Augmented Thresholds for MONI

César Martínez-Guardiola*, Nathaniel K. Brown†, Fernando Silva-Coira*,
Dominik Köppl‡, Travis Gagie† and Susana Ladra*

*Universidade da Coruña
CITIC, A Coruña, Spain
{first.last}@udc.es

†Dalhousie U
Halifax, Canada
{first.last}@dal.ca

‡TMDU
Tokyo, Japan
koeppl.dsc@tmd.ac.jp

2023 Data Compression Conference (DCC) | 979-8-3503-4795-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/DCC55655.2023.00035

Abstract

MONI (Rossi et al., 2022) can store a pangenomic dataset T in small space and later, given a pattern P, quickly find the maximal exact matches (MEMs) of P with respect to T. In this paper we consider its one-pass version (Boucher et al., 2021), whose query times are dominated in our experiments by longest common extension (LCE) queries. We show how a small modification lets us avoid most of these queries which significantly speeds up MONI in practice while only slightly increasing its size.

1 Introduction

The FM-index [1] is one of the most successful compact data structures; DNA alignment has been its "killer app", with FM-based aligners such as Bowtie [2, 3] and BWA [4] racking up tens of thousands of citations and seeing every-day use in labs and clinics worldwide. Standard FM-indexes can handle only a few human genomes at once, however, and geneticists now realize that aligning against only a few standard references biases their research results and medical diagnoses [5]. Among other concerns, this bias undermines personalized medicine particularly for people from ethnic groups — such as African, Central/South Asian, Indigenous, Latin American and Middle Eastern populations — whose genotypes are not reflected well in the standard references or even in public databases of genomes [6]. Countries such as China [7] and Denmark [8] have assembled their own reference sequences, but it is not clear whether and how we can do this fairly for multi-ethnic populations. Bioinformaticians and data-structure designers have therefore been looking for ways to index models that better capture the genetic diversity of whole species, especially humanity. The most publicized approach so far is building and indexing pangenome graphs [9], but we can also try scaling FM-indexes up to handle a dozen or so representative sample genomes [10] or, more ambitiously, to handle even thousands of genomes at once. Indexing thousands of genomes at once is technically challenging, of course, but it should give us a different functionality than pangenome graphs.

Mäkinen et al. [11, 12] initiated the study of indexing massive genomic datasets with their index based on the run-length compressed Burrows-Wheeler Transform (RLBWT), which stores such a pangenomic dataset $T[1..n]$ in space proportional to the number r of runs in the BWT of T and allows us to quickly *count* the number of exact matches of any pattern $P[1..m]$ in T. Policriti and Prezza [13] showed that, if we augment Mäkinen et al.'s index with the entries of the suffix array (SA) sampled

979-8-3503-4795-1/23/$31.00 ©2023 IEEE
DOI 10.1109/DCC55655.2023.00035

at BWT run boundaries, then we can quickly locate *one* of P's matches in T. Gagie, Navarro and Prezza [14] then showed how we can store that SA sample such that we can quickly locate *all* of P's matches in T. (For the sake of brevity, we assume the reader is familiar with the BWT, SA, LF-mapping, etc.; otherwise, we refer them to Mäkinen et al.'s and Navarro's texts [15, 16].) Gagie et al. called their data structure the r-index, after its $O(r)$ space bound; Nishimoto and Tabei [17] recently sped it up to answer queries in optimal time when T is over a polylog(n)-sized alphabet, while still using $O(r)$ space. Boucher et al. [18, 19] showed how we can build an r-index efficiently in practice using a technique they called *prefix-free parsing* (PFP).

Because approximate pattern matching is often more important in bioinformatics than exact matching, Bannai, Gagie and I [20] designed a version of the r-index that can efficiently find maximal exact matches (MEMs), which are commonly used for approximate pattern matching in tools such as BWA-MEM [21]. Bannai et al.'s is not a true r-index because it requires fast random access to T and we do not know how to support that in worst-case $O(r)$ space, but Gagie et al. [22, 23] showed how we can use PFP to build a *straight-line program* (SLP) for T that gives us this random access and in practice takes significantly less space than the r-index itself. The key idea behind Bannai et al.'s index is to store the positions of r thresholds in the RLBWT, one between each consecutive pair of runs of the same character, but they did not give an algorithm for finding those thresholds. Rossi et al. [24] showed that we can choose the thresholds based on the *longest common prefix* (LCP) array and build Bannai et al.'s index efficiently with PFP. They presented a complete implementation, called MONI (Finnish for "multi", as it indexes many genomes at once), and demonstrated its practicality for pangenomic alignment.

By default, MONI makes two passes over P, one right-to-left and then the other left-to-right. Boucher et al. [25] noted, however, that by using an SLP to support longest common extension (LCE) queries instead of random access, MONI can run in one pass. For long patterns, MONI in two-pass mode buffers a significant amount of data during its first pass, so switching to one-pass mode reduces its workspace and allows us to run more queries in parallel. We can also use one-pass MONI for applications that are inherently online, such as recognizing and ejecting non-target DNA strands from nanopore sequencers [26]. Even though one-pass MONI processes most characters in P without LCE queries, the LCE queries it does compute still take most of the query time [25]. We show in this paper that by precomputing and storing two LCE values for each threshold, in practice we can avoid many of those queries and thus significantly speed up one-pass MONI while increasing its size only slightly.

2 MONI

Bannai et al. defined a *threshold* between two consecutive runs BWT$[s_1..e_1]$ and BWT$[s_2..e_2]$ of the same character, to be a position t with $e_1 < t \leq s_2$ such that $\text{LCE}(e_1, k) \geq \text{LCE}(k, s_2)$ for $k < t$, and $\text{LCE}(e_1, k) \leq \text{LCE}(k, s_2)$ for $k \geq t$. (Rossi et al.'s construction is based on the observation that we can set t to the position of a minimum in LCP$[e_1 + 1..s_2]$.) Bannai et al. showed how adding these thresholds to an r-index lets us compute MEMs by computing the matching statistics MS$[1..m]$ of

P with respect to T, where the ith matching statistics MS[i].pos and MS[i].len are defined such that

$$T\left[\text{MS}[i].\text{pos}..\text{MS}[i].\text{pos} + \text{MS}[i].\text{len} - 1\right] = P[i..i + \text{MS}[i].\text{len} - 1]$$

and $P[i..i + \text{MS}[i].\text{len}]$ does not occur in T. In other words, MS[i].pos is a pointer to the starting position in T of a longest match for $P[i..m]$ and MS[i].len is the length of that match, where a *longest match* for $P[i..m]$ is an occurrence in T of the longest prefix of $P[i..m]$ that occurs in T.

Suppose we have already computed MS[$i+1$].pos and the position j of $T[\text{MS}[i+1].\text{pos}-1]$ in the BWT. If $\text{BWT}[j] = P[i]$, then $\text{MS}[i].\text{pos} = \text{MS}[i+1].\text{pos}-1$ and we can continue once we compute the position $\text{LF}(j)$ of $T[\text{MS}[i].\text{pos} - 1]$ in the BWT. Otherwise, let BWT[e] be the last occurrence of $P[i]$ before BWT[j], and BWT[s] be the first occurrence of $P[i]$ after BWT[j]. By the definitions of the BWT and thresholds, if BWT[j] is strictly above the threshold between BWT[e] and BWT[s], then a prefix of $T[\text{SA}[e]..n]$ is a longest match for $P[i..m]$; otherwise, a prefix of $T[\text{SA}[s]..n]$ is a longest match for $P[i..m]$. Since BWT[e] is the end of a run and BWT[s] is the start of a run, we have SA[e] and SA[s] stored. Therefore, depending on whether BWT[j] is above or below the threshold, either we can "jump up" from BWT[j] to BWT[e] and set $\text{MS}[i].\text{pos} = \text{SA}[e]$ (so the position of $T[\text{MS}[i].\text{pos} - 1]$ in the BWT is $\text{LF}(e)$), or we can "jump down" from BWT[j] to BWT[s] and set $\text{MS}[i].\text{pos} = \text{SA}[e]$ (so the position of $T[\text{MS}[i].\text{pos} - 1]$ in the BWT is $\text{LF}(s)$).

By default, MONI makes a right-to-left pass over P to compute MS[$1..m$].pos, and then a left-to-right pass over P to compute MS[$1..m$].len. If we use the SLP to support LCE queries instead of random access, however, then we need only one pass over P. To see why, suppose that when we compute MS[i].pos, we have already computed MS[$i+1$].len as well as MS[$i+1$].pos. If we jump up from BWT[j] to BWT[e], then

$$\text{MS}[i].\text{len} \;=\; \min\left(\text{LCE}(\text{MS}[i+1].\text{pos}, \text{SA}[e]), \text{MS}[i+1].\text{len}\right) + 1\,; \tag{1}$$

if we jump down from BWT[j] to BWT[s], then

$$\text{MS}[i].\text{len} \;=\; \min\left(\text{LCE}(\text{MS}[i+1].\text{pos}, \text{SA}[s]), \text{MS}[i+1].\text{len}\right) + 1\,. \tag{2}$$

In fact, if we compute both $\text{LCE}(\text{MS}[i+1].\text{pos}, \text{SA}[e])$ and $\text{LCE}(\text{MS}[i+1].\text{pos}, \text{SA}[s])$, then we do need not to check the threshold between BWT[e] and BWT[s] at all. MONI stores the thresholds in order to use only one LCE query for each jump, because the thresholds collectively do not take much space compared to the RLBWT and the SA samples, and the LCE queries are slow compared to the LF-steps.

3 Augmented Thresholds

In practice, MONI's jumps and resultant LCE queries tend to occur in bunches: if a character $P[i]$ is a sequencing error or a variation not in T, then we will probably jump for $P[i]$, find a short longest match, and then also jump for several more characters

of P in rapid succession, until the longest matches are finally long enough again to reorient us in the BWT. Because the lengths of the longest matches can only increment for each character of P we process in such cases, most of the comparisons in Equations 1 and 2 between the LCE values and the length of the current longest match will simply return the length of the current match. This observation led us to wonder if all those LCE queries are really necessary.

k	SA[k]	BWT[k]	T[SA[k]..n]
⋮	⋮	⋮	⋮
1234	8765	A	GAGACATCA...
$e_1 =$ 1235	1519	A	GATACATTA...
1236	5450	C	GATAGATTA...
$j =$ 1237	1004	G	GATATAGAA...
1238	4242	G	GATCCAATA...
$t =$ 1239	3110	G	GATTACATA...
1240	1102	T	GATTACTTA...
1241	1978	T	GATTAGATA...
$s_2 =$ 1242	2505	A	GATTATCAT...
1243	2022	A	GATTATGAA...
⋮	⋮	⋮	⋮

Figure 1: Suppose we want to compute MS[i].len for some i such that $P[i] = \text{BWT}[e_1] = \text{BWT}[s_2]$, and the position j of $T[\text{MS}[i+1].\text{pos} - 1]$ in the BWT is between $e_1 + 1$ and $t - 1$. If $\text{MS}[i+1].\text{len} \leq \text{LCE}(\text{SA}[e_1], \text{SA}[t-1])$ then, since $\text{LCE}(\text{SA}[e_1], \text{SA}[t-1]) \leq \text{LCE}(\text{SA}[e_1], \text{SA}[j])$, by transitivity $\text{MS}[i+1].\text{len} \leq \text{LCE}(\text{SA}[e_1], \text{SA}[j])$ and we can safely set $\text{MS}[i].\text{len} = \text{MS}[i+1].\text{len} + 1$.

Suppose that, at the threshold t between two consecutive runs $\text{BWT}[s_1..e_1]$ and $\text{BWT}[s_2..e_2]$ of the same character, we store $\text{LCE}(\text{SA}[e_1], \text{SA}[t-1])$ and $\text{LCE}(\text{SA}[t], \text{SA}[s_2])$. Furthermore, suppose we later want to compute MS[i].len for some i such that $P[i] = \text{BWT}[e_1] = \text{BWT}[s_2]$ and the position j of $T[\text{MS}[i+1].\text{pos} - 1]$ in the BWT is between $e_1 + 1$ and $s_2 - 1$. If $j < t$ and

$$\text{MS}[i+1].\text{len} \leq \text{LCE}(\text{SA}[e_1], \text{SA}[t-1])\,,$$

or $j \geq t$ and

$$\text{MS}[i+1].\text{len} \leq \text{LCE}(\text{SA}[t], \text{SA}[s_2])$$

then, as illustrated in Figure 1, we can safely set $\text{MS}[i].\text{len} = \text{MS}[i+1].\text{len} + 1$ without using an LCE query. Algorithm 1 shows how these values are used to compute MS[1..m] for a given pattern P[1..m] by storing the thresholds alongside these "threshold LCEs".

Threshold LCEs can be computed using LCE queries and SA samples, but their relationship to thresholds allows us to compute both simultaneously. Rossi et al. observed that we can set t to be the position of $min(\text{LCP}[e_1 + 1..s_2])$ using a range-minimum query (RMQ), which they support space-efficiently through PFP and a

Algorithm 1 Computes MS using a variation of one-pass MONI [25] which stores augmented thresholds (thresholds and thr_lce arrays)

$j \leftarrow \text{BWT.select}_{P[m]}(1)$
$\text{MS}[m] \leftarrow (\text{pos} : \text{SA}[j], \text{len} : 1)$
for $i = m - 1$ **down to** 1 **do**
    if $\text{BWT}[j] = P[i]$ **then**
        $\text{MS}[i] \leftarrow (\text{pos} : \text{MS}[i+1].\text{pos} - 1, \text{len} : \text{MS}[i+1].\text{len} + 1)$
    else
        $c \leftarrow \text{BWT.rank}_{P[i]}(j)$
        $e_1 \leftarrow \text{BWT.select}_{P[i]}(c)$
        $s_2 \leftarrow \text{BWT.select}_{P[i]}(c+1)$
        $x \leftarrow \text{BWT.run\_of\_position}(s_2)$ ▷ Position s_2 belongs to the xth run
        $t \leftarrow \text{thresholds}[x]$
        if $j < t$ **then** ▷ thr_lce_e stores $\text{LCE}(\text{SA}[e_1], \text{SA}[t-1])$
            if $\text{MS}[i+1].\text{len} \leq \text{thr\_lce}_e[x]$ **then**
                $\text{MS}[i].\text{len} \leftarrow \text{MS}[i+1].\text{len} + 1$
            else
                $\text{MS}[i].\text{len} \leftarrow min(\text{MS}[i+1].\text{len}, \text{LCE}(\text{SA}[e_1], \text{MS}[i+1].\text{pos})) + 1$
            end if
            $\text{MS}[i].\text{pos} \leftarrow \text{SA}[e_1]$
            $j \leftarrow \text{LF}(e_1)$
        else ▷ thr_lce_s stores $\text{LCE}(\text{SA}[t], \text{SA}[s_2])$
            if $\text{MS}[i+1].\text{len} \leq \text{thr\_lce}_s[x]$ **then**
                $\text{MS}[i].\text{len} \leftarrow \text{MS}[i+1].\text{len} + 1$
            else
                $\text{MS}[i].\text{len} \leftarrow min(\text{MS}[i+1].\text{len}, \text{LCE}(\text{SA}[s_2], \text{MS}[i+1].\text{pos})) + 1$
            end if
            $\text{MS}[i].\text{pos} \leftarrow \text{SA}[s_2]$
            $j \leftarrow \text{LF}(s_2)$
        end if
    end if
end for

range-minimum data structure over the LCP array [24]. We can also define LCE queries as RMQs over the LCP array [27], such that $\text{LCE}(\text{SA}[e_1], \text{SA}[t-1]) = min(\text{LCP}[e_1 + 1..t-1])$ and $\text{LCE}(\text{SA}[t], \text{SA}[s_2]) = min(\text{LCP}[t + 1..s_2])$. These minimums can be computed alongside the thresholds by performing RMQs for the given ranges as thresholds are found by scanning each run boundary, building both the thresholds and the threshold LCEs with only a slight modification to the original MONI method.

4 Experiments

We focus on selected variants of augmented thresholds, which differ in storing the threshold LCEs, and compare them against the unmodified approach:

- `PHONI`: Standard version of one-pass MONI described as $\texttt{PHONI}_{std}$ in original paper [25].
- `Aug-Full`: One-pass MONI modified with augmented thresholds described previously, using $\lg n$-bits per threshold LCE stored.
- `Aug-1`: As above, but caps the size to one byte per threshold LCE. In the event of an overflow, we default to performing a single LCE query.
- `Aug-BV-Full`: Stores a bitvector marking which threshold LCEs are used/non-zero, storing just these values with $O(\lg n)$-bits for each.
- `Aug-BV-1`: As above, but ignores storing values greater than one byte (default to LCE query).
- `Aug-DAC`: Stores threshold LCEs using a directly addressable code (DAC) with escaping, as described and tested on the LCP array by Brisaboa et al. [28].
- `Aug-BV-DAC`: Same as `Aug-BV-Full`, but substituting in a DAC to store defined values.

Our C++ code is available at `https://github.com/drnatebrown/aug_phoni` and is based on the original one-pass MONI code at `https://github.com/koeppl/phoni`. All experiments were executed single-threaded on a server with an Intel(R) Xeon(R) Bronze 3204 CPU and 512 GiB RAM.

To compare against `PHONI` and its existing results, we re-ran Boucher et al.'s query experiments using the same datasets, consisting of chromosome 19 haplotypes (`chr19`), building the data structures for concatenations of 16, 32, 64, 128, 256, 512, and 1000 sequences of `chr19` and querying them with 10 different `chr19` sequences. To support random access and LCE queries efficiently we construct SLPs; both the SLP compressed text of the original one-pass MONI experiments (SLP_{comp}), and the naive uncompressed version of Gagie et al. (SLP_{plain}) [23] that sacrifices space for speed. The datasets and SLP sizes are reported in Table 1. The average query times (computing MS for a single pattern) is shown in Figure 2 where results for both SLP types are accentuated. Similarly, Figure 3 shows the disk sizes for all variants using SLP_{plain}.

5 Conclusion

With respect to query times, we can see that any variants using augmented thresholds are always faster than `PHONI`, and with respect to size, always larger. Introducing the SLP_{plain} clearly benefits all methods by speeding up LCE queries, and although it can be over twice as large as SLP_{comp} when compared directly against each other (Table 1), the difference is much smaller when comparing the total sizes of the data structures shown in Figure 3. This LCE speedup reduces the gap between query times compared to `PHONI`, since it spends a larger percentage of execution on them; however, the LCE queries skipped by augmented thresholds still result faster executions.

#	$n/10^6$	$r/10^4$	n/r	SLP_{comp} [MB]	SLP_{plain} [MB]
16	946.01	3240.02	29.20	36.10	70.54
32	1892.01	3282.51	57.64	37.80	74.75
64	3784.01	3334.06	113.50	39.48	79.84
128	7568.01	3405.40	222.24	42.11	88.89
256	15136.04	3561.98	424.93	47.43	102.52
512	30272.08	3923.60	771.54	58.00	131.09
1,000	59125.12	4592.68	1287.38	80.63	186.98

Table 1: Table summarizing the datasets and sizes of SLPs built over them. The first column describes the number of concatenated sequences of `chr19` representing the text T, where n represents the length of T and r the number of runs.

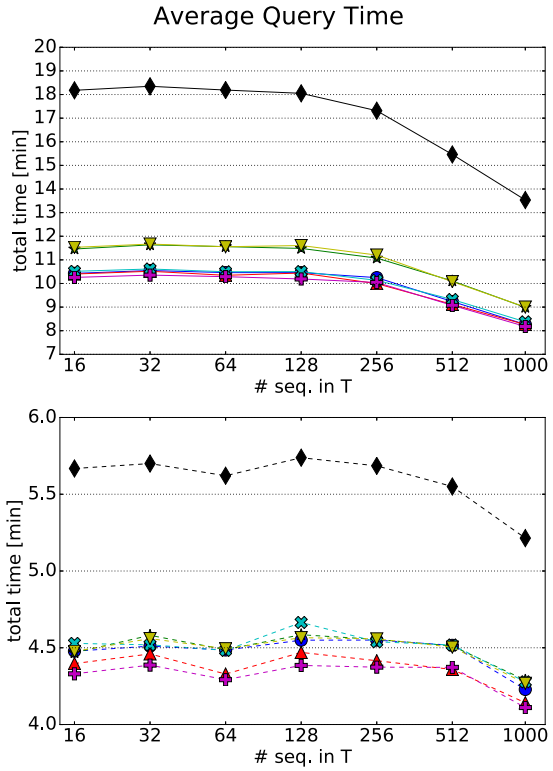

Figure 2: The average query time to compute MS using 10 distinct `chr19` sequences as patterns, using 16, 32, 64, 128, 256, 512, and 1000 sequences of `chr19` as the text T. Data structures shown are as described above. Solid lines use SLP_{comp} (above plot), dashed lines use SLP_{plain} (below plot). Legend is consistent with Figure 3.

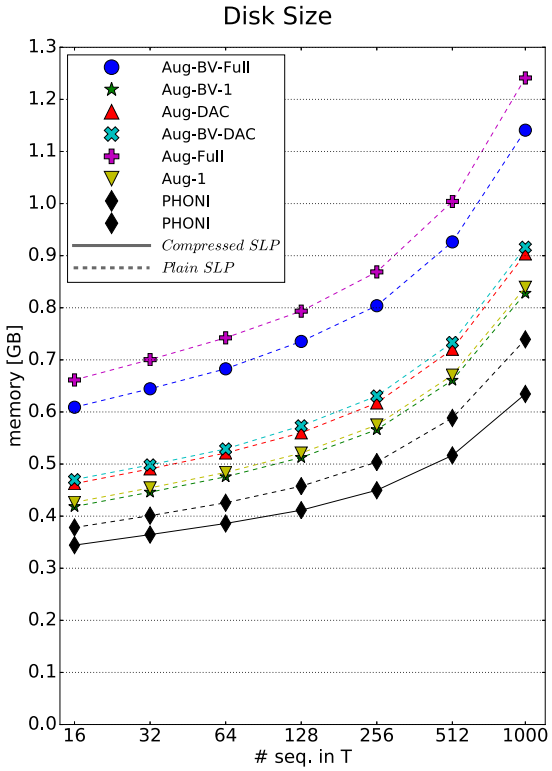

Figure 3: The disk size in GB for each data structure built on 16, 32, 64, 128, 256, 512, and 1000 sequences of `chr19` using mainly a plain SLP. `PHONI` using SLP_{comp} is included to visualize the size difference of the SLP choice; these gaps are not affected by data structure variants and would be equal if others were plotted.

We highlight some standout variants when compared to `PHONI` for the largest text size (1000 sequences of `chr19`). `Aug-DAC` is in the fastest of options regardless of

which SLP is used: 48.37% faster and 22.89% larger for SLP_{comp}, and 22.92% faster and 19.97% larger for SLP_{plain}; a significant improvement compared to the original `PHONI` method (SLP_{comp}) and a direct time/space tradeoff for the introduced SLP_{plain}. The success of this variant aligns with expectation: this DAC is engineered for the LCP array and threshold LCEs consist of small LCP values and zeroes. `Aug-1` is in the smallest class: 40.22% faster and only 14.60% larger for SLP_{comp}, while 19.95% faster and 12.66% larger for SLP_{plain}. Although `Aug-Full` is in the fastest class with `Aug-DAC`, it is much larger. Other variants fall between these approaches in both time and space.

When compared to the original one-pass MONI of Boucher et. al (`PHONI` with SLP_{comp}), our best augmented threshold approaches showed over 40% speed improvements with under 20% space increase on the largest dataset, and similar results across all data. When compared to uncompressed threshold LCEs, our applied compression schemes are space-efficient whilst still being faster than unmodified one-pass MONI. Introducing an uncompressed SLP (SLP_{plain}) experimentally was shown to be of great benefit to both LCE and total query speed, only requiring a small size increase for computing matching statistics on repetitive texts. Using this SLP, results show augmented thresholds to allow a direct time/space tradeoff (increase speed/space by $\approx 20\%$), or a size decrease whilst maintaining a comparable speed increase.

Acknowledgments

Many thanks to Massimiliano Rossi for guidance on implementing this modification.

CMG, FSC and SL supported by CITIC, as Research Center accredited by Galician University System, funded by "Consellería de Cultura, Educación e Universidade from Xunta de Galicia", supported 80% through ERDF Funds, ERDF Operational Programme Galicia 2014-2020, and the remaining 20% by "Secretaría Xeral de Universidades" (Grant ED431G 2019/01); Xunta de Galicia/ERDF under Grant [ED431C 2021/53]; GAIN/ERDF under Grant [IN852D 2021/3];Ministerio de Ciencia e Innovación MCIN/AEI/10.13039/501100011033 and "NextGenerationEU"/PRTR under Grants [TED2021-129245B-C21; PID2020-114635RB-I00; PDC2021-121239-C31; PID2019-105221RB-C41; RTI-2018-098309-B-C32]. NKB and TG supported by National Institutes of Health (NIH) NIAID (grant no. HG011392), the National Science Foundation NSF IIBR (grant no. 2029552) and the Natural Science and Engineering Research Council (NSERC) Discovery Grant (grant no. RGPIN-07185-2020) and a CGS-M scholarship. DK was supported by JSPS KAKENHI Grant Numbers JP22H03551, JP21K17701, and JP21H05847.

References

[1] Paolo Ferragina and Giovanni Manzini, "Indexing compressed text," *Journal of the ACM (JACM)*, vol. 52, no. 4, pp. 552–581, 2005.

[2] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome biology*, vol. 10, no. 3, pp. 1–10, 2009.

[3] Ben Langmead and Steven L Salzberg, "Fast gapped-read alignment with Bowtie 2," *Nature methods*, vol. 9, no. 4, pp. 357–359, 2012.

[4] Heng Li and Richard Durbin, "Fast and accurate short read alignment with Burrows–Wheeler transform," *bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

[5] Sharon Begley, "The reference genome is threatening dream of personalized medicine," https://www.statnews.com/2019/03/11/human-reference-genome-shortcomings, 2019.

[6] Latrice G Landry, Nadya Ali, David R Williams, Heidi L Rehm, and Vence L Bonham, "Lack of diversity in genomic databases is a barrier to translating precision medicine research into practice," *Health Affairs*, vol. 37, no. 5, pp. 780–785, 2018.

[7] Jun Wang, Wei Wang, Ruiqiang Li, Yingrui Li, Geng Tian, Laurie Goodman, Wei Fan, Junqing Zhang, Jun Li, Juanbin Zhang, et al., "The diploid genome sequence of an Asian individual," *Nature*, vol. 456, no. 7218, pp. 60–65, 2008.

[8] Lasse Maretty, Jacob Malte Jensen, Bent Petersen, Jonas Andreas Sibbesen, Siyang Liu, Palle Villesen, Laurits Skov, Kirstine Belling, Christian Theil Have, Jose MG Izarzugaza, et al., "Sequencing and de novo assembly of 150 genomes from Denmark as a population reference," *Nature*, vol. 548, no. 7665, pp. 87–91, 2017.

[9] Jouni Sirén, Jean Monlong, Xian Chang, Adam M Novak, Jordan M Eizenga, Charles Markello, Jonas A Sibbesen, Glenn Hickey, Pi-Chuan Chang, Andrew Carroll, et al., "Pangenomics enables genotyping of known structural variants in 5202 diverse genomes," *Science*, vol. 374, no. 6574, pp. abg8871, 2021.

[10] Nae-Chyun Chen, Brad Solomon, Taher Mun, Sheila Iyer, and Ben Langmead, "Reference flow: reducing reference bias using multiple population genomes," *Genome biology*, vol. 22, no. 1, pp. 1–17, 2021.

[11] Jouni Sirén, Niko Välimäki, Veli Mäkinen, and Gonzalo Navarro, "Run-length compressed indexes are superior for highly repetitive sequence collections," in *Proc. SPIRE*. Springer, 2008, pp. 164–175.

[12] Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki, "Storage and retrieval of highly repetitive sequence collections," *Journal of Computational Biology*, vol. 17, no. 3, pp. 281–308, 2010.

[13] Alberto Policriti and Nicola Prezza, "Lz77 computation based on the run-length encoded BWT," *Algorithmica*, vol. 80, no. 7, pp. 1986–2011, 2018.

[14] Travis Gagie, Gonzalo Navarro, and Nicola Prezza, "Fully functional suffix trees and optimal text searching in BWT-runs bounded space," *Journal of the ACM (JACM)*, vol. 67, no. 1, pp. 1–54, 2020.

[15] Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, and Alexandru I Tomescu, *Genome-scale algorithm design*, Cambridge University Press, 2015.

[16] Gonzalo Navarro, *Compact data structures: A practical approach*, Cambridge University Press, 2016.

[17] Takaaki Nishimoto and Yasuo Tabei, "Optimal-time queries on BWT-runs compressed indexes," in *Proc. ICALP*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[18] Christina Boucher, Travis Gagie, Alan Kuhnle, Ben Langmead, Giovanni Manzini, and Taher Mun, "Prefix-free parsing for building big BWTs," *Algorithms for Molecular Biology*, vol. 14, no. 1, pp. 1–15, 2019.

[19] Alan Kuhnle, Taher Mun, Christina Boucher, Travis Gagie, Ben Langmead, and Giovanni Manzini, "Efficient construction of a complete index for pan-genomics read alignment," *Journal of Computational Biology*, vol. 27, no. 4, pp. 500–513, 2020.

[20] Hideo Bannai, Travis Gagie, and Tomohiro I, "Refining the r-index," *Theoretical Computer Science*, vol. 812, pp. 96–108, 2020.

[21] Heng Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM," *arXiv preprint arXiv:1303.3997*, 2013.

[22] Travis Gagie, Giovanni Manzini, Gonzalo Navarro, Hiroshi Sakamoto, and Yoshimasa Takabatake, "Rpair: rescaling RePair with rsync," in *Proc. SPIRE*. Springer, 2019, pp. 35–44.

[23] Travis Gagie, Giovanni Manzini, Gonzalo Navarro, Hiroshi Sakamoto, Louisa Seelbach Benkner, and Yoshimasa Takabatake, "Practical random access to SLP-compressed texts," in *Proc. SPIRE*. Springer, 2020, pp. 221–231.

[24] Massimiliano Rossi, Marco Oliva, Ben Langmead, Travis Gagie, and Christina Boucher, "MONI: A pangenomic index for finding maximal exact matches," *Journal of Computational Biology*, vol. 29, no. 2, pp. 169–187, 2022.

[25] Christina Boucher, Travis Gagie, Tomohiro I, Dominik Köppl, Ben Langmead, Giovanni Manzini, Gonzalo Navarro, Alejandro Pacheco, and Massimiliano Rossi, "PHONI: Streamed matching statistics with multi-genome references," in *Proc. DCC*. IEEE, 2021, pp. 193–202.

[26] Omar Ahmed, Massimiliano Rossi, Sam Kovaka, Michael C Schatz, Travis Gagie, Christina Boucher, and Ben Langmead, "Pan-genomic matching statistics for targeted nanopore sequencing," *Iscience*, vol. 24, no. 6, pp. 102696, 2021.

[27] Lucian Ilie, Gonzalo Navarro, and Liviu Tinta, "The longest common extension problem revisited and applications to approximate string searching," *Journal of Discrete Algorithms*, vol. 8, no. 4, pp. 418–428, 2010.

[28] Nieves R. Brisaboa, Susana Ladra, and Gonzalo Navarro, "DACs: Bringing direct access to variable-length codes," *Information Processing & Management*, vol. 49, no. 1, pp. 392–404, 2013.