Ranked Document Retrieval in External Memory

RAHUL SHAH, Department of Computer Science, Louisiana State University
CHENG SHENG, Department of Computer Science and Engineering, Chinese University of Hong Kong
SHARMA THANKACHAN, Department of Computer Science, North Carolina State University
JEFFREY VITTER, Department of Computer and Information Science, University of Mississippi

The ranked (or top-k) document retrieval problem is defined as follows: preprocess a collection $\{T_1, T_2, \ldots, T_d\}$ of d strings (called documents) of total length n into a data structure, such that for any given query (P, k), where P is a string (called pattern) of length $p \geq 1$ and $k \in [1, d]$ is an integer, the identifiers of those k documents that are most relevant to P can be reported, ideally in the sorted order of their relevance. The seminal work by Hon et al. [FOCS 2009 and Journal of the ACM 2014] presented an $O(n)$-space (in words) data structure with $O(p + k \log k)$ query time. The query time was later improved to $O(p + k)$ [SODA 2012] and further to $O(p/\log_\sigma n + k)$ [SIAM Journal on Computing 2017] by Navarro and Nekrich, where σ is the alphabet size. We revisit this problem in the external memory model and present three data structures. The first one takes $O(n)$-space and answer queries in $O(p/B + \log_B n + k/B + \log^*(n/B))$ I/Os, where B is the block size. The second one takes $O(n \log^*(n/B))$ space and answer queries in optimal $O(p/B + \log_B n + k/B)$ I/Os. In both cases, the answers are reported in the unsorted order of relevance. To handle sorted top-k document retrieval, we present an $O(n \log(d/B))$ space data structure with optimal query cost.

CCS Concepts: • **Theory of computation** → *Data structures design and analysis;*

Additional Key Words and Phrases: Data structures, text indexing, external memory

ACM Reference format:
Rahul Shah, Cheng Sheng, Sharma Thankachan, and Jeffrey Vitter. 2023. Ranked Document Retrieval in External Memory. *ACM Trans. Algorithms* 19, 1, Article 5 (March 2023), 12 pages.
https://doi.org/10.1145/3559763

1 INTRODUCTION AND RELATED WORK

The inverted index is the most fundamental data structure in the field of information retrieval [25]. It is the backbone of every known search engine today [15]. For each word in any document collection, the inverted index maintains a list of all documents in that collection that contain the word. Despite its power to answer various types of queries, the inverted index becomes inefficient, for

This research is supported by US NSF Grants CCF–1017623 and CCF–1218904.
Authors' addresses: R. Shah, Department of Computer Science, Louisiana State University, 3325 Patrick F. Taylor Hall, Baton Rouge, Louisiana 70803, USA; email: rahul@csc.lsu.edu; C. Sheng, Department of Computer Science and Engineering, Chinese University of Hong Kong; email: jeru.sheng@gmail.com; S. Thankachan, Department of Computer Science, North Carolina State University, Engineering Building II–Campus Box 8206, 890 Oval Dr., Raleigh, North Carolina 27606, USA; email: sharma.thankachan@gmail.com; J. Vitter, Department of Computer and Information Science, University of Mississippi, 201 Weir Hall University, Oxford, Mississippi 38677, USA; email: jsv@olemiss.edu.

1549-6325/2023/03-ART5 $15.00
https://doi.org/10.1145/3559763

example, when queries are phrases instead of words [20]. Similar problems also occur in applications when word boundaries do not exist or cannot be identified uniquely 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 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. This approach can take quadratic space and hence is neither theoretically interesting nor sensible from a practical viewpoint.

The first framework for answering document retrieval queries was proposed by Matias et al. [12]. Their data structures solve the *document listing problem*, where the task is to index a document collection $\mathcal{D} = \{T_1, T_2, \ldots, T_d\}$, such that whenever a string P (called pattern) of length p comes as a query, the index report the identifiers of all those documents containing P (i.e., as a substring). Later Muthukrishnan [14] initiated the study of relevance metric-based document retrieval, which was then formalized by Hon et al. [10] as follows:

Problem 1 (Top-k Document Retrieval Problem). Let $\mathcal{D} = \{T_1, T_2, \ldots, T_d\}$ be a collection of d strings (called documents) of total length n over a totally ordered alphabet Σ of size σ. Also for any pattern P, let $w(P, T_i)$ be the relevance of T_i w.r.t. P. The task is to build an index over $\mathcal{D}$ answering the following query: Given a string P (called pattern) of length $p \geq 1$ and an integer $k \in [1, d]$, report the identifiers of those k documents that are most relevant to P (ties are broken arbitrarily), ideally in the sorted order of their $w(P, .)$ values.

The relevance metrics considered in the problem can be either pattern independent (e.g., PageRank) or pattern dependent. In the latter case, $w(P, T_i)$ can take into account information like the frequency of the pattern occurrences (or *term-frequency*, the number of occurrences of P in a document T_i) and even the locations of the occurrences (e.g., *min-dist* [10], which takes proximity of two closest occurrences of pattern as the score). As in the previous works, we assume that other than a static weight that is fixed for each document i, $w(P, T_i)$ is dependent only on the set of occurrences (i.e., starting positions) of P in T_i. This means, metrics like $w(P, T_i)$ as the size of the maximal set of non-overlapping occurrences of P in T_i do not qualify as they not only depends on the set of occurrences but also the length of the pattern.

The framework of Hon et al. [9, 10] takes linear space and answers the query in $O(p + k \log k)$ time, assuming integer alphabet, i.e., $\Sigma = \{0, 1, 2, \ldots, n^{O(1)}\}$. They reduced this problem to a four-sided orthogonal range query in three-dimensional (3D) grid, which is defined as follows: The data consist of a set S of 3D points (with integer coordinate values), and the query consists of four (integer) 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 any data structure for (general) four-sided orthogonal range searching in optimal time needs super-linear space [5], the desired bounds can nevertheless be achieved by identifying a particular property that one dimension of the reduced subproblem can only have p distinct values. The query time was then brought down to $O(p + k)$ [16] and further to $O(p/\log_\sigma n + k)$ [17] by Navarro and Nekrich. Recently, Munro et al. [13] proposed a linear space structure, which can answer a more powerful query, called ranked document selection (report the kth most relevant document) in $O(p + \log k)$ time. Even though there has been a series of work on this topic, including in theory as well as in practical IR communities, most implementations (as well as theoretical results) have focused on RAM-based compressed indexes (see References [8, 15] for surveys on this topic).

We introduce an alternative framework for solving the top-k document retrieval problem and obtain the first non-trivial (comparison-based) external memory solutions. This model (a.k.a. cache-aware model, I/O model, and disk access model) was introduced by Aggarwal and Vitter [2]. Here

the CPU is connected directly to an internal memory (whose size is M words), which is then connected to a very large external memory (disk). The disk is partitioned into blocks (pages), and the size of each block is B words. The CPU can only work on data inside the internal memory. Therefore, to work on some data in the external memory, the corresponding blocks have to be transferred to internal memory. The transfer of a block from external memory to internal memory (or vice versa) is referred to as an I/O operation. The operations inside the internal memory are orders of magnitude faster than the time for an I/O operation. Therefore, they are considered free, and an algorithm's efficiency is measured by the number of I/O operations.

We now present our main results.[1] Unlike the previous results in the RAM model, we do not make the integer alphabet assumption. Instead, we assume that the characters are encoded in $O(1)$ machine words, and any two characters stored in the internal memory can be compared for free. The optimal query cost is $O(p/B + \log_B n + k/B)$ I/Os, since p is the input size, k is the output size, and $\Omega(\log_B n)$ I/Os are necessarily for any (comparison-based) searching.

Theorem 1. *There exists an $O(n)$-word structure for answering top-k (unsorted) document retrieval queries in $O(p/B + \log_B n + k/B + \log^*(n/B))$ I/Os, where B is the block size.*

Theorem 2. *There exists an $O(n \log^*(n/B))$-word structure for answering top-k (unsorted) document retrieval queries in optimal $O(p/B + \log_B n + k/B)$ I/Os, where B is the block size.*

Theorem 3. *There exists an $O(n \log(d/B))$-word structure for answering top-k (sorted) document retrieval queries in optimal $O(p/B + \log_B n + k/B)$ I/Os, where B is the block size.*

We remark that some of our techniques are closely related to the work by Larsen and Walderveen [11] on colored range queries in 2D. Here, the task is to construct a data structure over a set S of n colored points in 2D so that given an orthogonal range query $[a, b] \times [c, \infty)$, we can report the set of distinct colors in $S \cap [a, b] \times [c, \infty)$ efficiently. They presented an $O(n \log^*(n/B))$ space data structure with optimal query cost of $O(\log_B n + k/B)$ I/Os, where k is the output size. For more results on this topic, see References [7, 19, 21].

2 PRELIMINARY: TOP-k FRAMEWORK

This section briefly explains the framework for top-k document retrieval based on the work of Hon et al. [10]. The **generalized suffix tree (GST)** of a document collection $\mathcal{D} = \{T_1, T_2, \ldots, T_d\}$ is the combined compact trie (a.k.a. Patricia trie) of all the non-empty suffixes of all the documents after appending each suffix with a special character that is not in Σ, which is unique to the corresponding document (say, $\$_i$ for T_i). Use n to denote the total length of all the documents, which is also the number of leaves in GST. For each node u in the GST (referred by its pre-order rank), consider the path from the root node to u. Let $depth(u)$ be the number of nodes on the path, $prefix(u)$ be the string obtained by concatenating all the edge labels of the path, and $size(u)$ be the number of leaves under u. For a pattern P that appears in at least one document, the *locus* of P, denoted by u_P, is the node closest to the root such that P is a prefix of $prefix(u_P)$. Moving forward, we assume that P is non-empty (i.e., $p \geq 1$); therefore, $u_P \neq root$.

Nodes are marked with documents as follows. A leaf node ℓ is marked with a (unique) document $T_i \in \mathcal{D}$ if the suffix represented by ℓ belongs to T_i. An internal node u is marked with T_i if it is the **lowest common ancestor (LCA)** of two leaves marked with T_i. Notice that an internal node can be marked with multiple documents. Additionally, we mark the root node with all documents. For each node $u \neq root$ and each of its marked documents T_i, define a *link* to be a quadruple $(origin, target, doc, score)$, where $origin = u$, $target$ is the lowest proper ancestor of u marked with T_i, $doc = i$ and $score = w(prefix(u), T_i)$. Let $\mathcal{L}$ denote the set of all such links.

[1] Theorem 2 was presented in the conference version of this article [22], but Theorems 1 and 3 are new.

Lemma 1 ([10]). *For each document T_i that contains a pattern P as a substring, there exists a unique link in $\mathcal{L}$ 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 T_i with respect to P.*

Thus, the top-k document retrieval problem can be reduced to the problem of finding the top-k links (according to their score) among all links in $\mathcal{L}$ stabbed by u_P. We say that a link is *stabbed* by a node u if its origin is in the subtree of u, and its target is a proper ancestor of u.

Lemma 2 ([10]). *The number of links originating from the subtree of any node u is at most $2 \cdot size(u) - 1$, where $size(u)$ denotes the number of leaves under u. Therefore, $|\mathcal{L}| \leq 2n - 1$.*

Moving forward, we shall assume that all score values are distinct integers within $[1, 2n - 1]$. Otherwise, we achieve this by first sorting all links in the ascending order of their score values (ties broken arbitrarily) and then replacing each link's score by its position in the sorted list.

3 EXTERNAL MEMORY STRUCTURES

This section is dedicated to proving our results. The initial phase of searching the locus node u_P of P can be performed in optimal $O(p/B + \log_B n)$ I/O's using a string B-tree [6] data structure over $\mathcal{D}$ and its space is $O(n)$. Assuming u_P exists and $u_P \neq root$, we now focus on reporting the top-k links stabbed by u_P. Instead of solving this top-k version, we first solve a threshold version of the problem, where the objective is to retrieve those links stabbed by u_P with *score* at least a given value τ. Among all the links in $\mathcal{L}$ stabbed by u_P, let $\mathcal{L}(u_P, k, \cdot)$ be the set of k highest scored links and $\mathcal{L}(u_P, \cdot, \tau)$ be the set of links with score $\geq \tau$. In Section 3.3, we propose a separate structure that reduces the original top-k-form query $(u_P, k, \cdot)$ into an equivalent threshold-form query $(u_P, \cdot, \tau)$. Therefore, we can answer top-k-form queries using our structure for threshold-form queries.

3.1 A Data Structure for Handling Threshold-form Queries

The query is of the form $(u_P, \cdot, \tau)$ and the task is to output $\mathcal{L}(u_P, \cdot, \tau)$. This problem can be decomposed into simpler queries, which consist of a 3D dominance reporting and $O(\log(n/B))$ three-sided range reporting in 2D; both problems can be solved efficiently using known structures. The main result is captured in Lemma 3 stated below.

Lemma 3. *By maintaining an $O(n)$-space structure, we can report $\mathcal{L}(u_P, \cdot, \tau)$ for any given threshold-form query $(u_P, \cdot, \tau)$ in $O(\log^2(n/B) + z/B)$ I/Os, where $z = |\mathcal{L}(u_P, \cdot, \tau)|$.*

For any node u in GST, define its *rank* as

$$rank(u) = \lfloor \log \lceil size(u)/B \rceil \rfloor.$$

Note that $rank(.) \in [0, \lfloor \log \lceil n/B \rceil \rfloor]$. A maximal connected sub-graph consisting of nodes with the same rank is called a *component*, and the *rank* of a component is the 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, we can see that a node and at most one of its children can have the same positive *rank*. Therefore, a component with $rank > 0$ forms a path that goes top-down in the tree.

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 $O(n)$ space. Additionally, we maintain a pointer from each node in that component to this list. Now, given a node u_P, if $rank(u_P) = 0$, then 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 challenging when $rank(u_P) > 0$. For handling this, consider the following classification of links based on the *rank* of their targets with respect to the *rank* of u_P:

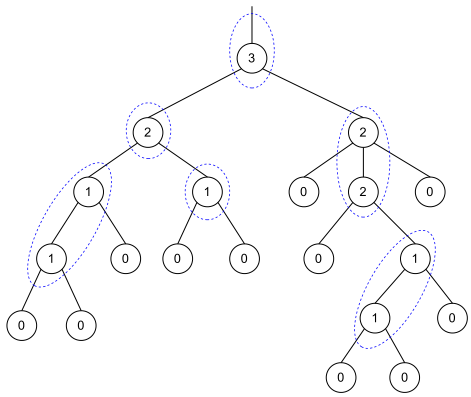

Fig. 1. An illustration of rank components with $B = 1$.

(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 all equi-ranked links in the output can be reduced to a 3D dominance query, and the problem of retrieving all high-ranked links in the output can be reduced to at most $\lfloor \log \lceil n/B \rceil \rfloor$ three-sided range queries in 2D. From now onwards, the origin, target, and score of a link $L_i = (o_i, t_i, \cdot, w_i)$ are represented by o_i, t_i, and w_i, respectively.

3.1.1 Processing Equi-ranked Links. Let C be a component and S_C be the set of all links L_i whose target t_i is a node in C. Since u_P is a node in C, among all equi-ranked links, we need to consider only the 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. For any link $L_i \in S_C$, let pseudo-origin s_i be the pre-order rank of the 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 $s_i \geq u$. 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 the subproblem of reporting equi-ranked links 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 those points. Now the output 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 $O(|S_C|)$ 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} is the output size. Thus overall these structures occupy $O(n)$-space.

Lemma 4. *Using an $O(n)$-space structure, we can report all the equi-ranked links in $\mathcal{L}(u_P, \cdot, \tau)$ for any given threshold-form query $(u_P, \cdot, \tau)$ with $rank(u_P) > 0$ in $O(\log_B n + z_{eq}/B)$ I/Os, where z_{eq} is the output size.*

3.1.2 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 high-ranked links in the output, the condition of t_i being a proper ancestor of u_P can be ignored. The reason is that it will be automatically satisfied if $o_i \in [u_P, \hat{u}_P]$, where $\hat{u}_P$ is (pre-order rank of) right-most leaf in the subtree rooted at u_P. Let G_r be the set of all links with rank equals r for $1 \leq r \leq \lfloor \log \lceil n/B \rceil \rfloor$. 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

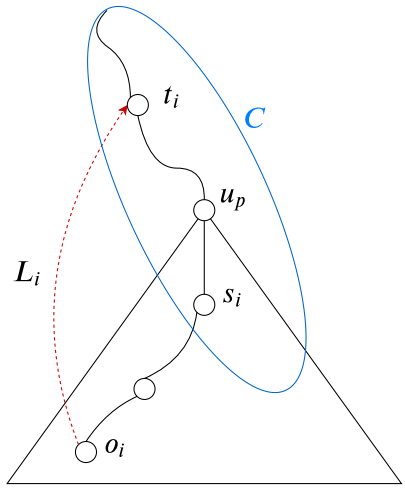

Fig. 2. An illustration of the pseudo-origin s_i of the link L_i with respect to component C.

score values. We treat each link $L_i \in G_r$ as a 2D point (o_i, w_i) and maintain a three-sided range query structure over them for all $r \in [1, \lfloor \log \lceil n/B \rceil \rfloor]$. 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 $r = \lceil rank(u_P) + 1, \lfloor \log \lceil n/B \rceil \rfloor \rfloor$. By using the linear space data structure in Reference [3], 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 are proportional to $\log_B n \cdot \log(n/B) + z_{hi}/B = O(\log^2(n/B) + z_{hi}/B)$, where z_{hi} represents the number of high-ranked output links.

Lemma 5. *Using an $O(n)$-space structure, we can report all the high-ranked links in $\mathcal{L}(u_P, \cdot, \tau)$ for any given threshold-form query $(u_P, \cdot, \tau)$ with $rank(u_P) > 0$ in $O(\log^2(n/B) + z_{hi}/B)$ I/Os, where z_{hi} is the output size.*

By combining Lemmas 4 and 5, and the solution for the case of $rank(u_P) = 0$, we obtain Lemma 3.

3.1.3 *A Straightforward Generalization of Lemma 3.*

Lemma 6. *For any fixed node u in GST, let $\mathcal{L}(u) \subseteq \mathcal{L}$ be the set of links originating from the subtree of u. Also, let S be any subset of $\mathcal{L}(u)$. Then, by maintaining an auxiliary structure of space $O(|S|)$ with the GST, we can report $\mathcal{L}(u_P, \cdot, \tau) \cap S$ for any given threshold-form query $(u_P, \cdot, \tau)$ in $O(\log^2(m/B) + z/B)$ I/Os, provided u_P is in the subtree of u. Here $m = size(u)$ and z is the output size.*

Proof. In the proof of Lemma 3, replace GST with the subtree rooted at u and $\mathcal{L}$ with S. Here we assume that GST (and therefore the subtree rooted at u) is already available. □

3.2 Better Data Structures for Handling Threshold-form Queries

We prove the following results in this section.

Theorem 4. *By maintaining an $O(n)$-space structure, we can report $\mathcal{L}(u_P, \cdot, \tau)$ for any given threshold-form query $(u_P, \cdot, \tau)$ in $O(\log_B n + z/B + \log^*(n/B))$ I/Os, where $z = |\mathcal{L}(u_P, \cdot, \tau)|$.*

Theorem 5. *By maintaining an $O(n \log^*(n/B))$ space structure, we can report $\mathcal{L}(u_P, \cdot, \tau)$ for any given threshold-form query $(u_P, \cdot, \tau)$ in $O(\log_B n + z/B+)$ I/Os, where $z = |\mathcal{L}(u_P, \cdot, \tau)|$.*

We start with a sampling scheme that will be used heavily in the rest of this article.

3.2.1 A Scheme for Sampling Nodes in a Given Tree [10]. Let $\mathcal{T}$ be any given tree (rooted and ordered) with n leaves and $g \in [1, n]$ be a parameter called the sampling factor. Also, assume that all internal nodes in $\mathcal{T}$ have at least two children. The following is a scheme that designates $O(n/g)$ nodes as marked nodes (we call them g-marked nodes): mark every gth leaf node (in the left to right order) and then mark the LCA of all pairs of marked leaves. Additionally, we ensure that the root is always marked. Given any node u with $size(u) \geq g$, at least one node under u is marked. For any unmarked node u its highest marked descendent u^* (if it exists) is unique, and we can locate it via a single LCA query in $O(1)$ time.[2] Moreover, $size(u) - size(u^*) < 2g$.

We categorize the nodes in GST and associated links into different types in $[1, \log^*(n/B)]$ based on this marking scheme as follows. Let $h \in [1, \log^*(n/B)]$, $g_h = B(\log^{(h)}(n/B))^3$ and $K_h = B(\log^{(h)}(n/B))^2$, where $\log^{(1)}(\cdot) = \log(\cdot)$ and $\log^{(h)}(\cdot) = \log\log^{(h-1)}(\cdot)$ for $h > 1$ and $\log^* x = \min\{j \mid \log^{(j)} x \leq 1\}$. We then replace each g_h and K_h by $2^{\lfloor \log g_h \rfloor}$ and $2^{\lfloor \log K_h \rfloor}$, respectively (i.e., rounding down to the nearest power of two). This guarantees that every g_h-marked node is also g_{h+1}-marked. Also define $g_0 = n + 1$.

- A node is of type 1 if at least one node in its subtree is g_1-marked. Any other node is of type $h \in [2, \log^*(n/B)]$ if in its subtree, at least one node is g_h-marked and no node is g_{h-1}-marked. Therefore, $size(\cdot)$ of any type-h node is $O(g_{h-1})$. The nodes remaining are also of type $\log^*(n/B)$ and their $size(\cdot)$ is $O(B)$. Note that the type of the parent of a type-h node is $\leq h$.
- A subtree rooted at a node u is a type-h subtree if u's type is h and u's parent's type is $< h$. When u is the root note, the tree rooted at u is the entire GST, which we call a type-1 subtree.
- A link is of type-h if its target is a type-h node,[3] and let $\mathcal{L}_h$ be the set of all type-h links.

Along with each node and link, we store its type explicitly.

Our approach to handle a query $(u_P, \cdot, \tau)$ is the following. Let r be the type of u_P. Recall our assumption that $u_P \neq root$ (otherwise, P is an empty string). When $r = \log^*(n/B)$, we extract $\mathcal{L}(u_P, \cdot, \tau) \subseteq \mathcal{L}(u_P)$ via a linear scan of $\mathcal{L}(u_P)$. It takes only $|\mathcal{L}(u_P)|/B \leq 2 \cdot size(u_P)/B = O(1)$ I/Os. Now if $r \in [1, \log^*(n/B))$, then we report all type-h links in the output, i.e., $\mathcal{L}_h \cap \mathcal{L}(u_P, \cdot, \tau)$ for all values of $h < r$ first, and then for $h = r$. All type-h links with $h > r$ can be ignored, because such links cannot be stabbed by any type-r node. This follows from the monotonicity of types (the parent of a type-h node is of type $h' \leq h$). We now present the details.

3.2.2 Structures for Reporting Type-h Links with $h < r$.

Lemma 7. *We can maintain an $O(n)$-space structure, such that given a query $(u_P, \cdot, \tau)$ with node u_P being of type-r, we can report $(\cup_{h=1}^{r-1} \mathcal{L}_h) \cap \mathcal{L}(u_P, \cdot, \tau)$ in $O(\log_B n + \log^*(n/B) + \sum_{h=1}^{r-1} z_h/B)$ I/Os, where $z_h = |\mathcal{L}_h \cap \mathcal{L}(u_P, \cdot, \tau)|$.*

Proof. Note that for all $h < r$, $\mathcal{L}_h \cap \mathcal{L}(u_P, \cdot, \tau)$ is the set of type-h links originating from the type-h subtree containing u_P (i.e., with origin within $[u_P, \hat{u}_P]$) and score $\geq \tau$, where $\hat{u}_P$ is the rightmost leaf under u_P. To report them efficiently, we maintain for all $h \in [1, \log^*(n/B)]$ and for each type-h subtree with H being the set of type-h links originating from it, a (linear-space) structure over the points in $\{(o_i, w_i) \mid (o_i, t_i, \cdot, w_i) \in H\}$ for answering 2D three-sided range reporting queries in optimal I/Os [1]. The total space over all such structures for a fixed h is proportional to the total number of type-h links. Therefore, total space over all values of h is $O(n)$. To report (the points corresponding to the links in) $\mathcal{L}_h \cap \mathcal{L}(u_P, \cdot, \tau)$, where $h < r$, we issue a 2D three-sided range

[2]Find the first and last marked leaves between the leftmost and the rightmost leaves, and then take their LCA. The LCA queries can be answered in $O(1)$ time by maintaining a structure of space $O(n)$ bits [18].
[3]Note the similarity with the definition of link's rank in Section 3.1.

reporting query $[u_P, \hat{u}_P] \times [\tau, \infty)$ on the structure associated with the type-h tree containing u_P. The I/Os required is $O(\log_B g_{h-1} + z_h/B)$ for a fixed h. Therefore, the I/Os required for all values of $h < r$ is proportional to $\sum_{h=1}^{r-1} \log_B g_{h-1} + \sum_{h=1}^{r-1} \lceil z_h/B \rceil = O(\log_B n + r + (\sum_{h=1}^{r-1} z_h)/B)$ and $r \leq \log^*(n/B)$. □

Lemma 8. *We can maintain an $O(n \log^*(n/B))$ space structure, such that given a threshold-form query $(u_P, \cdot, \tau)$ with node u_P being of type-r, we can report $(\cup_{h=1}^{r-1} \mathcal{L}_h) \cap \mathcal{L}(u_P, \cdot, \tau)$ in $O(\log_B n + \sum_{h=1}^{r-1} z_h/B)$ I/Os, where $z_h = |\mathcal{L}_h \cap \mathcal{L}(u_P, \cdot, \tau)|$.*

Proof. Modify the proof of Lemma 7 as follows: replace H with H', where H' is the set of all type-j links originating from that type-h subtree for all values of $j < h$. We can now report $(\cup_{h=1}^{r-1} \mathcal{L}_h) \cap \mathcal{L}(u_P, \cdot, \tau)$ via a single 2D three-sided range reporting query $[u_P, \hat{u}_P] \times [\tau, \infty)$ on the (modified) structure associated with the type-r subtree containing u_P in I/Os proportional to $\log_B(g_{r-1}/B) + \sum_{h=1}^{r-1} z_h/B = O(\log_B n + \sum_{h=1}^{r-1} z_h/B)$. This modification increases the asymptotic space complexity to $O(n \log^*(n/B))$, because each type-h link can now belong to $h = O(\log^*(n/B))$ structures. □

3.2.3 Structures for Reporting Type-r Links.

Lemma 9. *We can maintain an $O(n)$-space structure, such that given a threshold-form query $(u_P, \cdot, \tau)$ with node u_P being of type-r, we can report $\mathcal{L}_r \cap \mathcal{L}(u_P, \cdot, \tau)$ in $O((\log^{(r)}(n/B))^2 + z_r/B)$ I/Os, where $z_r = |\mathcal{L}_r \cap \mathcal{L}(u_P, \cdot, \tau)|$.*

Proof. For all $h \in [0, \log^*(n/B)]$ and for each type-h subtree with H being the set of type-h links originating from it, maintain the (linear-space) structure in Lemma 6 over H. The total space over all such structures for a fixed h is proportional to the total number of type-h links. Therefore, total space over all values of h is $O(n)$. To find $\mathcal{L}_r \cap \mathcal{L}(u_P, \cdot, \tau)$ we query on the structure associated with the type-r subtree containing u_P. The I/Os required is $\log^2(g_{r-1}/B) + z_r/B = O((\log^{(r)}(n/B))^2 + z_r/B)$. □

Note that the query cost in Lemma 9 is optimal when $z_r \geq K_r$. Therefore, we use that structure only when $z_r \geq K_r$. For the case where $z_r < K_r$, we introduce another structure in Lemma 11. Since we do not know z_r in advance, we use the following strategy for deciding which structure to use: For each node u in GST with h being its type, we store τ_u, the score of K_hth type-h link stabbed by u. Then, $z_r \geq K_r$ iff $\tau \leq \tau_{u_P}$. This takes only $O(n)$ extra space and the choice can be made in $O(1)$ I/Os.

Before we present Lemma 11, we introduce some additional definitions. Let g be a sampling factor and u^* be a g-marked node. Also, let u' be the last node on the path from u^* to root, before another g-marked node. Therefore, u' is the highest node such that the highest marked node in its subtree is u^*. Consider all links originating from the subtree of u'. We classify them into four groups (see Figure 3 for an illustration) and make some useful observations.

- $farLinks(u^*, g)$ is the set of links stabbed by both u^* and u'.
- $smallLinks(u^*, g)$ is the set of links originating from u^*'s subtree, but not stabbed by u^*.
- $nearLinks(u^*, g)$ is the set of links stabbed by u^*, but not by u'.
- $fringeLinks(u^*, g)$ is the set of links originating not from the subtree of u^*.

Lemma 10. *For any g-marked node u^*, the size of $nearLinks(u^*, g) \cup fringeLinks(u^*, g)$ is $O(g)$.*

Proof. Let F denote the set of leaves in the subtree of u', but not in the subtree of u^*. Then, $|nearLinks(u^*, g)| \leq |F|$, because for every document T_j, there is at most one link $(\cdot, \cdot, j, \cdot) \in nearLinks(u^*, g)$, and it exists iff a leaf in F and a leaf under u^* are marked with T_j. To bound the size

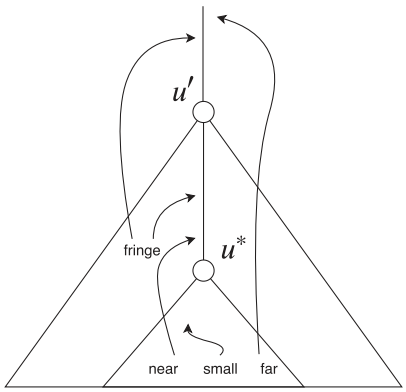

Fig. 3. Categorization of Links originating from the subtree of u'.

of $fringeLinks(u^*, g)$, let us partition it into two sets, say, A and B. All links in $fringeLinks(u^*, g)$ with their origin on the path from u^* to u' are in A and $B = fringeLinks(u^*, g) \backslash A$. Then, $|B| \leq 2|F|$, $|A| \leq |B|$ and $|fringeLinks(u^*, g)| = |A| + |B| \leq 4|F|$. Finally, $|F| = size(u') - size(u^*) \leq 2g$. □

Let u be any node on the path from u^* to u'. Then, u stabs (i) all links in $farLinks(u^*, g)$, (ii) none of the links in $smallLinks(u^*, g)$, (iii) a link $(o_i, t_i, \cdot, w_i) \in nearLinks(u^*, g)$ iff $t_i < u$, and (iv) a link $(o_i, t_i, \cdot, w_i) \in fringeLinks(u^*, g)$ iff t_i is a proper ancestor of u and $LCA(o_i, u^*) \geq u$. With these observations, we now present Lemma 11.

Lemma 11. *We can maintain an $O(n)$-space structure, such that given a threshold-form query $(u_P, \cdot, \tau)$ with node u_P being of type-r, we can report $\mathcal{L}_r \cap \mathcal{L}(u_P, \cdot, \tau)$ in $O(\log_B n + z_r/B)$ I/Os, if $z_r < K_r$. Here $z_r = |\mathcal{L}_r \cap \mathcal{L}(u_P, \cdot, \tau)|$.*

Proof. For each $h \in [1, \log^*(n/B)]$, we maintain a structure, which is constructed as follows. Identify all g_h-marked nodes. Then, for each g_h-marked node u^*, with u' being the last node on the path from u^* to root, before another g-marked node, obtain the set $Q_h(u^*)$ by collecting all type-h links in $nearLinks(u^*, g) \cup fringeLinks(u^*, g)$ and the top-K_h links from the S, where S is the set of all type-h links in $farLinks(u^*, g)$. Note that a node u on the path from u^* to u' stabs a link $(o_i, t_i, \cdot, w_i) \in Q(u^*)$ iff $t_i < u_P \leq LCA(o_i, u^*)$. Therefore, the top-$k$ type-h links stabbed by u are guaranteed to form a subset of $Q(u^*)$ if $k < K_h$. We map each link $(o_i, t_i, \cdot, w_i) \in Q_h(u^*)$ into a 3D point $(LCA(o_i, u^*), t_i, w_i)$. These points are preprocessed into an $O(|Q_h(u^*)|)$ space data structure that can answer 3D dominance reporting queries in optimal I/Os. This completes the description of our data structure. To bound its space, note that the same link cannot be in $nearLinks(\cdot, g) \cup fringeLinks(\cdot, g)$ for two different g-marked nodes. Therefore, the total space taken by all structures for a fixed h is $O(n_h + nK_h/g_h)$, where n_h is the number of type-h links. The final space is proportional to

$$\sum_{h=1}^{\log^*(n/B)} n_h + n \cdot \frac{K_h}{g_h} = \sum_{h=1}^{\log^*(n/B)} n_h + \frac{n}{\Theta(\log^{(h)}(n/B))} = O(n).$$

To answer a query (u_P, τ), we first identify the highest g_r-marked node (say u_P^*) in the subtree of u_P. Then issue a 3D dominance reporting query $[u_P, \infty) \times (-\infty, u_P) \times [\tau, \infty)$ on the set of points corresponding to $Q_r(u_P^*)$. Our answer is the set of links corresponding to the reported points and the I/Os required is proportional to $\log_B |Q_h(u_P^*)| + z_r/B$, i.e., $O(\log_B n + z_r/B)$. □

By combing the results in Lemmas 9 and 11, we obtain the following result.

Lemma 12. *We can maintain an $O(n)$-space structure, such that given a threshold-form query $(u_P, \cdot, \tau)$ with node u_P being of type-r, we can report $\mathcal{L}_r \cap \mathcal{L}(u_P, \cdot, \tau)$ in $O(\log_B n + z_r/B)$ I/Os, where $z_r = |\mathcal{L}_r \cap \mathcal{L}(u_P, \cdot, \tau)|$.*

Finally, Theorem 4 (respectively, Theorem 5) follows from Lemma 7, (respectively, Lemma 8) and Lemma 12.

3.3 Completing the Proofs of Theorems 1–3

We first present and prove Lemma 13, which is a reduction the top-k version to the threshold version of our problem.

Lemma 13. *By maintaining an $O(n)$-space structure, we can compute a threshold value τ for any given $(u_P, k, \cdot)$ in $O(1)$ time, such that $\mathcal{L}(u_P, k, \cdot) \subseteq \mathcal{L}(u_P, \cdot, \tau)$ and $|\mathcal{L}(u_P, \cdot, \tau)| = O(k + \log n)$.*

Proof. The structure is constructed as follows: Identify all the g-marked nodes in the GST for $g = \lceil \log n \rceil$. At every g-marked node u^*, store the score of qth highest scored link stabbed by u^* for $q = 1, 2, 4, 8, \ldots$. The total space is $(n/g) \log n = O(n)$ words. To answer a query (u_P, k), find the highest marked node u^* (if it exists) in the subtree of u_P in $O(1)$ time. Now compute $i = \lceil \log(k+2g) \rceil$ and report τ as the score of 2^ith highest scored link stabbed by u^*. The correctness follows from the following facts: (i) $size(u^*) - size(u_P) < 2g$ and (ii) $w(prefix(u_P), T_j) \neq w(prefix(u^*), T_j)$ only if there exists a leaf marked with T_j and located under u_P, but not under u^*. In the remaining case when there is no marked node under u_P, we report 1 (the lowest possible score after rank-space reduction). This works because the size of $\mathcal{L}(u_P, \cdot, 1)$ is trivially bounded by $size(u_P) < g = O(\log n)$. □

3.3.1 Proof of Theorem 1. We maintain the structures in Theorem 4, Lemma 13, and a string B-tree over $\mathcal{D}$ in $O(n)$ total space. To answer a query (P, k), we follow the steps below:

- Locate the locus node u_P of P via querying the string B-tree in $O(p/B + \log_B n)$ I/Os.
- Convert our top-k-form query $(u_P, k, \cdot)$ into a corresponding threshold-form query $(u_P, \cdot, \tau)$ using the structure in Lemma 13.
- Obtain $\mathcal{L}(u_P, \cdot, \tau)$ in $O(\log_B n + \log^*(n/B) + z/B)$ I/Os using the structure in Theorem 4, where $z = |\mathcal{L}(u_P, \cdot, \tau)|$.
- Finally, extract $\mathcal{L}(u_P, k, \cdot)$ from $\mathcal{L}(u_P, \cdot, \tau)$ in I/Os proportional to $|\mathcal{L}(u_P, \cdot, \tau)|/B = O((\log n + k)/B)$ as follows: select the kth highest scored link using a optimal external-memory selection algorithm [4, 23, 24] and discard every link that has score lower than that.

The total I/Os are $O(p/B + \log_B n + k/B + \log^*(n/B))$.

3.3.2 Proof of Theorem 2. Just modify the proof of Theorem 1 by replacing Theorem 4 with Theorem 5. This makes the total space $O(n \log^*(n/B))$ and query cost optimal.

3.3.3 Proof of Theorem 3. We prove the following lemma first.

Lemma 14. *For every integer g, there exists an $O(n)$-word structure, such that given any top-k-form query $(u_P, k, \cdot)$ with $k \leq g$, we can report the top-k links stabbed by u_P in the sorted order in $O(g/B)$ I/Os.*

Proof. The structure is constructed as follows. Identify all g-marked nodes. Then, for each g-marked node u^*, obtain the set $Q(u^*)$ by collecting all links in $nearLinks(u^*, g)$, $fringeLinks(u^*, g)$ and the top-g links in $farLinks(u^*, g)$. Therefore, $|Q(u^*)| = O(g)$ (refer to Lemma 10). Also, for all nodes u, $\mathcal{L}(u, g, \cdot) \subseteq Q(u^*)$, where u^* is the highest g-marked node in the subtree of u. We maintain

$Q(\cdot)$ for all g-marked nodes explicitly as a list of links sorted by scores. Additionally, for each maximal subtree containing no g-marked node, we maintain a sorted list of all links originating from the subtree. The total space is $n + g \cdot n/g = O(n)$. To answer a query $(u_P, k, \cdot)$ with u^* being the highest g-marked node in the subtree of u_P, simply go through the sorted list associated with u^* and report first k links that are stabbed by u_P. If there is no g-marked node in the subtree of u_P, then go through the sorted list associated with the maximal subtree (with no g marked node) containing u_P and report the first k links in that list stabbed by u_P. In both cases, the size of the list is bounded by $O(g)$; therefore, I/Os required is $O(g/B)$. □

To obtain the result in Theorem 3, we maintain a string B-tree over $\mathcal{D}$ and the structure in Lemma 14 for each $g \in \{B, 2B, 4B, \ldots, 2^{\lceil \log(d/B) \rceil} B\}$. The total space is $O(n \log(d/B))$ words. We answer a query $(u_P, k, \cdot)$ using the structure for $g = B2^{\lceil \log(k/B) \rceil}$ in $g/B = O(1 + k/B)$ I/Os. Combining this with the cost of the initial pattern search in the string B-tree gives the claimed result.

REFERENCES

[1] Peyman Afshani. 2008. On dominance reporting in 3D. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA'08), Lecture Notes in Computer Science*, Dan Halperin and Kurt Mehlhorn (Eds.), Vol. 5193. Springer, 41–51. https://doi.org/10.1007/978-3-540-87744-8_4

[2] Alok Aggarwal and Jeffrey Scott Vitter. 1988. The input/output complexity of sorting and related problems. *Commun. ACM* 31, 9 (1988), 1116–1127. https://doi.org/10.1145/48529.48535

[3] Lars Arge, Vasilis Samoladas, and Jeffrey Scott Vitter. 1999. On two-dimensional indexability and optimal range search indexing. In *Proceedings of the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Victor Vianu and Christos H. Papadimitriou (Eds.). ACM Press, 346–357. https://doi.org/10.1145/303976.304010

[4] Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. 1973. Time bounds for selection. *J. Comput. Syst. Sci.* 7, 4 (1973), 448–461. https://doi.org/10.1016/S0022-0000(73)80033-9

[5] Bernard Chazelle. 1990. Lower bounds for orthogonal range searching: I. the reporting case. *J. ACM* 37, 2 (1990), 200–212. https://doi.org/10.1145/77600.77614

[6] Paolo Ferragina and Roberto Grossi. 1999. The string B-tree: A new data structure for string search in external memory and its applications. *J. ACM* 46, 2 (1999), 236–280. https://doi.org/10.1145/301970.301973

[7] Arnab Ganguly, J. Ian Munro, Yakov Nekrich, Rahul Shah, and Sharma V. Thankachan. 2019. Categorical range reporting with frequencies. In *Proceedings of the 22nd International Conference on Database Theory (ICDT'19), LIPIcs'19*, Pablo Barceló and Marco Calautti (Eds.), Vol. 127. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 9:1–9:19. https://doi.org/10.4230/LIPIcs.ICDT.2019.9

[8] Wing-Kai Hon, Manish Patil, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. 2013. Indexes for document retrieval with relevance. In *Space-Efficient Data Structures, Streams, and Algorithms—Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday, Lecture Notes in Computer Science)*, Andrej Brodnik, Alejandro López-Ortiz, Venkatesh Raman, and Alfredo Viola (Eds.), Vol. 8066. Springer, 351–362. https://doi.org/10.1007/978-3-642-40273-9_22

[9] Wing-Kai Hon, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. 2014. Space-efficient frameworks for top-k string retrieval. *J. ACM* 61, 2 (2014), 9:1–9:36. https://doi.org/10.1145/2590774

[10] Wing-Kai Hon, Rahul Shah, and Jeffrey Scott Vitter. 2009. Space-efficient framework for top-k string retrieval problems *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS'09).* 713–722. https://doi.org/10.1109/FOCS.2009.19

[11] Kasper Green Larsen and Freek van Walderveen. 2013. Near-optimal range reporting structures for categorical data. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'13)*, Sanjeev Khanna (Ed.). SIAM, 265–276. https://doi.org/10.1137/1.9781611973105.20

[12] Yossi Matias, S. Muthukrishnan, Süleyman Cenk Sahinalp, and Jacob Ziv. 1998. Augmenting suffix trees, with applications *(ESA'98).* Springer-Verlag, London, UK, 67–78.

[13] J. Ian Munro, Gonzalo Navarro, Rahul Shah, and Sharma V. Thankachan. 2020. Ranked document selection. *Theor. Comput. Sci.* 812 (2020), 149–159. https://doi.org/10.1016/j.tcs.2019.10.008

[14] S. Muthukrishnan. 2002. Efficient algorithms for document retrieval problems. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, David Eppstein (Ed.). ACM/SIAM, 657–666.

[15] Gonzalo Navarro. 2013. Spaces, trees, and colors: The algorithmic landscape of document retrieval on sequences. *ACM Comput. Surv.* 46, 4 (2013), 52:1–52:47. https://doi.org/10.1145/2535933

[16] Gonzalo Navarro and Yakov Nekrich. 2012. Top-k document retrieval in optimal time and linear space. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'12)*, Yuval Rabani (Ed.). SIAM, 1066–1077. https://doi.org/10.1137/1.9781611973099.84

[17] Gonzalo Navarro and Yakov Nekrich. 2017. Time-optimal top-k document retrieval. *SIAM J. Comput.* 46, 1 (2017), 80–113. https://doi.org/10.1137/140998949

[18] Gonzalo Navarro and Kunihiko Sadakane. 2014. Fully functional static and dynamic succinct trees. *ACM Trans. Algor.* 10, 3 (2014), 16:1–16:39. https://doi.org/10.1145/2601073

[19] Yakov Nekrich. 2014. Efficient range searching for categorical and plain data. *ACM Trans. Datab. Syst.* 39, 1 (2014), 9:1–9:21. https://doi.org/10.1145/2543924

[20] Manish Patil, Sharma V. Thankachan, Rahul Shah, Wing-Kai Hon, Jeffrey Scott Vitter, and Sabrina Chandrasekaran. 2011. Inverted indexes for phrases and strings *(SIGIR'11)*. 555–564. https://doi.org/10.1145/2009916.2009992

[21] Manish Patil, Sharma V. Thankachan, Rahul Shah, Yakov Nekrich, and Jeffrey Scott Vitter. 2014. Categorical range maxima queries. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'14)*, Richard Hull and Martin Grohe (Eds.). ACM, 266–277. https://doi.org/10.1145/2594538.2594557

[22] Rahul Shah, Cheng Sheng, Sharma V. Thankachan, and Jeffrey Scott Vitter. 2013. Top-k document retrieval in external memory. In *21st Annual European Symposium on Algorithms (ESA'13), Lecture Notes in Computer Science*, Hans L. Bodlaender and Giuseppe F. Italiano (Eds.), Vol. 8125. Springer, 803–814. https://doi.org/10.1007/978-3-642-40450-4_68

[23] Jop F. Sibeyn. 2006. External selection. *J. Algor.* 58, 2 (2006), 104–117. https://doi.org/10.1016/j.jalgor.2005.02.002

[24] Yufei Tao. 2014. *Lecture 1: External Memory Model and Sorting*. Lecture Notes. Chinese University of Hong Kong.

[25] Justin Zobel and Alistair Moffat. 2006. Inverted files for text search engines. *ACM Comput. Surv.* 38, 2, Article 6 (July 2006). https://doi.org/10.1145/1132956.1132959

Received 11 May 2020; revised 24 August 2022; accepted 24 August 2022