C. Sheng and Y. Tao. Dynamic top-k range reporting in external memory. In *PODS*, pages 121–130, 2012.
40. Y. Tao. Lecture 1: External memory model and sorting.
41. N. Välimäki, S. Ladra, and V. Mäkinen. Approximate all-pairs suffix/prefix overlaps. In *CPM*, pages 76–87, 2010.
42. N. Välimäki and V. Mäkinen. Space-efficient algorithms for document retrieval. In *CPM*, pages 205–215, 2007.
43. J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2), July 2006.