Top-k Term-Proximity in Succinct Space

J. Ian Munro[1], Gonzalo Navarro[2], Jesper Sindahl Nielsen[3], Rahul Shah[4], and Sharma V. Thankachan[5](✉)

[1] Cheriton School of CS, University of Waterloo, Waterloo, Canada
imunro@uwaterloo.ca
[2] Department of CS, University of Chile, Santiago, Chile
gnavarro@dcc.uchile.cl
[3] MADALGO, Aarhus University, Aarhus, Denmark
jasn@cs.au.dk
[4] School of EECS, Louisiana State University, Louisiana, USA
rahul@csc.lsu.edu
[5] School of CSE, Georgia Institute of Technology, Georgia, USA
sharma.thankachan@gmail.com

Abstract. Let $\mathcal{D} = \{\mathsf{T}_1, \mathsf{T}_2, \ldots, \mathsf{T}_D\}$ be a collection of D string documents of n characters in total, that are drawn from an alphabet set $\Sigma = [\sigma]$. The *top-k d*ocument retrieval problem is to preprocess $\mathcal{D}$ into a data structure that, given a query $(P[1..p], k)$, can return the k documents of $\mathcal{D}$ most relevant to pattern P. The relevance is captured using a predefined ranking function, which depends on the set of occurrences of P in T_d. For example, it can be the term frequency (i.e., the number of occurrences of P in T_d), or it can be the term proximity (i.e., the distance between the closest pair of occurrences of P in T_d), or a pattern-independent importance score of T_d such as PageRank. Linear space and optimal query time solutions already exist for this problem. Compressed and compact space solutions are also known, but only for a few ranking functions such as term frequency and importance. However, space efficient data structures for term proximity based retrieval have been evasive. In this paper we present the first sub-linear space data structure for this relevance function, which uses only $o(n)$ bits on top of any compressed suffix array of $\mathcal{D}$ and solves queries in time $O((p + k) \operatorname{polylog} n)$.

1 Introduction

Ranked document retrieval, that is, returning the documents that are most relevant to a query, is the fundamental task in Information Retrieval (IR) [1,6]. Muthukrishnan [19] initiated the study of this family of problems in the more general scenario where both the documents and the queries are general strings over arbitrary alphabets, which has applications in several areas [20]. In this scenario, we have a collection $\mathcal{D} = \{\mathsf{T}_1, \mathsf{T}_2, \ldots, \mathsf{T}_D\}$ of D string documents of total length n, drawn from an alphabet $\Sigma = [\sigma]$, and the query is a pattern $P[1..p]$

Funded in part by NSERC of Canada and the Canada Research Chairs program, Fondecyt Grant 1-140796, Chile, and NSF Grants CCF–1017623, CCF–1218904.

H.-K. Ahn and C.-S. Shin (Eds.): ISAAC 2014, LNCS 8889, pp. 169–180, 2014.
DOI: 10.1007/978-3-319-13075-0_14

over Σ. Muthukrishnan considered a family of problems called *thresholded* document listing: given an additional parameter K, list only the documents where some function $\mathsf{score}(P, d)$ of the occurrences of P in T_d exceeded K. For example, the *document mining* problem aims to return the documents where P appears at least K times, whereas the *repeats* problem aims to return the documents where two occurrences of P appear at distance at most K. While document mining has obvious connections with typical term-frequency measures of relevance [1,6], the repeats problem is more connected to various problems in bioinformatics [4,12]. Also notice that the repeats problem is closely related to the term proximity based document retrieval in IR field [5,29,32–34]. Muthukrishnan achieved optimal time for both problems, with $O(n)$ space (in words) if K is specified at indexing time and $O(n \log n)$ if specified at query time.

A more natural version of the thresholded problems, as used in IR, is *top-k retrieval*: Given P and k, return k documents with the best $\mathsf{score}(P, d)$ values. Hon et al. [15,16] gave a general framework to solve top-k problems for a wide variety of $\mathsf{score}(P, d)$ functions, which takes $O(n)$ space, allows k to be specified at query time, and solves queries in $O(p + k \log k)$ time. Navarro and Nekrich [22] reduced the time to $O(p + k)$, and finally Shah et al. [30] achieved time $O(k)$ given the locus of P in the generalized suffix tree of $\mathcal{D}$. Recently, Munro et al. [18] introduced an $O(n)$-word index, that can find the top-kth document in $O(\log k)$ time, once the locus of P is given.

The problem is far from closed, however. Even the $O(n)$ space (i.e., $O(n \log n)$ bits) is excessive compared to the size of the text collection itself ($n \log \sigma$ bits), and in data-intensive scenarios it often renders all these solutions impractical by a wide margin. Hon et al. [16] also introduced a general framework for *succinct indexes*, which use $o(n)$ bits[1] on top of a *compressed suffix array* (CSA) [21], which represents $\mathcal{D}$ in a way that also provides pattern-matching functionalities on it, all within space ($|\mathsf{CSA}|$) close to that of the *compressed* collection. A CSA finds the suffix array interval of $P[1..p]$ in time $t_{\mathsf{s}}(p)$ and retrieves any cell of the suffix array or its inverse in time t_{SA}. Hon et al. achieved $O(t_{\mathsf{s}}(p) + k\, t_{\mathsf{SA}} \log^{3+\varepsilon} n)$ query time, using $O(n/\log^{\varepsilon} n)$ bits. Subsequent work (see [20,26]) improved the initial result up to $O(t_{\mathsf{s}}(p) + k\, t_{\mathsf{SA}} \log^2 k \log^{\varepsilon} n)$ [24], and also considered *compact* indexes, which may use $o(n \log n)$ bits on top of the CSA. For example, these achieve $O(t_{\mathsf{s}}(p) + k\, t_{\mathsf{SA}} \log k \log^{\varepsilon} n)$ query time using $n \log \sigma + o(n)$ further bits [14], or $O(t_{\mathsf{s}}(p) + k \log^* k)$ query time using $n \log D + o(n \log n)$ further bits [25].

However, all these succinct and compact indexes work *exclusively* for the term frequency (or closely related, e.g., TF-IDF) measure of relevance. For the simpler case where documents have a fixed relevance independent of P, succinct indexes achieve $O(t_{\mathsf{s}}(p) + k\, t_{\mathsf{SA}} \log k \log^{\varepsilon} n)$ query time [3], and compact indexes using $n \log D + o(n \log D)$ bits achieve $O(t_{\mathsf{s}}(p) + k \log(D/k))$ time [10]. On the other hand, there have been *no succinct nor compact indexes for the term proximity measure of relevance*, $\mathsf{tp}(P, d) = \min\{\{|i-j| > 0, \mathsf{T}_d[i..i+p-1] = \mathsf{T}_d[j..j+p-1] = P\} \cup \{\infty\}\}$. In this paper we introduce the first such result as follows.

[1] If $D = o(n)$, which we assume for simplicity in this paper. Otherwise it is $D \log(n/D) + O(D) + o(n)$ bits.

Theorem 1. *Using a* CSA *plus* $o(n)$ *bits data structure, one can answer top-*k *term proximity queries in* $O(t_s(p) + (\log^2 n + k(t_{SA} + \log k \log n)) \log^{2+\varepsilon} n)$ *time, for any constant* $\varepsilon > 0$.

2 Basic Concepts

Let $\mathsf{T}[1..n] = \mathsf{T}_1 \circ \mathsf{T}_2 \circ \cdots \mathsf{T}_D$ be the text (from an alphabet $\Sigma = [\sigma] \cup \{\$\}$) obtained by concatenating all the documents in $\mathcal{D}$. Each document is terminated with a special symbol \$, which does not appear anywhere else. A suffix $\mathsf{T}[i..n]$ of T belongs to T_d iff i is in the region corresponding to T_d in T. Thus, it holds $d = 1 + rank_B(i-1)$, where $B[1..n]$ is a bitmap defined as $B[j] = 1$ iff $\mathsf{T}[j] = \$$ and $rank_B(i-1)$ is the number of 1s in $B[1..i-1]$. This operation is computed in $O(1)$ time on a representation of B that uses $D \log(n/D) + O(D) + o(n)$ bits [28]. For simplicity, we assume $D = o(n)$, and thus B uses $o(n)$ bits.

Suffix Tree [31] of T is a compact trie containing all of its suffixes, where the ith leftmost leaf, ℓ_i, represents the ith lexicographically smallest suffix. It is also called the generalized suffix tree of $\mathcal{D}$, GST. Each edge in GST is labeled by a string, and $path(x)$ is the concatenation of the edge labels along the path from the GST root to node x. Then $path(\ell_i)$ is the ith lexicographically smallest suffix of T. The highest node x with $path(x)$ prefixed by $P[1..p]$ is the *locus* of P, and is found in time $O(p)$ from the GST root. The GST uses $O(n)$ words of space.

Suffix Array [17] of T, $\mathsf{SA}[1..n]$, is defined as $\mathsf{SA}[i] = n + 1 - |path(\ell_i)|$, the starting position in T of the ith lexicographically smallest suffix of T. The *suffix range* of P is the range $\mathsf{SA}[sp, ep]$ pointing to the suffixes that start with P, $\mathsf{T}[\mathsf{SA}[i]..\mathsf{SA}[i] + p - 1] = P$ for all $i \in [sp, ep]$. Also, ℓ_{sp} (resp., ℓ_{ep}) are the leftmost (resp., rightmost) leaf in the subtree of the locus of P.

Compressed Suffix Array [8,11,21] of T, CSA, is a compressed representation of SA, and usually also of T. Its size in bits, $|\mathsf{CSA}|$, is $O(n \log \sigma)$ and usually much less. The CSA finds the interval $[sp, ep]$ of P in time $t_s(p)$. It can output any value $\mathsf{SA}[i]$, and even of its inverse permutation, $\mathsf{SA}^{-1}[i]$, in time t_{SA}. For example, a CSA using $nH_h(\mathsf{T}) + o(n \log \sigma)$ bits [2] gives $t_s(p) = O(p)$ and $t_{SA} = O(\log^{1+\epsilon} n)$ for any constant $\epsilon > 0$, where H_h is the hth order empirical entropy.

Compressed Suffix Tree of T, CST, is a compressed representation of GST, where node identifiers are their corresponding suffix array ranges. The CST can use $o(n)$ bits on top of a CSA [23] and compute (among others) the lowest common ancestor (LCA) of two leaves ℓ_i and ℓ_j, in time $O(t_{SA} \log^{\epsilon} n)$, and the Weiner link $\mathsf{Wlink}(a, v)$, which leads to the node with path label $a \circ path(v)$, in time $O(t_{SA})$.[2]

Orthogonal Range Successor/Predecessor. Given n points in $[n] \times [n]$, an $O(n \log n)$-bit data structure can retrieve the point in a given rectangle with lowest

[2] Using $O(n/\log^{\epsilon} n)$ bits and no special implementation for operations $\mathsf{SA}^{-1}[\mathsf{SA}[i] \pm 1]$.

y-coordinate value, in time $O(\log^{\epsilon} n)$ for any constant $\epsilon > 0$ [27]. Combined with standard range tree partitioning, the following result easily follows.

Lemma 1. *Given n' points in $[n] \times [n] \times [n]$, a structure using $O(n' \log^2 n)$ bits can support the following query in $O(\log^{1+\epsilon} n)$ time, for any constant $\epsilon > 0$: find the point in a region $[x, x'] \times [y, y'] \times [z, z']$ with the lowest/highest x-coordinate.*

3 An Overview of Our Data Structure

The top-k term proximity is related to a problem called *range restricted searching*, where one must report all the occurrences of P that are within a text range $\mathsf{T}[i..j]$. It is known that succinct data structures for that problem are unlikely to exist in general, whereas indexes of size $|\mathsf{CSA}| + O(n/\log^{\epsilon} n)$ bits do exist for patterns longer than $\Delta = \log^{2+\epsilon} n$ (see [13]). Therefore, our basic strategy will be to have a separate data structure to solve queries of length $p = \pi$, for each $\pi \in \{1, \ldots, \Delta\}$. Patterns with length $p > \Delta$ can be handled with a single succinct data structure. More precisely, we design two different data structures that operate on top of a CSA:

– An $O(n \log\log n/(\pi \log^{\gamma} n))$-bits structure for handling queries of fixed length $p = \pi$, in time $O(t_{\mathsf{s}}(p) + k(t_{\mathsf{SA}} + \log\log n + \log k)\, \pi \log^{\gamma} n)$.
– An $O(n/\log^{\epsilon} n + (n/\Delta) \log^2 n)$-bits structure for handling queries with $p > \Delta$ in time $O(t_{\mathsf{s}}(p) + \Delta(\Delta + t_{\mathsf{SA}}) + k \log k \log^{2\epsilon} n (t_{\mathsf{SA}} + \Delta \log^{1+\epsilon} n))$.

By building the first structure for every $\pi \in \{1, \ldots, \Delta\}$, any query can be handled using the appropriate structure. The Δ structures for fixed pattern length add up to $O(n(\log\log n)^2/ \log^{\gamma} n) = o(n/\log^{\gamma/2} n)$ bits, whereas that for long patterns uses $O(n/\log^{\epsilon} n)$ bits. By choosing $\varepsilon = 4\epsilon = 2\gamma$, the space is $O(n/\log^{\varepsilon/4} n)$ bits. As for the time, the structures for fixed $p = \pi$ are most costly for $\pi = \Delta$, where their time is $k(t_{\mathsf{SA}} + \log\log n + \log k)\, \Delta \log^{\gamma} n$. Adding up the time of the second structure, we get $O(t_{\mathsf{s}}(p) + \Delta(\Delta + k(t_{\mathsf{SA}} + \log k \log^{1+\epsilon} n) \log^{2\epsilon} n)$, which is upper bounded by $O(t_{\mathsf{s}}(p) + (\log^2 n + k(t_{\mathsf{SA}} + \log k \log n)) \log^{2+\varepsilon} n)$. This yields Theorem 1.

Now we introduce some formalization to convey the key intuition. The term proximity $\mathsf{tp}(P, d)$ can be determined by just two occurrences of P in T_d, which are the closest up to ties. We call them *critical occurrences*, and a pair of two closest occurrences is a *critical pair*. There can be multiple critical pairs.

Definition 1. *An integer $i \in [1, n]$ is an* occurrence *of P in T_d if the suffix $\mathsf{T}[i..n]$ belongs to T_d and $\mathsf{T}[i..i + p - 1] = P[1..p]$. The set of all occurrences of P in T is called $\mathsf{Occ}(P)$.*

Definition 2. *An occurrence i_d of P in T_d is a* critical occurrence *if there exists another occurrence i'_d of P in T_d such that $|i_d - i'_d| = \mathsf{tp}(P, d)$. The pair (i_d, i'_d) is called a* critical pair *of T_d with respect to P.*

A key concept in our solution is that of *candidate sets* of occurrences, which contain sufficient information to solve the top-k query (note that, due to ties, a top-k query may have multiple valid answers).

Definition 3. *Let* $\mathsf{Topk}(P,k)$ *be a valid answer for the top-k query* (P,k)*. A set* $\mathsf{Cand}(P,k) \subseteq \mathsf{Occ}(P)$ *is a* candidate set *of* $\mathsf{Topk}(P,k)$ *if, for each document identifier* $d \in \mathsf{Topk}(P,k)$*, there exists a critical pair* (i_d, i'_d) *of* T_d *with respect to* P *such that* $i_d, i'_d \in \mathsf{Cand}(P,k)$*.*

Lemma 2. *Given a* CSA *on* $\mathcal{D}$*, a valid answer to query* (P,k) *can be computed from* $\mathsf{Cand}(P,k)$ *in* $O(z \log z)$ *time, where* $z = |\mathsf{Cand}(P,k)|$*.*

Proof. Sort the set $\mathsf{Cand}(P,k)$ and traverse it sequentially. From the occurrences within each document T_d, retain the closest consecutive pair (i_d, i'_d), and finally report k documents with minimum values $|i_d - i'_d|$. This takes $O(z \log z)$ time.

We show that this returns a valid answer set. Since $\mathsf{Cand}(P,k)$ is a candidate set, it contains a critical pair (i_d, i'_d) for $d \in \mathsf{Topk}(P,k)$, so this critical pair (or another with the same $|i_d - i'_d|$ value) is chosen for each $d \in \mathsf{Topk}(P,k)$. If the algorithm returns an answer other than $\mathsf{Topk}(P,k)$, it is because some document $d \in \mathsf{Topk}(P,k)$ is replaced by another $d' \notin \mathsf{Topk}(P,k)$ with the same score $\mathsf{tp}(P,d') = |i_{d'} - i'_{d'}| = |i_d - i'_d| = \mathsf{tp}(d)$. ☐

Our data structures aim to return a small candidate set (as close to size k as possible), from which a valid answer is efficiently computed using Lemma 2.

4 Data Structure for Queries with Fixed $p = \pi \leq \Delta$

We build an $o(n/\pi)$-bits structure for handling queries with pattern length $p = \pi$.

Lemma 3. *There is an* $O(n \log\log n/(\pi \log^\gamma n))$*-bits data structure solving queries* $(P[1..p],k)$ *with* $p = \pi$ *in* $O(t_{\mathsf{s}}(p) + k(t_{\mathsf{SA}} + \log\log n + \log k)\pi \log^\gamma n)$ *time.*

The idea is to build an array $\mathsf{F}[1,n]$ such that a candidate set of size $O(k)$, for any query (P,k) with $p = \pi$, is given by $\{\mathsf{SA}[i], i \in [sp, ep] \wedge \mathsf{F}[i] \leq k\}$, $[sp, ep]$ being the suffix range of P. The key property to achieve this is that the ranges $[sp, ep]$ are disjoint for all the patterns of a fixed length π. We build F as follows.

1. Initialize $\mathsf{F}[1..n] = n + 1$.
2. For each pattern Q of length π,
   (a) Find the suffix range $[\alpha, \beta]$ of Q.
   (b) Find the list $\mathsf{T}_{r_1}, \mathsf{T}_{r_2}, \mathsf{T}_{r_3}, \ldots$ of documents in the ascending order of $\mathsf{tp}(Q, \cdot)$ values (ties broken arbitrarily).
   (c) For each document T_{r_κ} containing Q at least twice, choose a *unique* critical pair with respect to Q, that is, choose two elements $j, j' \in [\alpha, \beta]$, such that $(i_{r_\kappa}, i'_{r_\kappa}) = (\mathsf{SA}[j], \mathsf{SA}[j'])$ is a critical pair of T_{r_κ} with respect to Q. Then assign $\mathsf{F}[j] = \mathsf{F}[j'] = \kappa$.

The following observation is immediate.

Lemma 4. *For a query $(P[1..p], k)$ with $p = \pi$ and suffix array range $[sp, ep]$ for P, the set $\{SA[j], j \in [sp, ep] \wedge \mathsf{F}[j] \leq k\}$ is a candidate set of size at most $2k$.*

Proof. A valid answer for (P, k) are the document identifiers $r_1, \ldots, r_k$ considered at construction time for $Q = P$. For each such document T_{r_κ}, $1 \leq \kappa \leq k$, we have found a critical pair $(i_{r_\kappa}, i'_{r_\kappa}) = (\mathsf{SA}[j], \mathsf{SA}[j'])$, for $j, j' \in [sp, ep]$, and set $\mathsf{F}[j] = \mathsf{F}[j'] = \kappa \leq k$. All the other values of $\mathsf{F}[sp, ep]$ are larger than k (or ∞). The size of the candidate set is thus at most $2k$ (or less, if there are less than k documents where P occurs twice). □

However, we cannot afford to maintain F explicitly within the desired space bounds. Therefore, we replace F by a *sampled* array F'. The sampled array is built by cutting F into blocks of size $\pi' = \pi \log^\gamma n$ and storing the logarithm of the minimum value for each block. This will increase the size of the candidate sets by a factor π'. More precisely, $\mathsf{F}'[1, n/\pi']$ is defined as

$$\mathsf{F}'[j] \quad = \quad \lceil \log \min F[(j-1)\pi' + 1..j\pi'] \rceil \, .$$

Since $\mathsf{F}'[j] \in [0.. \log n]$, the array can be represented using $n \log \log n / \log^\gamma n$ bits. We maintain F' with a multiary wavelet tree [9], which maintains the space in $O(n \log \log n / \log^\gamma n)$ bits and, since the alphabet size is logarithmic, supports in constant time operations *rank* and *select* on F'. Operation $rank(j, \kappa)$ counts the number of occurrences of κ in $\mathsf{F}'[1..j]$, whereas $select(j, \kappa)$ gives the position of the jth occurrence of κ in F'.

Query Algorithm. To answer a query $(P[1..p], k)$ with $p = \pi$ using a CSA and F', we compute the suffix range $[sp, ep]$ of P in time $t_{\mathsf{s}}(p)$, and then do as follows.

1. Among all the blocks of F overlapping the range $[sp, ep]$, identify those containing an element $\leq 2^{\lceil \log k \rceil}$, that is, compute the set
$$S_{blocks} = \{j, \lceil sp/\pi' \rceil \leq j \leq \lceil ep/\pi' \rceil \wedge \mathsf{F}'[j] \leq \lceil \log k \rceil\}.$$
2. Generate $\mathsf{Cand}(P, k) = \{\mathsf{SA}[j'], j \in S_{blocks} \wedge j' \in [(j-1)\pi' + 1, j\pi']\}$.
3. Find the query output from the candidate set $\mathsf{Cand}(P, k)$, using Lemma 2.

For step 1, the wavelet tree representation of F' generates S_{blocks} in time $O(1 + |S_{blocks}|)$: All the 2^t positions[3] $j \in [sp, ep]$ with $\mathsf{F}'[j] = t$ are $j = select(rank(sp - 1, t) + i, t)$ for $i \in [1, 2^t]$. We notice if there are no sufficient documents if we obtain a $j > ep$, in which case we stop.

The set $\mathsf{Cand}(P, k)$ is a candidate set of (P, k), since any $j \in [sp, ep]$ with $\mathsf{F}[j] \leq k$ belongs to some block of S_{blocks}. Also the number of $j \in [sp, ep]$ with $\mathsf{F}[j] \leq 2^{\lceil \log k \rceil}$ is at most $2 \cdot 2^{\lceil \log k \rceil} \leq 4k$, therefore $|S_{blocks}| \leq 4k$.

Now, $\mathsf{Cand}(P, k)$ is of size $|S_{blocks}|\pi' = O(k\pi')$, and it is generated in step 2 in time $O(k \, t_{\mathsf{SA}} \, \pi')$. Finally, the time for generating the final output using Lemma 2 is $O(k\pi' \log(k\pi'))) = O(k\pi \log^\gamma n(\log k + \log \log n + \log \pi))$. By considering that $\pi \leq \Delta = \log^{2+\epsilon} n$, we obtain Lemma 3.

[3] Except for $t = 0$, which has 2 positions.

5 Data Structure for Queries with $p > \Delta$

We prove the following result in this section.

Lemma 5. *There is an $O(n/\log^{\epsilon} n + (n/\Delta)\log^2 n)$-bits structure solving queries $(P[1..p], k)$, with $p > \Delta$, in $O(t_{\mathsf{s}}(p) + \Delta(\Delta + t_{\mathsf{SA}}) + k \log k \log^{2\epsilon} n(t_{\mathsf{SA}} + \Delta \log^{1+\epsilon} n))$ time.*

We start with a concept similar to that of a candidate set, but weaker in the sense that it is required to contain only one element of each critical pair.

Definition 4. *Let $\mathsf{Topk}(P, k)$ be a valid answer for the top-k query (P, k). A set $\mathsf{Semi}(P, k) \subseteq [n]$ is a* semi-candidate set *of $\mathsf{Topk}(P, k)$ if it contains at least one critical occurrence i_d of P in T_d for each document identifier $d \in \mathsf{Topk}(P, k)$.*

Our structure in this section generates a semi-candidate set $\mathsf{Semi}(P, k)$. Then, a candidate set $\mathsf{Cand}(P, k)$ is generated as the union of $\mathsf{Semi}(P, k)$ and the set of occurrences of P that are immediately before and immediately after every position $i \in \mathsf{Semi}(P, k)$. This is obviously a valid candidate set. Finally, we apply Lemma 2 on $\mathsf{Cand}(P, k)$ to compute the final output.

5.1 Generating a Semi-candidate Set

This section proves the following result.

Lemma 6. *A structure of $O(n(\log\log n)^2/\log^{\delta} n)$ bits plus a CSA can generate a semi-candidate set of size $O(k \log k \log^{\delta} n)$ in time $O(t_{\mathsf{SA}}\, k \log k \log^{\delta} n)$.*

Let $\mathsf{Leaf}(x)$ (resp., $\mathsf{Leaf}(y)$) be the set of leaves in the subtree of node x (resp., y) in GST, $\mathsf{Leaf}(x \backslash y) = \mathsf{Leaf}(x) \setminus \mathsf{Leaf}(y)$. The following lemma holds.

Lemma 7. *The set $\mathsf{Semi}(path(y), k) \cup \{\mathsf{SA}[j], \ell_j \in \mathsf{Leaf}(x\backslash y)\}$ is a semi-candidate set of $(path(x), k)$.*

Proof. Let $d \in \mathsf{Topk}(path(x), k)$, then our semi-candidate set should contain i_d or i'_d for some critical pair (i_d, i'_d). If there is some such critical pair where i_d or i'_d are occurrences of $path(x)$ but not of $path(y)$, then ℓ_j or $\ell_{j'}$ are in $L(x\backslash y)$, for $\mathsf{SA}[j] = i_d$ and $\mathsf{SA}[j'] = i'_d$, and thus our set contains it. If, on the other hand, both i_d and i'_d are occurrences of $path(y)$ for all critical pairs (i_d, i'_d), then $\mathsf{tp}(path(y), d) = \mathsf{tp}(path(x), d)$, and the critical pairs of $path(x)$ are the critical pairs of $path(y)$. Thus $\mathsf{Semi}(y, k)$ contains i_d or i'_d for some such critical pair. □

Our approach is to precompute and store $\mathsf{Semi}(path(y), k)$ for carefully selected nodes $y \in \mathsf{GST}$ and k values, so that any arbitrary $\mathsf{Semi}(path(x), k)$ set can be computed efficiently. The succinct framework of Hon et al. [16] is adequate for this.

Node Marking Scheme. The idea [16] is to mark a set Mark_g of nodes in GST based on a *grouping factor* g: Every gth leaf is marked, and the LCA of any two consecutive marked leaves is also marked. Then the following properties hold.

1. $|\mathsf{Mark}_g| \leq 2n/g$.
2. If there exists no marked node in the subtree of x, then $|\mathsf{Leaf}(x)| < 2g$.
3. If it exists, then the highest marked descendant node y of any unmarked node x is unique, and $|\mathsf{Leaf}(x \backslash y)| < 2g$.

We use this idea, and a later refinement [14]. Let us first consider a variant of Lemma 6 where $k = \kappa$ is fixed at construction time. We use a CSA and an $O(n/\log^{\delta} n)$-bit CST on it, see Section 2. We choose $g = \kappa \log \kappa \log^{1+\delta} n$ and, for each node $y \in \mathsf{Mark}_g$, we explicitly store a candidate set $\mathsf{Semi}(\mathit{path}(y), \kappa)$ of size κ. The space required is $O(|\mathsf{Mark}_g|\kappa \log n) = O(n/(\log \kappa \log^{\delta} n))$ bits.

To solve a query (P, κ), we find the suffix range $[sp, ep]$, then the locus node of P is $x = \mathsf{LCA}(\ell_{sp}, \ell_{ep})$. Then we find $y = \mathsf{LCA}(\ell_{g\lceil sp/g \rceil}, \ell_{g\lfloor ep/g \rfloor})$, the highest marked node in the subtree of x. Then, by the given properties of the marking scheme, combined with Lemma 7, a semi-candidate set of size $O(g + \kappa) = O(\kappa \log \kappa \log^{1+\delta} n)$ can be generated in $O(t_{\mathsf{SA}} \kappa \log \kappa \log^{1+\delta} n)$ time.

To reduce this time, we employ dual marking scheme [14]. We identify a larger set $\mathsf{Mark}_{g'}$ of nodes, for $g' = \kappa \log \kappa \log^{\delta} n$. To avoid confusion, we call these *prime* nodes, not marked nodes. For each node $y' \in \mathsf{Mark}_{g'}$, we precompute a candidate set $\mathsf{Semi}(\mathit{path}(y'), \kappa)$ of size κ. Let y be the (unique) highest marked node in the subtree of y'. Then we store κ bits in y' to indicate which of the κ nodes stored in $\mathsf{Semi}(\mathit{path}(y), \kappa)$ also belong to $\mathsf{Semi}(\mathit{path}(y'), \kappa)$. By the same proof of Lemma 7, elements in $\mathsf{Semi}(\mathit{path}(y'), \kappa) \setminus \mathsf{Semi}(\mathit{path}(y), \kappa)$ must have a critical occurrence in $\mathsf{Leaf}(y' \backslash y)$. Then, instead of explicitly storing the critical positions $i_d \in \mathsf{Semi}(\mathit{path}(y'), \kappa) \setminus \mathsf{Semi}(\mathit{path}(y), \kappa)$, we store their left-to-right position in $\mathsf{Leaf}(y' \backslash y)$. Storing κ such positions in leaf order requires $O(\kappa \log(g/\kappa)) = O(\kappa \log \log n)$ bits, using for example gamma codes. The total space is $O(|\mathsf{Mark}_{g'}|\kappa \log \log n) = O(n \log \log n/(\log \kappa \log^{\delta}))$ bits.

Now we can apply the same technique to obtain a semi-candidate set from $\mathsf{Mark}_{g'}$, yet of smaller size $O(g' + \kappa) = O(\kappa \log \kappa \log^{\delta} n)$, in time $O(t_{\mathsf{SA}} \kappa \log \kappa \log^{\delta} n)$.

We are now ready to complete the proof Lemma 6. We maintain structures as described for all the values of κ that are powers of 2, in total $O((n \log \log n/\log^{\delta} n) \cdot \sum_{i=1}^{\log D} 1/i) = O(n(\log \log n)^2/\log^{\delta} n)$ bits of space. To solve a query (P, k), we compute $\kappa = 2^{\lceil \log k \rceil} < 2k$ and return the semi-candidate set of (P, κ) using the corresponding structure.

5.2 Generating the Candidate Set

The problem boils down to the task of, given $P[1..p]$ and an occurrence q, finding the occurrence of P closest to q. In other words, finding the first and the last occurrence of P in $\mathsf{T}[q+1..n]$ and $\mathsf{T}[1..q+p-1]$, respectively. We employ suffix sampling to obtain the desired space-efficient structure. The idea is to exploit the fact that, if $p > \Delta$, then for every occurrence q of P there must be an integer $j = \Delta\lceil q/\Delta \rceil$ (a multiple of Δ) and $t \leq \Delta$, such that $P[1..t]$ is a suffix of $\mathsf{T}[1..j]$ and $P[t+1..p]$ is a prefix of $\mathsf{T}[j+1..n]$. We call q an *offset-t occurrence* of P. Then, $\mathsf{Cand}(P, k)$ can be computed as follows:

1. Find $\mathsf{Semi}(P, k)$ using Lemma 6.
2. For each $q \in \mathsf{Semi}(P, k)$ and $t \in [1, \Delta]$, find the offset-t occurrences of P that are immediately before and immediately after q.
3. The occurrences found in the previous step, along with the elements in $\mathsf{Semi}(P, k)$, constitute $\mathsf{Cand}(P, k)$.

In order to perform step 2 efficiently, we maintain the following structures.

– **Sparse Suffix Tree (SST)**: A suffix $\mathsf{T}[\Delta i + 1..n]$ is a *sparse suffix*, and the trie of all sparse suffixes is a *sparse suffix tree*. The *sparse suffix range* of a pattern Q is the range of the sparse suffixes in SST that are prefixed by Q. Given the suffix range $[sp, ep]$ of a pattern, its sparse suffix range $[ssp, sep]$ can be computed in constant time by maintaining a bitmap $B[1..n]$, where $B[j] = 1$ iff $\mathsf{T}[\mathsf{SA}[j]..n]$ is a sparse suffix. Then $ssp = 1 + rank_B(sp - 1)$ and $sep = rank_B(sp)$. Since B has n/Δ 1s, it can be represented in $O((n/\Delta) \log \Delta)$ bits while supporting $rank_B$ operation in constant time for any $\Delta = O(\text{polylog}\, n)$ [28].
– **Sparse Prefix Tree (SPT)**: A prefix $\mathsf{T}[1..\Delta i]$ is a *sparse prefix*, and the trie of the *reverses* of all sparse prefixes is a *sparse prefix tree*. The *sparse prefix range* of a pattern Q is the range of the sparse prefixes in SPT with Q as a suffix. The SPT can be represented as a blind trie [7] using $O((n/\Delta) \log n)$ bits. Then the search for the sparse prefix range of Q can be done in $O(|Q|)$ time, by descending using the reverse of Q[4]. Note that the blind trie may return a fake node when Q does not exist in the SPT.
– **Orthogonal Range Successor/Predecessor Search Structure** over a set of $\lceil n/\Delta \rceil$ points of the form (x, y, z), where the yth leaf in SST corresponds to $\mathsf{T}[x..n]$ and the zth leaf in SPT corresponds to $\mathsf{T}[1..(x-1)]$. The space needed is $O((n/\Delta) \log^2 n)$ bits (recall Lemma 1).

The total space of the structures is $O((n/\Delta) \log^2 n)$ bits. They allow computing first offset-t occurrence of P in $\mathsf{T}[q+1..n]$ as follows: find $[ssp_t, sep_t]$ and $[ssp'_t, sep'_t]$, the sparse suffix range of $P[t+1..p]$ and the sparse prefix range of $P[1..t]$, respectively. Then, using an orthogonal range successor query, find the point $(e, \cdot, \cdot)$ with the lowest x-coordinate value in $[q+t+1, n] \times [ssp_t, sep_t] \times [ssp'_t, sep'_t]$. Then, $e-t$ is the answer. Similarly, the last offset-t occurrence of P in $\mathsf{T}[1..q-1]$ is $f-t$, where $(f, \cdot, \cdot)$ is the point in $[1, q+t-1] \times [ssp_t, sep_t] \times [ssp'_t, sep'_t]$ with the highest x-coordinate value.

First, we compute all the ranges $[ssp_t, sep_t]$ using the SST. This requires knowing the interval $\mathsf{SA}[sp_t, ep_t]$ of $P[t+1..p]$ for all $1 \le t \le \Delta$. We compute these by using the CSA to search for $P[\Delta+1..p]$ (in time at most $t_{\mathsf{s}}(p)$), which gives $[sp_\Delta, ep_\Delta]$, and then computing $[sp_{t-1}, ep_{t-1}] = \mathsf{Wlink}(P[t], [sp_t, ep_t])$ for $t = \Delta - 1, \ldots, 1$. Using an $o(n)$-bits CST (see Section 2), this takes $O(\Delta\, t_{\mathsf{SA}})$ time. Then the SST finds all the $[ssp_t, sep_t]$ values in time $O(\Delta)$. Thus the time spent on the SST searches is $O(t_{\mathsf{s}}(p) + \Delta\, t_{\mathsf{SA}})$.

[4] Using perfect hashing to move in constant time towards the children.

Second, we search the SPT for reverse pattern prefixes of lengths 1 to Δ, and thus they can all be searched for in time $O(\Delta^2)$. Since the SPT is a blind trie, it might be either that the intervals $[ssp'_t, sep'_t]$ it returns are the correct interval of $P[1..t]$, or that $P[1..t]$ does not terminate any sparse prefix. A simple way to determine which is the case is to perform the orthogonal range search as explained, asking for the successor e_0 of position 1, and check whether the resulting position, $e_0 - t$, is an occurrence of P, that is, whether $\mathsf{SA}^{-1}[e_0 - t] \in [sp, ep]$. This takes $O(t_{\mathsf{SA}} + \log^{1+\epsilon} n)$ time per verification. Considering the searches plus verifications, the time spent on the SPT searches is $O(\Delta(\Delta + t_{\mathsf{SA}} + \log^{1+\epsilon} n))$.

Finally, after determining all the intervals $[ssp_t, sep_t]$ and $[ssp'_t, sep'_t]$, we perform $O(|\mathsf{Semi}(P,k)|\Delta)$ orthogonal range searches for positions q, in time $O(|\mathsf{Semi}(P,k)|\Delta \log^{1+\epsilon} n)$, and keep the closest one for each q.

Lemma 8. *Given a semi-candidate set* $\mathsf{Semi}(P,k)$*, where* $p > \Delta$*, a candidate set* $\mathsf{Cand}(P,k)$ *of size* $O(|\mathsf{Semi}(P,k)|)$ *can be computed in time* $O(t_{\mathsf{s}}(p) + \Delta(\Delta + t_{\mathsf{SA}} + |\mathsf{Semi}(P,k)| \log^{1+\epsilon} n))$ *using a data structure of* $O((n/\Delta)\log^2 n)$ *bits.*

Thus, by combining Lemma 6 using $\delta = 2\epsilon$ (so its space is $o(n/\log^{\epsilon} n)$ bits) and Lemma 8, we obtain Lemma 5.

6 Concluding Remarks

We have obtained the first succinct result for top-k term-proximity queries. The following additional results will be presented in the full version of this paper.

1. Another trade-off for top-k term-proximity queries with space and query time $2n\log\sigma + o(n\log\sigma) + O(n\log\log n)$ bits and $O(p + k\log k \log^{1+\epsilon} n)$, respectively. Notice that, when $\log\log n = o(\log\sigma)$, the trade-off matches with the best known result for top-k term-frequency queries [15].
2. In a more realistic scenario, $\mathsf{score}(\cdot,\cdot)$ is a weighted sum of PageRank, term-frequency and term-proximity with predefined non-negative weights [33]. Top-k queries with such ranking functions can be handled using an index of space $2n\log\sigma + o(n\log\sigma)$ bits in time $O(p + k\log k\log^{4+\epsilon} n)$.

References

1. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval, 2nd edn. Addison-Wesley (2011)
2. Belazzougui, D., Navarro, G.: Alphabet-independent compressed text indexing. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 748–759. Springer, Heidelberg (2011)
3. Belazzougui, D., Navarro, G., Valenzuela, D.: Improved compressed indexes for full-text document retrieval. J. Discr. Alg. **18**, 3–13 (2013)
4. Benson, G., Waterman, M.: A fast method for fast database search for all k-nucleotide repeats. Nucleic Acids Research **22**(22) (1994)

5. Broschart, A., Schenkel, R.: Index tuning for efficient proximity-enhanced query processing. In: Geva, S., Kamps, J., Trotman, A. (eds.) INEX 2009. LNCS, vol. 6203, pp. 213–217. Springer, Heidelberg (2010)
6. Büttcher, S., Clarke, C.L.A., Cormack, G.: Information Retrieval: Implementing and Evaluating Search Engines. MIT Press (2010)
7. Ferragina, P., Grossi, R.: The string B-tree: A new data structure for string search in external memory and its applications. J. ACM **46**(2), 236–280 (1999)
8. Ferragina, P., Manzini, G.: Indexing compressed text. J. ACM **52**(4), 552–581 (2005)
9. Ferragina, P., Manzini, G., Mäkinen, V., Navarro, G.: Compressed representations of sequences and full-text indexes. ACM Trans. Alg. **3**(2), art. 20 (2007)
10. Gagie, T., Navarro, G., Puglisi, S.J.: New algorithms on wavelet trees and applications to information retrieval. Theor. Comp. Sci. **426–427**, 25–41 (2012)
11. Grossi, R., Vitter, J.S.: Compressed suffix arrays and suffix trees with applications to text indexing and string matching. SIAM J. Comput. **35**(2), 378–407 (2005)
12. Gusfield, D.: Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press (1997)
13. Hon, W.-K., Shah, R., Thankachan, S.V., Vitter, J.S.: On position restricted substring searching in succinct space. J. Discr. Alg. **17**, 109–114 (2012)
14. Hon, W.-K., Shah, R., Thankachan, S.V., Vitter, J.S.: Faster compressed top-k document retrieval. In: Proc. 23rd DCC, pp. 341–350 (2013)
15. Hon, W.-K., Shah, R., Thankachan, S.V., Scott Vitter, J.: Space-efficient frameworks for top- k string retrieval. J. ACM **61**(2), 9 (2014)
16. Hon, W.-K., Shah, R., Vitter, J.S.: Space-efficient framework for top-k string retrieval problems. In: Proc. 50th FOCS, pp. 713–722 (2009)
17. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. SIAM J. Comp. **22**(5), 935–948 (1993)
18. Munro, J.I., Navarro, G., Shah, R., Thankachan, S.V.: Ranked document selection. In: Ravi, R., Gørtz, I.L. (eds.) SWAT 2014. LNCS, vol. 8503, pp. 344–356. Springer, Heidelberg (2014)
19. Muthukrishnan, S.; Efficient algorithms for document retrieval problems. In: Proc. 13th SODA, pp. 657–666 (2002)
20. Navarro, G.: Spaces, trees and colors: The algorithmic landscape of document retrieval on sequences. ACM Comp. Surv. **46**(4), art. 52 (2014)
21. Navarro, G., Mäkinen, V.: Compressed full-text indexes. ACM Comp. Surv. **39**(1), art. 2 (2007)
22. Navarro, G., Nekrich, Y.: Top-k document retrieval in optimal time and linear space. In: Proc. 23rd SODA, pp. 1066–1078 (2012)
23. Navarro, G., Russo, L.: Fast fully-compressed suffix trees. In: Proc. 24th DCC, pp. 283–291 (2014)
24. Navarro, G., Thankachan, S.V.: Faster top-k document retrieval in optimal space. In: Kurland, O., Lewenstein, M., Porat, E. (eds.) SPIRE 2013. LNCS, vol. 8214, pp. 255–262. Springer, Heidelberg (2013)
25. Navarro, G., Thankachan, S.V.: Top-k document retrieval in compact space and near-optimal time. In: Cai, L., Cheng, S.-W., Lam, T.-W. (eds.) Algorithms and Computation. LNCS, vol. 8283, pp. 394–404. Springer, Heidelberg (2013)
26. Navarro, G., Thankachan, S.V.: New space/time tradeoffs for top-k document retrieval on sequences. Theor. Comput. Sci. **542**, 83–97 (2014)
27. Nekrich, Y., Navarro, G.: Sorted range reporting. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 271–282. Springer, Heidelberg (2012)

28. Pătraşcu, M.: Succincter. In: Proc. 49th FOCS, pp. 305–313 (2008)
29. Schenkel, R., Broschart, A., Hwang, S., Theobald, M., Weikum, G.: Efficient text proximity search. In: Ziviani, N., Baeza-Yates, R. (eds.) SPIRE 2007. LNCS, vol. 4726, pp. 287–299. Springer, Heidelberg (2007)
30. Shah, R., Sheng, C., Thankachan, S.V., Vitter, J.S.: Top-k document retrieval in external memory. In: Bodlaender, H.L., Italiano, G.F. (eds.) ESA 2013. LNCS, vol. 8125, pp. 803–814. Springer, Heidelberg (2013)
31. Weiner, P.: Linear pattern matching algorithm. In: Proc. 14th Annual IEEE Symposium on Switching and Automata Theory, pp. 1–11 (1973)
32. Yan, H., Shi, S., Zhang, F., Suel, T., Wen, J.-R.: Efficient term proximity search with term-pair indexes. In: CIKM, pp. 1229–1238 (2010)
33. Zhu, M., Shi, S., Li, M., Wen, J.-R.: Effective top-k computation in retrieving structured documents with term-proximity support. In: CIKM, pp. 771–780 (2007)
34. Zhu, M., Shi, S., Yu, N., Wen, J.-R.: Can phrase indexing help to process non-phrase queries? In: CIKM, pp. 679–688 (2008)