Efficient taxa identification using a pangenome index

Omar Ahmed[1] Massimiliano Rossi[2] Christina Boucher[2] Ben Langmead[1]

[1]Department of Computer Science,
Johns Hopkins University, Baltimore, MD, USA
{oahmed6, blangme2}@jhu.edu

[2]Department of Computer and Information Science and Engineering,
Herbert Wertheim College of Engineering,
University of Florida, Gainesville, FL, USA
{rossi.m, christinaboucher}@ufl.edu

Abstract

Tools that classify sequencing reads against a database of reference sequences require efficient index data structures. The r-index is a compressed full-text index that answers substring presence/absence, count and locate queries in space proportional to the amount of distinct sequence in the database: $\mathcal{O}(r)$ space where r is the number of Burrows-Wheeler runs. To date, the r-index has lacked the ability to quickly classify matches according to which reference sequences (or sequence groupings, i.e. taxa) a match overlaps. We present new algorithms and methods for solving this problem. Specifically, given a collection $\mathcal{D}$ of d documents $\mathcal{D} = \{T_1, T_2, \ldots, T_d\}$ over an alphabet of size σ, we extend the r-index with $\mathcal{O}(rd)$ additional words to support document listing queries for a pattern $S[1..m]$ that occurs in $ndoc$ documents in $\mathcal{D}$ in $\mathcal{O}(m \log \log_w(\sigma + n/r) + ndoc)$ time and $\mathcal{O}(rd)$ space, where w is the machine word size. Applied in a bacterial mock community experiment, our method is up to 3 times faster than a comparable method that uses the standard r-index locate queries. We show that our method classifies both simulated and real nanopore reads at the strain level with higher accuracy compared to other approaches. Finally, we present strategies for compacting this structure in applications where read lengths or match lengths can be bounded.

Introduction

Metagenomic read (Wood et al., 2019) classification allows researchers to study organisms present in an environmental sample. Tools like Kraken 2 (Wood et al., 2019) and Centrifuge (Kim et al., 2016) accomplish this using an index of the reference sequences. Kraken 2 (Wood et al., 2019) builds a compact hash table that maps minimizer sequences onto the taxonomic lowest-common ancestor of the genomes it occurs in. Centrifuge (Kim et al., 2016) uses an FM-index (Ferragina and Manzini, 2000) to find substring matches which are combined to make classification decisions. But as databases of reference sequences continue to grow, these tools encounter difficulties with scaling and accuracy. Nasko et al. (Nasko et al., 2018) showed that the specificity of k-mer based approaches like Kraken 2 can suffer as the reference database (i.e., RefSeq) grows, since the addition of new sequences causes more k-mers (or minimizers) to co-occur in distant parts of the taxonomy. The FM-index at the core of Centrifuge does not naturally scale to pangenomes; rather, it requires an initial work-intensive step that compresses the genomes in a way that elides some of the underlying genetic variation.

The r-index (Gagie et al., 2020) is a successor to the FM-index that indexes repetitive texts using $\mathcal{O}(r)$-space, where r is the number of runs in the text's Burrows-Wheeler Transform (BWT). Since r grows only with the amount of *distinct* sequence in the collection, the r-index scales naturally to large pangenomes and reference databases like the ones used for taxonomic classification. Since it is a full-text index, the r-index can find matches of any length, unconstrained by a particular choice of k-mer length.

While the r-index has already been applied to pangenomic pattern-matching (Kuhnle et al., 2020; Rossi et al., 2022) and binary classification (Ahmed et al., 2021), it has so far lacked the ability to solve multi-class classification problems in an accurate and efficient manner. A straightforward approach would be to use standard backward search in the r-index, then use locate queries to locate the offsets in the concatenated text where the pattern occurs. These offsets can then be cross-referenced with another structure to determine which documents they occur in. This requires an amount of work proportional to the number of occurrences occ, which is expensive, particularly for repetitive matches against a pangenome.

We hypothesized that extending the r-index to multi-class classification could be accomplished by augmenting it with efficient facilities for *document listing*, i.e. the ability to report all the reference sequences (documents) where a particular pattern occurs. A document—which we will sometimes call a "class"—could consist of a single genome or a collection of related genomes.

An early study by Muthukrishnan (Muthukrishnan, 2002) described a specialized index for document listing consisting of a generalized suffix tree and a document array. It provided $\mathcal{O}(m + ndoc)$ queries, where m is the length of the pattern and $ndoc$ is the number of distinct documents it occurs in. But this came at the cost of $\mathcal{O}(n \log n)$ bits of space, where n is the total length of the texts, which is impractical for large pangenome databases. Sadakane (Sadakane, 2007) improved on this by introducing a new succinct document array representation and building on succinct representations of suffix trees and arrays. He showed how to reduce the index size to $|CSA| + \mathcal{O}(n)$ bits, where $|CSA|$ is the size of the compressed suffix array using statistical compression with an increased time complexity of $\mathcal{O}(m + ndoc \cdot \log n)$, a high cost for repetitive text collections (Cobas and Navarro, 2019). Later efforts further reduced the required space using grammar-compression (Cobas and Navarro, 2019) and relative Lempel-Ziv compression (Puglisi and Zhukova, 2021).

We present a new method that solves the document listing problem in $\mathcal{O}(m \log \log_w \sigma + ndoc)$-time and $\mathcal{O}(rd)$-space using the r-index. Importantly, we also show how to use the prefix-free parsing process to build the profile simultaneously with the BWT. This document-array structure can be sampled and stored at the run boundaries of the BWT, yielding a space complexity of $\mathcal{O}(rd)$. At query time, after performing backward search for a pattern, we can report the document listing by simply examining the current document array profile which is an array of d integers—as opposed to performing a query for

each occurrence of a pattern. We also discuss practical optimizations that can be used to reduce the space usage of this data structure even further in the context of metagenomic read classification. In our evaluations, we compare the query time and index size for our approach to an alternative that uses the standard r-index locate query to report document listings. Furthermore, we attempt to classify different strains of *Escherichia coli* and *Salmonella enterica* using our document array profiles in comparison into using SPUMONI 2's sampled document array (Ahmed et al., 2022). Finally, we believe that our theoretical guarantees will prove useful for the community by allowing read classification to be compared in a grounded manner that complements practical evaluation.

Results

We performed all the experiments on an Intel Xeon Gold 6248R 32-core processor running at 3.00GHz with 1.59 TB of RAM with 64-bit Linux. Time was measured using `std::chrono::system_clock` from the C++ standard library. Our source code can be found at `https://github.com/oma219/docprofiles`, and our experimental code can be found at `https://github.com/oma219/docprof-experiments`. The r-index code used in our experiments can be found at `https://github.com/maxrossi91/r-index`.

Comparing the query time and index size

To assess the speed of document listing, we compared the query time for the document array profiles to the query time for locate queries using the r-index. We attempted to compare our solution to the method of Cobas et al. (Cobas and Navarro, 2019), however, we ran into various run-time errors when using it as described, so we were not able to include it in the results.

We built a series of indexes over genomes from different collections of bacterial species, described in Table 2. We simulated nanopore sequencing reads using PBSIM2 (Ono et al., 2021) at 95% read accuracy. We then used MONI (Rossi et al., 2022) to query each read against the pangenome index, extracting a total of 1 million maximal exact matches (MEMs) for each class.

We tested two variants of the document array profile data-structure. The first (labeled "Doc. Array" in Figure 1) uses the standard document array profile, where the width of each profile entry requires $\lceil \log_2(|S|) \rceil$ bits. The second (labeled "Doc. Array (optimized)") instead stores truncated lcp values, so that lcps greater than 255 are stored as 255, so that only $\lceil \log_2(255) \rceil = 8$ bits are required per entry. This optimization is appropriate in real-world situations where either the reads are known to be short (e.g. Illumina sequencing reads), or where we would otherwise expect MEMs longer than 255 to be rare.

We observed that the query time using document array profiles was faster than the r-index locate query. For the 3-class database, the document array profiles ranged from 1.6–3.2 times faster. As more genomes were added to the database, query time for the 3-class r-index increased by 2.2-fold (214.87 sec vs 96.2 sec), while query time for the document array profile was essentially constant (67.8 sec vs 61.7 sec). This exhibits a key advantage of our document-listing; unlike when using the r-index locate queries, our query time is independent of the number of pattern occurrences.

We noted that the size of the r-index stayed relatively constant as the number of classes increased. However, for the document array profile (both standard and optimized), the index size grew with the number of classes, consistent with its $\mathcal{O}(rd)$ space complexity. As an example in the "30+" genome database, focusing on the standard document array, the 8-class document array was 2.32 times larger than the 5-class document array. Since d increased by 1.67 times, and r increased by 1.43 times (79,722,710 vs 55,559,459), therefore, we would expect to see an index increase of about 2.39 times (1.67 x 1.43), which is close to what we see in practice (2.32).

We also observed for the 3-class, "30+" genome database, the optimized document array was smaller than the r-index. The r-index stores a run-length encoded BWT (RLEBWT) along with the suffix array

i	BWT[i]_T	BWM_T	SA[i]_T	LF(i)	DA[i]_T	LCP[i]_T	$P_{DA}[i]_T$
1	C	**#**ATATGGC$GTAGAAT$TATGAAC	24	12	3	0	[0, 0, 1]
2	C	**$GTAGAAT$TATGAAC#**ATATGGC	8	13	1	0	[17, 1, 0]
3	T	**$TATGAAC#**ATATGGC$GTAGAAT	16	19	2	1	[1, 9, 0]
4	G	**AAC#**ATATGGC$GTAGAAT$TATG	21	14	3	0	[1, 2, 4]
5	G	**AAT$TATGAAC#**ATATGGC$GTAG	13	15	2	2	[1, 12, 2]
6	A	**AC#**ATATGGC$GTAGAAT$TATGA	22	4	3	1	[1, 1, 3]
7	T	**AGAAT$TATGAAC#**ATATGGC$GT	11	20	2	1	[1, 14, 1]
8	A	**AT$TATGAAC#**ATATGGC$GTAGA	14	5	2	1	[2, 11, 2]
9	#	**ATATGGC$GTAGAAT$TATGAAC#**	1	1	1	2	[24, 2, 2]
10	T	**ATGAAC#**ATATGGC$GTAGAAT$T	18	21	3	2	[3, 2, 7]
11	T	**ATGGC$GTAGAAT$TATGAAC#**AT	3	22	1	3	[22, 2, 3]
12	A	**C#**ATATGGC$GTAGAAT$TATGAA	23	6	3	0	[1, 0, 2]
13	G	**C$GTAGAAT$TATGAAC#**ATATGG	7	16	1	1	[18, 0, 1]
14	T	**GAAC#**ATATGGC$GTAGAAT$TAT	20	23	3	0	[1, 3, 5]
15	A	**GAAT$TATGAAC#**ATATGGC$GTA	12	7	2	3	[1, 13, 3]
16	G	**GC$GTAGAAT$TATGAAC#**ATATG	6	17	1	1	[19, 1, 1]
17	T	**GGC$GTAGAAT$TATGAAC#**ATAT	5	24	1	1	[20, 1, 1]
18	$	**GTAGAAT$TATGAAC#**ATATGGC$	9	2	2	1	[1, 16, 1]
19	A	**T$TATGAAC#**ATATGGC$GTAGAA	15	8	2	0	[1, 10, 1]
20	G	**TAGAAT$TATGAAC#**ATATGGC$G	10	18	2	1	[2, 15, 2]
21	$	**TATGAAC#**ATATGGC$GTAGAAT$	17	3	3	2	[4, 2, 8]
22	A	**TATGGC$GTAGAAT$TATGAAC#**A	2	9	1	4	[23, 2, 4]
23	A	**TGAAC#**ATATGGC$GTAGAAT$TA	19	10	3	1	[2, 1, 6]
24	A	**TGGC$GTAGAAT$TATGAAC#**ATA	4	11	1	2	[21, 1, 2]

(A) Example of document array profiles (P_{DA}) for the text $\mathcal{T}$ which is the concatenation of the following three documents: {ATATGGC$, GTAGAAT$, TATGAAC#}

Character:	Start:	End:	Current Profile:	Notes:
G	1	25	[1, 3, 5]	Sample profile from $P_{DA}[\text{LF}(i)]$ where BWT[i] is a run boundary with a G (i=4, LF(4)=14)
T	14	19	[2, 1, 6]	Sample profile from $P_{DA}[\text{LF}(i)]$ where BWT[i] is a run boundary with a T (i=14, LF(14)=23)
A	23	25	[3, 2, 7]	Increment profile by 1
T	10	12	[4, 3, 8]	Increment profile by 1

(B) Querying the document array profiles (P_{DA}) for the pattern P = TATG by using backward search. The table shows the document array profile at each step of the search, and $[4, 3, 8]$ is the final profile, which means that the pattern P occurs in document 1 and 3 since $|P| \leq 4$ and $|P| \leq 8$.

Table 1: (A) Shows an example of document array profiles for three documents and (B) shows the results of querying a small pattern to determine its document listing.

Species Used in Each Dataset			
Dataset:	3-class	5-class	8-class
Species:	*Echerichia coli* *Salmonella enterica* *Bacillus subtilis*	*Echerichia coli* *Salmonella enterica* *Bacillus subtilis* *Listeria monocytogenes* *Pseudomonas aeruginosa*	*Echerichia coli* *Salmonella enterica* *Bacillus subtilis* *Listeria monocytogenes* *Pseudomonas aeruginosa* *Lactobacillus fermentum* *Enterococcus faecalis* *Staphylococcus aureus*

Table 2: Species included the datasets of Figure 1. For each target database size (30, 100, 300 genomes), we include an equal number of genomes from each class. For example, for the 3-class dataset, we include 10 genomes of each species in the 30-genome database, 34 genomes of each in the 100-genome database and 100 genomes of each species in the 300-genome database. Note that this leads to collections that slightly exceed the target size, e.g. 3×34 leads to an index of 102 genomes. For this reason, we refer to the database sizes as "30+", "100+" and "300+".

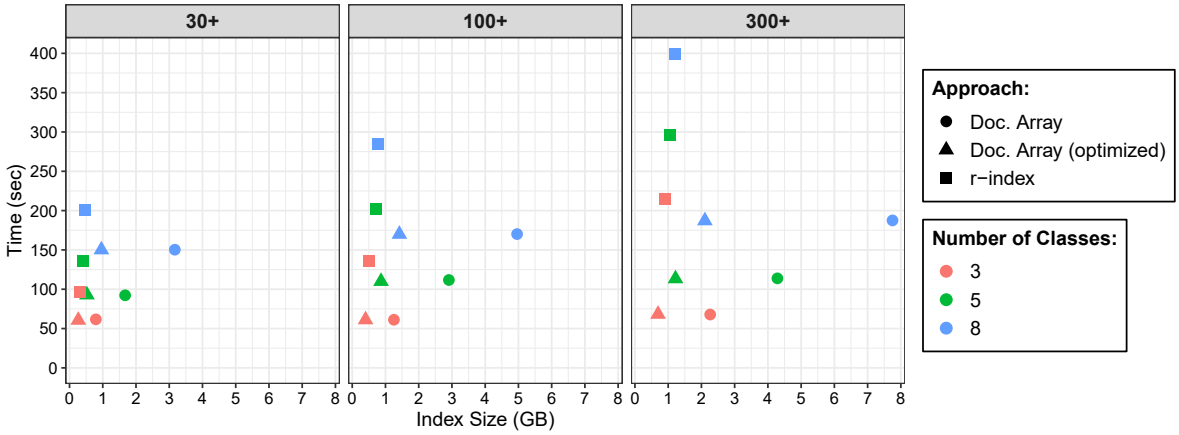
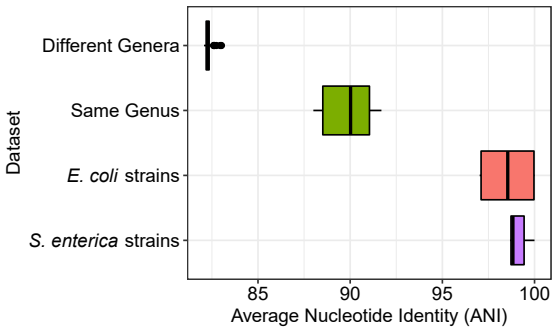


Figure 1: Query time and index size when performing document listing queries using the document array profiles and the r-index. We varied the size of the database increasing from 30 bacterial genomes to 300 bacterial genomes. For each species/class, we would simulate nanopore reads at 95% accuracy and extract 1 million maximal-exact matches (MEMs) to query the data-structures. Therefore, for the 3-class, 5-class, and 8-class indexes, we queried them with 3 million, 5 million and 8 millions MEMs respectively. This explains why the query time would increase for the indexes containing more classes.

sampled at run boundaries in the BWT where each sample is stored in 5 bytes. The optimized document array also stores a RLEBWT however instead of the suffix array, it replaces it with the document array profiles. Since it is a 3-class database, each profile sampled at the run boundaries will only consist of 3 bytes which explains why overall the optimized document array is smaller than the r-index for those conditions.

Additionally, as expected, the optimized document array profile was smaller than the standard profile; for the 300 genome database, it was 3.3 times smaller. We suggest further optimizations to reduce the



(A) Sequence similarity in each dataset

Dataset:	Classes (1, 2, 3, 4):
Different Genera	*Escherichia coli, Salmonella enterica, Listeria monocytogenes, & Pseudomonas aeruginosa*
Same Genus	*Escherichia coli, Escherichia albertii, Escherichia fergusonii, & Escherichia marmotae*
E. coli strains	GCF_025426235.1, GCF_025563475.1, GCF_025563515.1, GCF_025563435.1
S. enterica strains	GCF_025398995.1, GCF_025399055.1, GCF_025399075.1, GCF_025399015.1

(B) Classes in each dataset

Figure 2: (A) Sequence homology, measured as Average Nucleotide Identity (ANI) for all across-class pairs of sequences. ANI was estimated with fastANI (Jain et al., 2018). (B) Lists the specific species and strains used for classes 1, 2, 3 and 4 for each of the four datasets. In the case of "Different Genera" and "Same Genus," we used 10 genomes per class. In the case of "*E. coli* strains" and "*S. enterica* strains" we used a single genome for each strain.

document array profile size in the Discussion below.

Species and strain-level classification

We hypothesized that the document array profiles could particularly improve read classification accuracy in difficult scenarios where it is important to be able to list all documents for each MEM. We compared the performance of the document array profile to another tool and structure designed for read classification: SPUMONI 2's (v2.0.0) (Ahmed et al., 2022) sampled document array. SPUMONI 2's sampled document array is quite simple; for each BWT run boundary, it simply converts the suffix array position to the document number that position occurs in. Using these document labels, it is capable of reporting one document that a particular exact match occurs in. This is sufficient in situations where reads contain many distinct matches (e.g. MEMs), so that document information can be pooled across the various matches to come to an overall conclusion. But in situations where the documents are very similar to each other, or where reads are short or have a high error rate, we expect the full document array profile to impart higher accuracy.

We tested the two structures on increasingly difficult datasets, with each dataset consisting of reference genomes from four distinct classes (Figure 2). We used used PBSIM2 (Ono et al., 2021) to simulate 50,000 nanopore reads from each class at 95% accuracy, then classified the reads using both document array approaches. Specifically, we identified all MEMs between the reads and the pangenome index, filtering to just MEMs of length 15 or longer. We then used the different document structures to obtain matching documents for each MEM; in the case of SPUMONI 2, we retrieved one document per MEM; in the case of our document array profile, we retrieved all documents where the MEM occurred. We then weighted the documents according to the length of the MEM and assigned each read to a document according to which received the largest total weight across all reported MEM/document combinations.

We observed that when the dataset consisted of classes with low between-class sequence similarity ("Different Genera" and "Same Genus"), both methods performed well, with low classification error (Figure 3). However for datasets with high sequence similarity (>97.5% ANI), such as the "*E. coli* strains" and "*S. enterica* strains," we see that the full document array profile provided greater classification accuracy compared to SPUMONI 2's one-document-per-match approach.

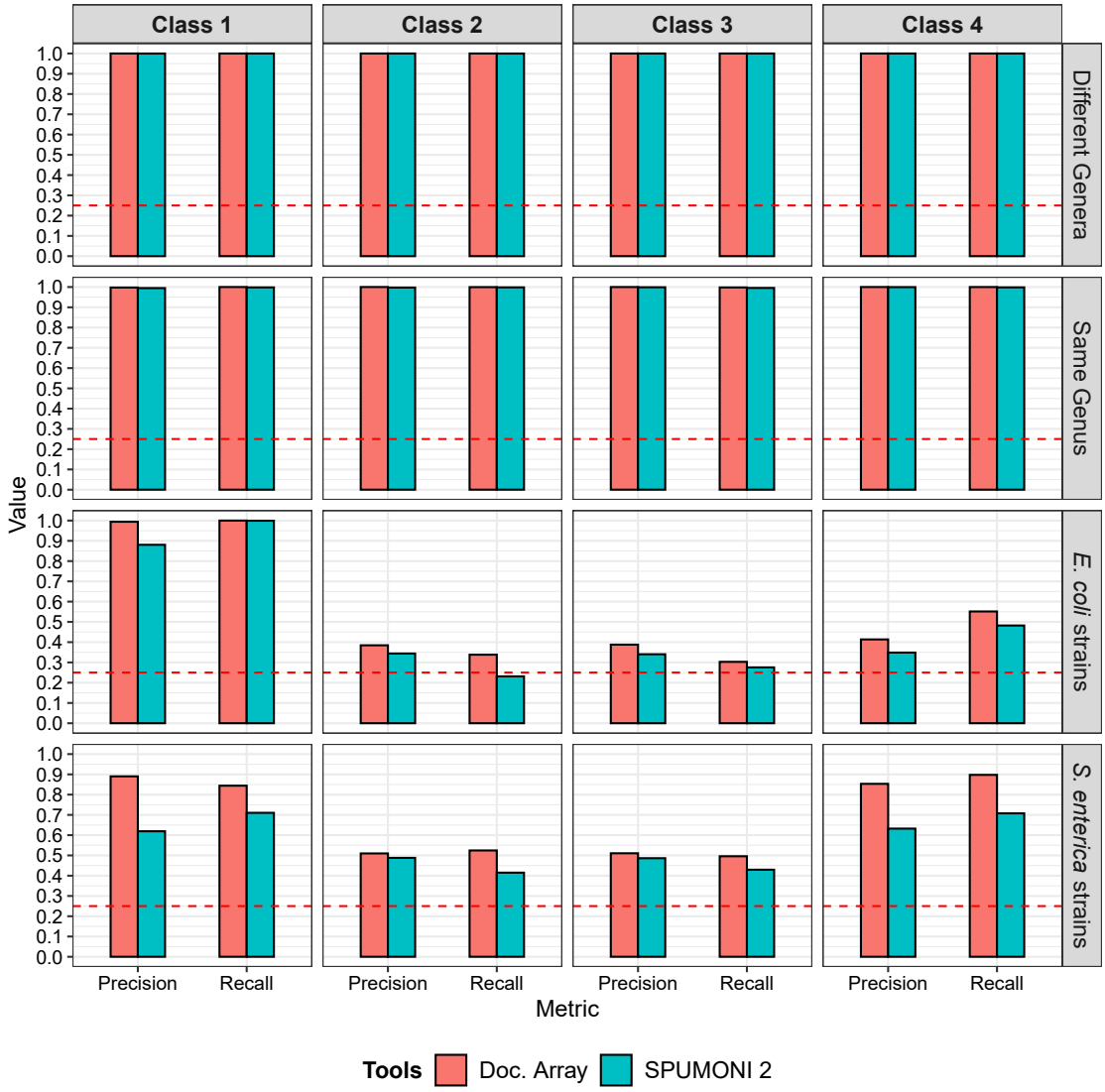


Figure 3: Classification results using the document array profiles and SPUMONI 2's (Ahmed et al., 2022) sampled document array across four different datasets each with four classes described in Figure 2

Classification using real nanopore mock community reads

We extended our analysis to real sequencing reads. We used nanopore reads from the UNCALLED (Kovaka et al., 2021) paper, which performed Oxford Nanopore sequencing of a Zymo Mock Community consisting of eight species [1]. We extracted a set of 582,042 reads from the dataset that uniquely mapped to one of the seven bacterial species using minimap2 (Li, 2018). We shortened each read to 2000 bp.

For each bacterial species, we constructed a database comprising of four strains from that species, one of which was chosen to be the actual strain used for the Zymo Mock community. The other three strains were obtained from Refseq. We then compared the strain-level classification accuracy of the two document-array structures using the same MEM-weighted approach as was used in the previous

[1] *Staphylococcus aureus, Salmonella enterica, Escherichia coli, Pseudomonas aeruginosa, Listeria monocytogenes, Enterococcus faecalis, Bacillus subtilis, and Saccharomyces cerevisiae*

experiment. As in the previous experiment, we observed that the document array profile enabled more accurate strain-level read classification (Figure 4). This was true for reads derived from all 7 of the bacterial species (though both approaches had near-perfect recall for B. subtilis reads).

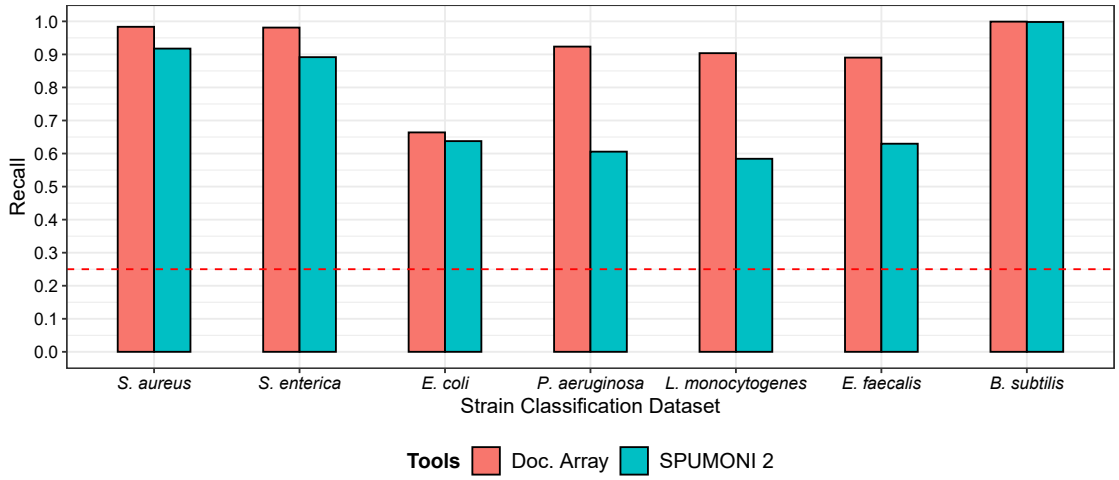


Figure 4: Comparing the document array profiles and SPUMONI 2's(Ahmed et al., 2022) sampled document array on seven different strain-level classification tasks using real nanopore reads from UNCALLED (Kovaka et al., 2021) project.

Discussion

We described a new data structure called the document array profile, along with an efficient algorithm for building the structure simultaneously with a pangenome r-index. This structure enables tools to simultaneously find exact matches with respect to a full-text pangeonme index, while simultaneously learning which reference sequences the matches belong to. This opens the door to new applications of pangenome indexes, including in metagenomics read classification.

The structure requires $\mathcal{O}(rd)$ space and can compute a full document listing for a match in $\mathcal{O}(m \log \log_w \sigma + ndoc)$ time. We showed that, as the pangenome database grows in size, the document array profile's speed advantage grows relative to the standard r-index and its locate queries. Further, we showed that the structure's ability to list all documents associated with a match enables greater accuracy compared to an existing alternative that considers only one document per match.

The main weakness of the document array profile is the fact that its space usage grows linearly with the number of documents d. This makes it difficult for it to be used in scenarios with a large number of documents (classes) which is the case in taxonomic read classification where there are thousands of species. However, this data-structure can optimized even further to reduce its space usage with domain-specific knowledge. For example in sequencing read classification, an exact match shorter than 15 bases might be too non-specific to be helpful for classification. In that case, each element of the document array profile could be made "sparse," consisting only of values greater than 14.

An additional optimization would be to adopt a "top k" strategy. That is, rather than store lcp values to all possible documents, we can restrict the structure to store only the lcp values to the k documents having the greatest lcp at the run boundary. This allows us to bound the size of the structure while retaining the strongest match-to-document associations.

Recently, Cobas et al. (Cobas et al., 2020) designed solutions to the *document listing with frequencies*

problem using the r-index as the text index. This problem is a more difficult task since it requires reporting not only the document listing, but also the frequency of the pattern in each document. The frequency information could add valuable data for taxonomic classification since it gives an indication if a pattern is "common" within a document, or if it is rather rare. Future work on the document array profiles will consist of exploring the possibility integrating elements of Cobas et al. solution (Cobas et al., 2020) to allow the document array profiles to report frequencies along with the document listing.

Methods

Preliminaries

A string $S[1..n]$ of length $|S| = n$ is the concatenation of characters $S[1] \cdots S[n]$ drawn from an alphabet Σ of size σ. We denote by ε the *empty* string that is the only string of length 0. We assume S is terminated by a special symbol $\$ \notin \Sigma$ lexicographically smaller than all symbols in Σ. Given two integers $1 \leq i, j \leq n$, we denote with $S[i..j] = S[i] \cdots S[j]$ the *substring* of S spanning positions i through j if $i \leq j$, and $S[i..j] = \varepsilon$ otherwise. Given two integers $1 \leq i, j \leq n$, we refer to $S[i..n]$ as the i-th *suffix* of S and to $S[1..j]$ as the j-th *prefix* of S. Given two strings $S[1..n]$ and $T[1..m]$, we denote with $\mathsf{lcp}(S, T)$ the length of the *longest common prefix* of S and T.

Suffix array, inverse suffix array, and longest common prefix array. Given a string S, the *suffix array* (Manber and Myers, 1993) $\mathsf{SA}_S[1..n]$ is the permutation of $\{1, \ldots, n\}$ that lexicographically sorts the suffixes of S. The *inverse suffix array* $\mathsf{ISA}_S[1..n]$ is the inverse permutation of $\mathsf{SA}_S[1..n]$, i.e., for all $i = 1, \ldots, n$ $\mathsf{SA}_S[\mathsf{ISA}_S[i]] = i$. The *longest common prefix* array $\mathsf{LCP}_S[1..n]$ stores the length of the longest common prefix between lexicographically consecutive suffixes of S, formally, $\mathsf{LCP}[1] = 0$ and for all $i = 2, \ldots, n$ $\mathsf{LCP}[i] = \mathsf{lcp}(S[\mathsf{SA}_S[i-1]..n], S[\mathsf{SA}_S[i]..n])$.

Burrows-Wheeler transform. Given a string S, the *Burrows-Wheeler transform* (Burrows and Wheeler, 1994) $\mathsf{BWT}_S[1..n]$ is the reversible permutation of S defined as the last column of the matrix of the lexicographically sorted rotations of S. When S is terminated by $\$$ we can define for all $i = 1, \ldots, n$, $\mathsf{BWT}_S[i] = S[\mathsf{SA}_S[i]-1]$ where $S[0] = S[n]$. The LF-*mapping* is the the permutation of $\{1, \ldots, n\}$ that maps every character in the BWT_S to its predecessor in text order, formally $\mathsf{LF}[i] = \mathsf{ISA}_S[(\mathsf{SA}_S[i] - 2 \mod n) + 1]$. We define r as the number of maximal equal-letter runs of BWT_S. When clear from the context we refer to SA_S, ISA_S, LCP_S, and BWT_S as SA, ISA, LCP, and BWT respectively.

r-index. The r-index (Gagie et al., 2020) is a text index that stores the run-length encoded BWT and the SA entries sampled at run boundaries. Given a text $S[1..n]$ and a pattern $P[1..m]$ the r-index allows you to find all occurrences of P in S in $\mathcal{O}(m \log \log_w(\sigma + n/r) + occs \log \log_w(n/r))$ and $\mathcal{O}(r)$ words of space, where $occs$ is the number of occurrences of P in S. This result was later improved to $\mathcal{O}(m \log \log_w(\sigma) + occs)$ (Nishimoto et al., 2022).

Document array. We denote with $\mathcal{D} = \{T_1, \ldots, T_d\}$ the *collection* of documents (strings) $T_1, \ldots, T_d$, and we denote with $\mathcal{T}[1..n] = T_1 \cdots T_d$ the concatenation of the documents. The *document array* (Muthukrishnan, 2002) $\mathsf{DA}[1..n]$ stores for each position i the document index of $\mathcal{T}[\mathsf{SA}_{\mathcal{T}}[i]..n]$. An important problem in document retrieval is the *document listing* problem.

Problem 1. *Given a collection $\mathcal{D} = \{T_1, \ldots, T_d\}$ and a pattern P, return the set of documents $\mathcal{L} \subseteq \mathcal{D}$ where P occurs.*

Supporting document listing on the r-index

Given the text $\mathcal{T}$ that is the concatenation of the documents of $\mathcal{D}$ such that $\mathcal{T}$ has length n, let BWT be the Burrows-Wheeler transform of $\mathcal{T}$ that has r equal-letter runs.

Definition 1. *For all positions $1 \leq i \leq n$ in the* BWT *of $\mathcal{T}$ we define the* profile of the document array *as the array $P_{\mathsf{DA}}[i][1..d]$ that stores for each position $j = 1, \ldots, d$ the length of the longest common prefix between $\mathcal{T}[\mathsf{SA}[i]..n]$ and all suffixes of document T_j. Formally,*

$$P_{\mathsf{DA}}[i][j] = \max\left\{\mathsf{lcp}(\mathcal{T}[\mathsf{SA}[i]..n], \mathcal{T}[\mathsf{SA}[k]..n]) \mid 1 \leq k \leq n \text{ and } \mathsf{DA}[k] = j\right\}$$

Lemma 1. *Given the* BWT *of $\mathcal{T}$ and the profile of the document array P_{DA}, for all pairs (i, ℓ) of positions and lengths corresponding to a substring $S = \mathcal{T}[\mathsf{SA}[i]..\mathsf{SA}[i] + \ell - 1]$, we can find the list of $ndoc$ documents where S occurs in $\mathcal{T}$ in $\mathcal{O}(d)$ time.*

Proof. By definition of $P_{\mathsf{DA}}[i]$ we have that S occurs in document T_j if and only if $\ell \leq P_{\mathsf{DA}}[i][j]$. Hence, we can scan the profile of the document array in position i. For all documents $j = 1, \ldots, d$ we check if the length of the substring is less than or equal to the value stored in the profile for the j-th document. This requires one comparison per document, or $\mathcal{O}(d)$ time. □

Example 1. *In the example in Table 1(A), if we look at $P_{DA}[4] = [1, 2, 4]$ corresponding to the suffix* AAC#*, we have that i) for the pair $(21, 1)$ the substring $S =$* A *occurs in documents* 1,2*, and* 3*, since all values of $P_{DA}[4]$ are not smaller than* 1*; ii) for the pair $(21, 2)$ the substring $S =$* AA *occurs in documents* 2*, and* 3 *since $2 > P_{DA}[4][1]$; iii) for the pair $(21, 3)$ the substring $S =$* AAC *occurs only in document* 3 *since* 3 *is greater than both $P_{DA}[4][1]$ and $P_{DA}[4][2]$.*

If we store each entry of the *profile of the document array* $P_{\mathsf{DA}}[i]$ as a list of sorted pairs $(P_{\mathsf{DA}}[i][j], j)$, the query time can be reduced to $\mathcal{O}(ndoc)$ by simply scanning the list of pairs from the document with the largest profile value, to the first document that has a profile value smaller than ℓ.

Sampling the profile of the document array.

Storing the entire profile of the document array requires $\mathcal{O}(nd)$ words of space, which will be excessive for pangenomes. We seek to compress the profile of the document array by sampling it similarly to how r-index samples the suffix array.

Let $\mathsf{BWT}[s..e]$ be a maximal equal-letter run of the BWT of $\mathcal{T}$. We store in position s and e the entries of the profile of the document array in positions $\mathsf{LF}(s)$ and $\mathsf{LF}(e)$ respectively. Applying the same reasoning as the toehold lemma (Policriti and Prezza, 2016) we can show that this is enough to recover the document listing for a query pattern S.

The first property of the profile of the document array that we show is an upper bound on the values of the profile, when performing an LF step.

Lemma 2. *For all positions $1 \leq i \leq n$ in the* BWT *of $\mathcal{T}$ such that $\mathsf{DA}[i] = \mathsf{DA}[\mathsf{LF}(i)]$, for all $j = 1, \ldots, d$ it holds that $P_{\mathsf{DA}}[\mathsf{LF}(i)][j] \leq P_{\mathsf{DA}}[i][j] + 1$.*

Proof. From the definition of $P_{\mathsf{DA}}[i][j]$ there exists a position $1 \leq k \leq n$ such that

$$\mathsf{lcp}(\mathcal{T}[\mathsf{SA}[i]..n], \mathcal{T}[\mathsf{SA}[k]..n]) \geq \max\left\{\mathsf{lcp}(\mathcal{T}[\mathsf{SA}[i]..n], \mathcal{T}[\mathsf{SA}[k']..n]) \mid 1 \leq k' \leq n \text{ and } \mathsf{DA}[k'] = j\right\}.$$

Hence, if we consider the character preceding $\mathsf{SA}[i]$, i.e. $\mathsf{BWT}[i]$, then by maximality of k we have that $\max\left\{\mathsf{lcp}(\mathcal{T}[\mathsf{SA}[i]-1..n], \mathcal{T}[\mathsf{SA}[k']..n]) \mid 1 \leq k' \leq n \text{ and } \mathsf{DA}[k'] = j\right\} \leq \mathsf{lcp}(\mathcal{T}[\mathsf{SA}[i]..n], \mathcal{T}[\mathsf{SA}[k]..n]) + 1$, concluding the proof. □

Example 2. *In the example in Table 1(A), if we look at* $P_{DA}[4] = [1, 2, 4]$ *and at* $P_{DA}[LF(4)] = P_{DA}[14] = [1, 3, 5]$*, we have that Lemma 2 is verified.*

We now show which elements of the profile of the document array can be extended when performing an LF mapping from a position in a maximal equal-letter run. Those are all the profiles corresponding to occurrences that are all preceded by the same character, i.e., the corresponding interval in the suffix array is contained in the maximal equal-letter run. We first recall that given a maximal equal-letter run $\mathsf{BWT}_{\mathcal{T}}[s..e]$, the length ℓ of the smallest substring S of $\mathcal{T}$ such that all occurrences of S in $\mathcal{T}$ are in $\mathsf{SA}_{\mathcal{T}}[s..e]$ is given by $\ell = \max(\mathsf{LCP}_{\mathcal{T}}[s], \mathsf{LCP}_{\mathcal{T}}[e+1]) + 1$, assuming $\mathsf{LCP}_{\mathcal{T}}[n+1] = 0$. Note that $\mathsf{SA}_{\mathcal{T}}[s..e]$ can also contain occurrences of substrings different from S.

Lemma 3. *Given a position* i *in the* BWT *of* $\mathcal{T}$ *such that* $\mathsf{DA}[i] = \mathsf{DA}[\mathsf{LF}(i)]$*, let* $\mathsf{BWT}[s..e]$ *be the maximal equal-letter run such that* $s \leq i \leq e$*, let* ℓ *be the length of the smallest substring* S *of* $\mathcal{T}$ *such that all occurrences of* S *in* $\mathcal{T}$ *are in* $\mathsf{SA}_{\mathcal{T}}[s..e]$*. Then for all* $j = 1, \ldots, d$ *such that* $P_{\mathsf{DA}}[i][j] \geq \ell$ *it holds that* $P_{\mathsf{DA}}[\mathsf{LF}(i)][j] = P_{\mathsf{DA}}[i][j] + 1$*.*

Proof. The first observation is that if $P_{\mathsf{DA}}[i][j] \geq \ell$, then there exists a $s \leq k \leq e$ such that $\mathsf{DA}[k] = j$ otherwise, by definition of ℓ and $P_{\mathsf{DA}}[i][j]$, $P_{\mathsf{DA}}[i][j] < \ell$. Hence, $\mathcal{T}[\mathsf{SA}[k]..n]$ is preceded by the same character as $\mathcal{T}[\mathsf{SA}[i]..n]$ because i and k are in the same BWT run. Therefore, if $\mathsf{DA}[k] = \mathsf{DA}[\mathsf{LF}[k]]$ we have that $\mathsf{lcp}(\mathcal{T}[\mathsf{SA}[\mathsf{LF}(k)]..n], \mathcal{T}[\mathsf{SA}[\mathsf{LF}(i)]..n]) = \mathsf{lcp}(\mathcal{T}[\mathsf{SA}[k]..n], \mathcal{T}[\mathsf{SA}[i]..n]) + 1$, which concludes the proof. □

Example 3. *In the example in Table 1(A), if we consider* $i = 4$*, we have that the maximal equal-letter run containing* i *is* $\mathsf{BWT}[4..5]$ *hence the the smallest substring* S *of* $\mathcal{T}$ *such that all occurrences of* S *in* $\mathcal{T}$ *are in* $\mathsf{SA}_{\mathcal{T}}[4..5]$ *is* AA *and its length is given by* $\ell = \max(\mathsf{LCP}[4], \mathsf{LCP}[6]) + 1 = \max(0, 1) + 1 = 2$*. Looking at* $P_{DA}[4] = [1, 2, 4]$ *and* $P_{DA}[LF(4)] = P_{DA}[14] = [1, 3, 5]$*, the only elements of* $P_{DA}[4]$ *that are not smaller than* 2 *are* $P_{DA}[4][2]$ *and* $P_{DA}[4][3]$ *and we have that* $P_{DA}[14][2] = P_{DA}[4][2] + 1$ *and* $P_{DA}[14][3] = P_{DA}[4][3] + 1$*, while* $P_{DA}[14][1] < P_{DA}[4][1] + 1$*.*

Note that the only case where we have that $\mathsf{DA}[i] \neq \mathsf{DA}[\mathsf{LF}(i)]$ is if the BWT runs is a run of \$s. Hence, the above lemma can be applied generally when performing pattern matching queries. We can summarize our solution to Problem 1 in the following theorem.

Theorem 1. *Given a collection* $\mathcal{D}$ *of* d *documents* $\mathcal{D} = \{T_1, T_2, \ldots, T_d\}$ *over an alphabet of size* σ*, we show how to extend the* r*-index with* $\mathcal{O}(rd)$ *additional words to support document listing queries for a pattern* $S[1..m]$ *that occurs in* $ndoc$ *documents in* $\mathcal{D}$ *in* $\mathcal{O}(m \log\log_w(\sigma + n/r) + ndoc)$ [2] *time and* $\mathcal{O}(rd)$ *space, where* w *is the machine word size.*

Proof. Given a collection $\mathcal{D}$, we store the BWT of the concatenation $\mathcal{T}$ of the documents of $\mathcal{D}$ and for all maximal equal-letter runs $\mathsf{BWT}[s..e]$ we store in the positions of s and e the SA samples $\mathsf{SA}[s]$ and $\mathsf{SA}[e]$, and the document array profile samples $P_{\mathsf{DA}}[\mathsf{LF}[s]]$ and $P_{\mathsf{DA}}[\mathsf{LF}[e]]$.

Let $S[1..m]$ be a pattern for which we want to compute the list of documents such that S occurs in $\mathcal{D}$. After we have processed $S[q..m]$ we have an interval $\mathsf{BWT}[s_q..e_q]$ containing all the occurrences of $S[q..m]$ in $\mathcal{T}$, and a profile P' such that for all documents j, $P'[j] \geq (m - q + 1)$ if $S[q..m]$ occurs in T_j, and $P'[j] < (m - q + 1)$ otherwise. Note that the profile is not required to be a document array profile entry for a given position.

If $q > 1$, we now want to extend the match of $S[q..m]$ to $S[q-1..m]$ and show how we can maintain the invariant of the profile P'. We consider two cases. The first case is if $\mathsf{BWT}[s_q..e_q]$ contains either the beginning or the end of a run of the character $S[q-1]$. Here, we can update

[2] query time can be improved to $\mathcal{O}(m \log\log_w \sigma + ndoc)$ by using the approach from Nishimoto et al. (2022)

the interval BWT$[s_{q-1}..e_{q-1}]$ with the standard backward-search and can select as P' the sample of the profile of the document array stored in the run boundary in BWT$[s_q..e_q]$. The invariant of P' is preserved by Lemma 1. The second case is when BWT$[s_q..e_q]$ is completely contained in a run, i.e., BWT$[s_q - 1]$ = BWT$[s_q]$ = ... = BWT$[e_q]$ = BWT$[e_q + 1]$, then we have that all occurrences of $S[q..m]$ are preceded by the same character, hence by Lemma 3 for all j such that $S[q..m]$ occurs in T_j, the profile of the document array P'' after the backward step is $P''[j] = P'[j] + 1 \geq (m - q)$. Furthermore, for all j such that $S[q..m]$ does not occur in T_j we have that $P'[j] < (m - q + 1)$, hence by Lemma 2 we have that $P''[j] \leq P'[j] + 1 < (m - q)$. Hence, if for all j we set $P''[j] = P'[j] + 1$, we have that the invariant requiring that for all documents j, $P''[j] \geq (m - q)$ if $S[q - 1..m]$ occurs in T_j, and $P''[j] < (m - q)$ otherwise is satisfied, concluding the proof. □

Computing the document array profiles

The computation of the document array profiles is performed by scanning the values of BWT, SA, LCP, and DA in a streaming fashion. For all position $i = 1, \ldots, n$ we base the computation of P_{DA}[LF(i)] on the observation that given a collection of documents $\mathcal{D}$ and a suffix u of document T_i. The suffix u of document T_j with the largest longest common prefix with u is the suffix of T_j that either immediately precedes or immediately follows the suffix u in SA order. Formally,

Proposition 2. *Given a collection of documents $\mathcal{D}$ let $\mathcal{T}$ be the concatenation of its documents. For all indexes $i = 1, \ldots, n$ let u_i be the suffix $\mathcal{T}$[SA$[i]..n]$ of document* DA$[i]$. *For all documents $k = 1, \ldots, d$ let v_k be a suffix $\mathcal{T}$[SA$[j]..n]$ of document* DA$[j] = k$ *with the largest longest common prefix with u_i. We assume w.l.o.g. that v_k is the only suffix with the largest longest common prefix with u_i. Then position j corresponding to v_k is either the position of the suffix preceding u_i that is a suffix of document k, i.e., $\max\{j <= i \mid \text{DA}[j] = k\}$, or the position of the suffix following u_i that is a suffix of document k, i.e., $\min\{j > i \mid \text{DA}[i] = k\}$.*

Therefore, we can divide the computation of P_{DA}[LF(i)] into two components, the computation of the longest common prefix of $\mathcal{T}$[SA$[i]..n]$ with the suffix of each document immediately preceding the suffix in position i, and the longest common prefix of $\mathcal{T}$[SA$[i]..n]$ with the suffix of each document immediately following the suffix in position i.

To compute the former, while scanning the values of BWT, SA, LCP, and DA from 1 to n, we maintain an auxiliary table Pred$[1..d][1..\sigma]$ such that at step i, for all documents $j = 1, \ldots, d$, and for all characters $c = 1, \ldots, \sigma$, Pred$[j][c]$ stores the length of the longest common prefix value between suffix $\mathcal{T}$[SA$[i]..n]$ and the immediately preceding suffix of document j that is preceded by character c. The values of Pred$[1..d][1..\sigma]$ can be iteratively computed from the values of Pred$[1..d][1..\sigma]$ at step $i - 1$ as Pred$[j][c] = \min($Pred$[j][c]$, LCP$[i])$, and we set Pred[DA[LF(i)]][BWT$[i]] = |\mathcal{T}$[SA$[i]..n]|$.

To compute the latter, the intuition is to simulate the maintenance of an auxiliary table equivalent to Pred but for following suffixes and apply an equivalent reasoning for Pred i.e., Succ$[1..d][1..\sigma]$ such that at step i, for all documents $j = 1, \ldots, d$, and for all characters $c = 1, \ldots, \sigma$, Succ$[j][c]$ stores the length of the longest common prefix value between suffix $\mathcal{T}$[SA$[i]..n]$ and the immediately following suffix of document j that is preceded by character c, if it exists. However, since we are scanning the values of BWT, SA, LCP, and DA from 1 to n the maintenance of such Succ table becomes more difficult. the intuition is that we will build the Succ table for position i, while evaluating the positions following i, and the table will be complete when when we have encountered at least one suffix of all documents $j = 1, \ldots, d$ that is preceded by the character BWT$[i]$. To achieve this at each step we maintain i) a queue LQ storing tuples of $(\text{pos}, \text{ch}, \text{doc}, \text{lcp})$; ii) a list of incomplete document array profiles that is the same size as LQ; iii) a table $LQC[1..d][1..\sigma]$ where for all documents $j = 1, \ldots, d$ and for all characters $c = 1, \ldots, \sigma$ $LQC[j][c]$ stores the number of tuples t in LQ such that $t.\text{doc} = j$ and $t.\text{ch} = c$.

At the step i we start by inserting the tuple $(i, \text{BWT}[i], \text{DA}[\text{LF}(i)], \text{LCP}[i])$ in the queue LQ, then we insert Pred[DA[LF[i]]][BWT[i]] in the list of incomplete document profiles, and and we increment the counter in LQC[DA[LF[i]]][BWT[i]] by one. Then, we update the incomplete document profiles by iterating through all tuples t in LQ starting from the last inserted element of the queue. While processing tuple t, let ℓ be the length of the longest common prefix between the suffix $\mathcal{T}[\text{SA}[i]..n]$ and the suffix $\mathcal{T}[\text{SA}[t.\text{pos}]..n]$, we update $P_{DA}[\text{LF}(t.\text{pos})][\text{DA}[\text{LF}(i)]]$ with $\max(P_{DA}[\text{LF}(t.\text{pos})][\text{DA}[\text{LF}(i)]], \ell)$ if $t.\text{ch} = \text{BWT}[i]$ and $t.\text{doc} \neq \text{DA}[\text{LF}(i)]$. Note that ℓ can be computed while scanning the tuples by initially setting $\ell = |\mathcal{T}[\text{SA}[i]..n]|$, and updating ℓ as ℓ as $\ell = \min(\ell, t.\text{lcp})$.

Finally, we scan the queue LQ from the first inserted element to check for finalized, completed document profiles that can be reported. Therefore, while scanning tuple t, $P_{DA}[\text{LF}(t.\text{pos})]$ is complete if all the values in $LQC[1..d][t.\text{ch}] > 0$, meaning we have encountered at least one suffix of all documents $j = 1, \ldots, d$ that is preceded by the character BWT[i]. We then output $P_{DA}[\text{LF}(t.\text{pos})]$ if it corresponds to a sampled position, i.e., if $t.\text{pos}$ is the beginning or the end of a run. We then decrement the counter in LQC[$t.\text{doc}$][$t.\text{ch}$] by 1 and proceed with the next tuple in the queue and we stop at the first tuple corresponding to an incomplete document profile.

We illustrate an example of the document array profiles in Table 1 along with an example query.

Software availability

Our source code (Ahmed et al., 2023) can be found at `https://github.com/oma219/docprofiles`, and our experimental code can be found at `https://github.com/oma219/docprof-experiments`.

Competing Interest Statement

The authors declare that they have no competing interests.

Acknowledgements

This work was carried out at the Advanced Research Computing at Hopkins (ARCH) core facility (rockfish.jhu.edu), which is supported by the National Science Foundation (NSF) grant number OAC 1920103.

Author Contributions

MR conceived of the document array profiles, and proved the theorems and lemmas. OA implemented the software that builds the document array profiles with assistance from BL and MR. All authors wrote, edited and approved the manuscript.

References

Ahmed O, Rossi M, Boucher C, and Langmead B. 2023. Efficient taxa identification using a pangenome index.

Ahmed O, Rossi M, Gagie T, Boucher C, and Langmead B. 2022. SPUMONI 2: Improved pangenome classification using a compressed index of minimizer digests. *bioRxiv* .

Ahmed O, Rossi M, Kovaka S, Schatz MC, Gagie T, Boucher C, and Langmead B. 2021. Pan-genomic matching statistics for targeted nanopore sequencing. *iScience* **24**: 102696.

Burrows M and Wheeler DJ. 1994. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation.

Cobas D, Mäkinen V, and Rossi M. 2020. Tailoring r-index for document listing towards metagenomics applications. In *Proceedings of the International Symposium on String Processing and Information Retrieval (SPIRE)*, pp. 291–306. Springer.

Cobas D and Navarro G. 2019. Fast, small, and simple document listing on repetitive text collections. In *Proceedings of the International Symposium on String Processing and Information Retrieval (SPIRE)*, pp. 482–498. Springer.

Ferragina P and Manzini G. 2000. Opportunistic data structures with applications. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 390–398. IEEE.

Gagie T, Navarro G, and Prezza N. 2020. Fully functional suffix trees and optimal text searching in bwt-runs bounded space. *Journal of the ACM (JACM)* **67**: 1–54.

Jain C, Rodriguez-R LM, Phillippy AM, Konstantinidis KT, and Aluru S. 2018. High throughput ANI analysis of 90k prokaryotic genomes reveals clear species boundaries. *Nature Communications* **9**: 1–8.

Kim D, Song L, Breitwieser FP, and Salzberg SL. 2016. Centrifuge: rapid and sensitive classification of metagenomic sequences. *Genome Research* **26**: 1721–1729.

Kovaka S, Fan Y, Ni B, Timp W, and Schatz MC. 2021. Targeted nanopore sequencing by real-time mapping of raw electrical signal with UNCALLED. *Nature Biotechnology* **39**: 431–441.

Kuhnle A, Mun T, Boucher C, Gagie T, Langmead B, and Manzini G. 2020. Efficient Construction of a Complete Index for Pan-Genomics Read Alignment. *Journal of Computational Biology* **27**: 500–513.

Li H. 2018. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* **34**: 3094–3100.

Manber U and Myers G. 1993. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing* **22**: 935–948.

Muthukrishnan S. 2002. Efficient algorithms for document retrieval problems. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 657–666.

Nasko DJ, Koren S, Phillippy AM, and Treangen TJ. 2018. RefSeq database growth influences the accuracy of k-mer-based lowest common ancestor species identification. *Genome Biology* **19**: 1–10.

Nishimoto T, Kanda S, and Tabei Y. 2022. An optimal-time RLBWT construction in BWT-runs bounded space. In *Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 229 of *LIPIcs*, pp. 99:1–99:20.

Ono Y, Asai K, and Hamada M. 2021. PBSIM2: a simulator for long-read sequencers with a novel generative model of quality scores. *Bioinformatics* **37**: 589–595.

Policriti A and Prezza N. 2016. Computing LZ77 in run-compressed space. In *Proceedings of Data Compression Conference (DCC)*, pp. 23–32.

Puglisi SJ and Zhukova B. 2021. Document retrieval hacks. In *Proceedings of the International Symposium on Experimental Algorithms (SEA)*, p. 12:1–12:12.

Rossi M, Oliva M, Langmead B, Gagie T, and Boucher C. 2022. Moni: A pangenomic index for finding maximal exact matches. *Journal of Computational Biology* **29**: 169–187.

Sadakane K. 2007. Succinct data structures for flexible text retrieval systems. *Journal of Discrete Algorithms* **5**: 12–22.

Wood DE, Lu J, and Langmead B. 2019. Improved metagenomic analysis with Kraken 2. *Genome Biology* **20**: 1–13.

Efficient taxa identification using a pangenome index

Omar Ahmed, Massimiliano Rossi, Christina Boucher, et al.

Genome Res. published online May 31, 2023
Access the most recent version at doi:10.1101/gr.277642.123