Faster Maximal Exact Matches with Lazy LCP Evaluation

Adrián Goga[*], Lore Depuydt[†], Nathaniel K. Brown[‡],
Jan Fostier[†], Travis Gagie[§] and Gonzalo Navarro[¶]

[*]Dept. of Comp. Sci.
Comenius University
Bratislava, Slovakia
adrian.goga@fmph.uniba.sk

[†]Dept. of Inf. Tech.
Ghent University
Ghent, Belgium
lore.depuydt@ugent.be,
jan.fostier@ugent.be

[‡]Dept. of Comp. Sci.
Johns Hopkins University
Baltimore, USA
nbrown99@jhu.edu

[§]CeBiB & Fac. of Comp. Sci.
Dalhousie University
Halifax, Canada
travis.gagie@dal.ca

[¶]CeBiB & Dept. of Comp. Sci.
University of Chile
Santiago, Chile
gnavarro@dcc.uchile.cl

Abstract

MONI (Rossi et al., *JCB* 2022) is a BWT-based compressed index for computing the matching statistics and maximal exact matches (MEMs) of a pattern (usually a DNA read) with respect to a highly repetitive text (usually a database of genomes) using two operations: LF-steps and longest common extension (LCE) queries on a grammar-compressed representation of the text. In practice, most of the operations are constant-time LF-steps but most of the time is spent evaluating LCE queries Adrian: concrete numbers?. In this paper we show how (a variant of) the latter can be evaluated lazily, so as to bound the total time MONI needs to process the pattern in terms of the number of MEMs between the pattern and the text, while maintaining logarithmic latency.

1 Introduction

The FM-index [1] is one of the most popular data structures in bioinformatics — its co-inventors, Paolo Ferragina and Giovanni Manzini, recently shared the Paris Kanellakis Award with Mike Burrows, the co-inventor of the Burrows-Wheeler Transform (BWT) on which the FM-index is based — and it drives the most popular short-read DNA aligners, such as Bowtie [2, 3] and BWA [4]. These aligners generally use a single reference genome but, as bioinformaticians have realized how much that biases research results and medical diagnoses, there have been efforts to scale the FM-index to thousands or more genomes at once.

The first such effort was the run-length compressed BWT (RLBWT) by Mäkinen, Navarro, Sirén, and Välimäki [5], which can support fast counting queries in $O(r)$ space, where r is the number of runs in the BWT of the indexed text. For a highly repetitive text, such as a collection of genomes from the same or closely related species, r is orders of magnitude smaller than the size of the uncompressed text. The RLBWT cannot support fast locating queries in $O(r)$ space, however, and it took several years before Gagie, Navarro, and Prezza [6] proposed the r-index, which can. Soon after, Bannai, Gagie, and I [7] showed how to modify the r-index — adding a grammar-compressed representation of the indexed text to support longest common extension (LCE) queries — to compute matching statistics and thus maximal exact matches (MEMs). MEMs have been among the most popular kinds of seeds for DNA alignments at least since the introduction of BWA-MEM [8].

Bannai et al.'s design was implemented by Rossi et al. [9] as their tool MONI, which was used as the basis for SPUMONI [10], SPUMONI 2 [11] and Sigmoni [12]. To compute the matching statistics of a pattern with respect to an indexed text $T[1..n]$, MONI uses two operations: constant-time LF-steps, where $\mathrm{LF}(i)$ is the position in the BWT of the character preceding $\mathrm{BWT}[i]$ in T, and longest common extension (LCE) queries, where $\mathrm{LCE}(i, j)$ is the length of the longest common prefix of the suffixes $T[i..n]$ and $T[j..n]$ of T. LF-steps take constant time in theory [13] and, when T is over a constant-sized alphabet, also in practice.

In contrast, LCE queries on a grammar-compressed representation of T are slow: Bille et al. [14] gave an $O(g)$-space data structure, where g is the number of rules in a given straight-line program (SLP) for T, that answers $\mathrm{LCE}(i, j)$ in $O(\log n + \log^2 \mathrm{LCE}(i, j))$ time; I [15] gave an $O(g)$-space structure with $O(\log n)$ query time, but it works only with SLPs built with recompression [16] (which are larger in practice than SLPs built with heuristics). MONI uses a simple $O(g)$-space representation of a height-balanced SLP — which is not a significant restriction in theory [17] or in practice — and a heuristic for queries that works fairly well in practice but can use $\Omega(\log n + \mathrm{LCE}(i, j))$ time to answer $\mathrm{LCE}(i, j)$ in the worst case. Martínez-Guardiola et al. [18] gave another heuristic that allows us to skip some LCE queries and found that in practice it gives a significant speedup, albeit without a theoretical bound.

Very recently, Baláž et al. [19] observed that we can replace LCE queries by simpler longest common prefix (LCP) queries between suffixes of the pattern $P[1..m]$ and the text $T[1..n]$, such that $\mathrm{LCP}(S_1, S_2)$ is the length of the longest common prefix of strings S_1 and S_2 (which may not be suffixes of the same string. If we precompute the Karp-Rabin hashes of the prefixes of P in $O(m)$ time, then we can use a modification of Bille et al.'s structure — assuming the SLP is height-balanced — to answer LCP queries between suffixes of P and T in $O(\log n)$ time and correctly with high probability. Therefore, we can compute the matching statistics and MEMs of P with respect to T in $O(m \log n)$ time and correctly with high probability.

In this paper, we show how using LCP queries also allows us to evaluate those queries *lazily* and thus compute the matching statistics and MEMs in $O(m + \mu \log(m/\mu) \log n)$ total time with high probability, where $\mu \leq m$ is the number of MEMs of P with respect to T. We can do this while always knowing the matching statistics for all but the $O(\log n)$ characters of P we have processed most recently. We alo briefly sketch two practical results: first, we show how Cobas, Gagie, and Navarro's [20] scheme for subsampling the suffix array can be simplified when used with MONI instead of with the original r-index; second, we extend our techniques to a simple and practical algorithm to find quickly either all MEMs of at least a given length or all the longest common substrings (LCSs) of P and T.

2 Preliminaries

2.1 Matching Statistics, MEMs and LCSs

The *matching statistics* of a pattern $P[1..m]$ with respect to a text $T[1..n]$ are an array $\mathrm{MS}[1..m]$ of $(\mathrm{pos}, \mathrm{len})$ pairs such that

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

and $P[i..i + \mathrm{MS}[i].\mathrm{len}]$ does not occur in T. In other words, $\mathrm{MS}[i].\mathrm{len}$ is the length of the longest prefix of $P[i..m]$ that occurs in T and $\mathrm{MS}[i].\mathrm{pos}$ is the position of one of its occurrences in T. Furthermore, for $\mathrm{x} \in \{\mathrm{pos}, \mathrm{len}\}$ we let $M[i..j].\mathrm{x} = M[i].\mathrm{x}, \ldots, M[j].\mathrm{x}$.

One of the main uses of matching statistics is computing *maximal exact matches* (MEMs). A substring $P[i..j]$ of P has an exact match in T if $j < i + \mathrm{MS}[i].\mathrm{len}$, and is called a MEM of P with respect to T if $j = i + \mathrm{MS}[i].\mathrm{len} - 1$ and $\mathrm{MS}[i-1].\mathrm{len} \leq \mathrm{MS}[i].\mathrm{len}$. In other words, $P[i..j]$ is a MEM if it occurs in T but $P[i-1..j]$ and $P[i..j+1]$ do not. Since we cannot have two MEMs $P[i_1..j_1]$ and $P[i_2..j_2]$ nested — that is, with $i_1 \leq i_2 \leq j_2 \leq j_1$ — the number μ of MEMs of P with respect to T is at most m, and is usually significantly less in practice.

The *longest common substrings* (LCSs) of P and T are the maximum MEMs. Finding LCSs is one of the classic problems of stringology, and indexes for it have played a key role at least since Weiner's [21] optimal-time solution with suffix trees.

2.2 MONI

The main component of MONI is a run-length compressed Burrows-Wheeler Transform (BWT) of the text $T[1..n]$, with the suffix array (SA) sampled at the beginning and end of every BWT run. Between any two consecutive runs $\mathrm{BWT}[s_1..e_1]$ and $\mathrm{BWT}[s_2..e_2]$ of the same character in the BWT, MONI also stores a *threshold* t with $e_1 < t \leq s_2$ such that $\mathrm{LCE}(\mathrm{SA}[q], \mathrm{SA}[e_1]) \geq \mathrm{LCE}(\mathrm{SA}[q], \mathrm{SA}[s_2])$ for $q < t$ and $\mathrm{LCE}(\mathrm{SA}[q], \mathrm{SA}[e_1]) \leq \mathrm{LCE}(\mathrm{SA}[q], \mathrm{SA}[s_2])$ for $q \geq t$, where the *longest common extension* $\mathrm{LCE}(x, y)$ of two suffixes $T[x..n]$ and $T[y..n]$ of T is the length of their longest common prefix. Rossi et al. noted that we can choose t to be the position of a minimum in $\mathrm{LCP}[e_1+1..s_2]$, where $\mathrm{LCP}[j]$ — not to be confused with $\mathrm{LCP}(S_1, S_2)$ — is the length of the longest common prefix of $T[\mathrm{SA}[j-1]..n]$ and $T[\mathrm{SA}[j]..n]$. Finally, MONI stores a balanced straight-line program (SLP) for T to support LCE queries.

Suppose we know $\mathrm{MS}[i+1]$ and the lexicographic rank q of $T[\mathrm{MS}[i+1].\mathrm{pos}..n]$ among the suffixes of T (so $\mathrm{SA}[q] = \mathrm{MS}[i+1].\mathrm{pos}$). If $\mathrm{BWT}[q] = P[i]$, then

$$\begin{aligned} \mathrm{MS}[i].\mathrm{pos} &= \mathrm{MS}[i+1].\mathrm{pos} - 1 \\ \mathrm{MS}[i].\mathrm{len} &= \mathrm{MS}[i+1].\mathrm{len} + 1 \end{aligned}$$

and the lexicographic rank of $T[\mathrm{MS}[i].\mathrm{pos}..n]$ among the suffixes of T is $\mathrm{LF}(q)$, where LF is the last-to-first function that maps the position in the BWT of a character to the position in the BWT of that character's predecessor in T.

If $\mathrm{BWT}[q] \neq P[i]$ and $\mathrm{BWT}[s_1..e_1]$ and $\mathrm{BWT}[s_2..e_2]$ are the runs of copies of $P[i]$ preceding and following $\mathrm{BWT}[q]$, respectively, then by the definition of the BWT either

$$\begin{aligned} \mathrm{MS}[i].\mathrm{pos} &= \mathrm{SA}[e_1] - 1 \\ \mathrm{MS}[i].\mathrm{len} &= \min\big(\mathrm{MS}[i+1].\mathrm{len}, \mathrm{LCE}(\mathrm{MS}[i+1].\mathrm{pos}, \mathrm{SA}[e_1])\big) + 1 \end{aligned}$$

or

$$\begin{aligned} \mathrm{MS}[i].\mathrm{pos} &= \mathrm{SA}[s_2] - 1 \\ \mathrm{MS}[i].\mathrm{len} &= \min\big(\mathrm{MS}[i+1].\mathrm{len}, \mathrm{LCE}(\mathrm{MS}[i+1].\mathrm{pos}, \mathrm{SA}[s_2])\big) + 1 \end{aligned}$$

and we can tell which by comparing q to the threshold between $\mathrm{BWT}[s_1..e_1]$ and $\mathrm{BWT}[s_2..e_2]$. Notice that, as e_1 and s_2 are the end of a run and the beginning of one, respectively, we have $\mathrm{SA}[e_1]$ and $\mathrm{SA}[s_2]$ stored.

MONI stores its SLP for T augmented such that each symbol is annotated with its expansion's length, allowing random access to a substring of T of length ℓ in $O(\log n + \ell)$ time. To evaluate $\mathrm{LCE}(x, y)$, we extract and compare $T[x..n]$ and $T[y..n]$, essentially performing depth-first traversals of the parse tree starting at the xth and yth leaves. Whenever those traversals would descend into copies of the same subtree at the same time, however, we know that the substrings we extract from them will be the same, so we can skip them. This heuristic works fairly well in practice but can use $\Omega(\log n + \mathrm{LCE}(x, y))$ time in the worst case. Of course, if we are evaluating $\mathrm{LCE}(x, y)$ in order to take the minimum of that value and the current matching-statistics length, then we can stop once we know $\mathrm{LCE}(x, y)$ is at least that length.

If we compute both $\mathrm{LCE}(\mathrm{MS}[i+1].\mathrm{pos}, \mathrm{SA}[e_1])$ and $\mathrm{LCE}(\mathrm{MS}[i+1].\mathrm{pos}, \mathrm{SA}[s_2])$, then we do not need the threshold t. In practice, however, the space for the thresholds is small and it is worthwhile to store them to halve the number of LCE queries we perform. Martínez-Guardiola et al. [18] showed that in practice we can often avoid even more LCE queries if we precompute and store the LCE values $\mathrm{LCE}(\mathrm{SA}[t - 1], \mathrm{SA}[e_1])$ and $\mathrm{LCE}(\mathrm{SA}[t], \mathrm{SA}[s_2])$ between each threshold t and the corresponding run boundaries e_1 and s_2, which takes a total of $O(r)$ space. This is because, if $e_1 < q < t$ then

$$\mathrm{LCE}(\mathrm{SA}[t-1], \mathrm{SA}[e_1]) \leq \mathrm{LCE}(\mathrm{SA}[q], \mathrm{SA}[e_1])\,,$$

so if

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

then $\mathrm{MS}[i].\mathrm{len} = \mathrm{MS}[i+1].\mathrm{len} + 1$; the case when $t \leq q < s_2$ is symmetric. At and after a sequencing error in a read, for example, we often find several mismatches bunched together with small matching-statistics lengths until we have processed enough characters of P to re-orient ourselves in the BWT, and Martínez-Guardiola et al.'s heuristic lets us avoid the corresponding LCE queries. In their experiments, they found this speeds up computing matching statistics by about 20%.

2.3 Bille et al.'s Structure and Computing MEMs with LCP Queries

Bille et al.'s [14] data structure is an SLP further augmented such that each symbol is annotated with the Karp-Rabin hash of its expansion, allowing hashing of $T[1..x-1]$ in $O(\log n)$ time and then subsequent hashing of $T[x..x+\ell-1]$ in $O(\log \ell)$ time for any ℓ. In theory it does not matter whether the SLP is height-balanced, since we can balance it without increasing its size by more than a constant factor [17], but in practice it should be. To compute $\mathrm{LCE}(x, y)$, we use exponential search to find the largest ℓ such that the hashes of $T[x..x+\ell-1]$ and $T[y..y+\ell-1]$ are equal, in $O(\log n + \log^2 \mathrm{LCE}(x, y))$ total time. With $O(n \log n)$ expected-time preprocessing, we can find a Karp-Rabin hash with no collisions between the substrings of T [22], with which Bille et al.'s structure answers all LCE queries correctly.

Baláž et al. [19] recently noted that

$$\begin{aligned}&\min\left(\mathrm{MS}[i+1].\mathrm{len}, \mathrm{LCE}(\mathrm{MS}[i+1].\mathrm{pos}, \mathrm{SA}[e_1])\right)\\ &\quad= \ \mathrm{LCP}\left(P[i+1..i+\mathrm{MS}[i+1].\mathrm{len}], T[\mathrm{SA}[e_1]..n]\right)\end{aligned}$$

$$\begin{aligned}&\min\left(\mathrm{MS}[i+1].\mathrm{len}, \mathrm{LCE}(\mathrm{MS}[i+1].\mathrm{pos}, \mathrm{SA}[s_2])\right)\\ &\quad= \ \mathrm{LCP}\left(P[i+1..i+\mathrm{MS}[i+1].\mathrm{len}], T[\mathrm{SA}[s_2]..n]\right) .\end{aligned}$$

In other words, we can replace LCE queries with LCP queries when we are computing matching statistics and MEMs.

If we precompute the Karp-Rabin hashes of all the prefixes of $P[1..m]$ in $O(m)$ total time, then afterward we can use Bille et al.'s structure — assuming the SLP is given height-balanced or we have balanced it — to answer $\mathrm{LCP}(P[i+1..i+\mathrm{MS}[i+1].\mathrm{len}], T[y..n])$ in $O(\log n)$ time and correctly with high probability. To do this, we descend to the $(y-1)$st leaf of the parse tree for T and compute the hash for $T[1..y-1]$; re-ascend the tree until we reach a symbol X with expansion $T[z..w]$ such that $w-y > \mathrm{MS}[i+1].\mathrm{len}$ or the hash of $T[y..w]$ is not equal to the hash of $P[i+1..(i+1)+w-y]$; and finally descend to the $(y + \mathrm{LCP}(P[i+1..i+\mathrm{MS}[i+1].\mathrm{len}], T[y..n]) - 1)$st leaf.

Since $P[i+1..i+\mathrm{MS}[i+1].\mathrm{len}]$ occurs in T, if we spend $O(n \log n)$ expected-time preprocessing choosing the Karp-Rabin hash function, Bille et al.'s structure answers $\mathrm{LCP}(P[i+1..i+\mathrm{MS}[i+1].\mathrm{len}], T[y..n])$ correctly. Since we compute the matching statistics from right to left, by induction, all the lengths are correct and we obtain the following result:

Theorem 1 ([19, Lemma 1]) *We can store a text $T[1..n]$ in $O(r+g)$ space, where r is the number of runs in the BWT of T and g is the number of rules in a given SLP for T, such that later, given $P[1..m]$, we can compute the matching statistics and MEMs of P correctly with respect to T in $O(m \log n)$ worst-case time.*

3 MEMs in $O(m + \mu \log(m/\mu) \log n)$ Time

Martínez-Guardiola et al.'s heuristic lets us compute matching lengths for some mismatches without evaluating the corresponding LCE queries but, when computing matching statistics with MONI and LCE queries, we know of no general way to compute matching-statistic lengths except in right-to-left order. Suppose we have $i_1 < i_2 < i_3$ and we computed $\mathrm{MS}[i_1..m].\mathrm{pos}$ and $\mathrm{MS}[i_3..m].\mathrm{len}$, for example, and

$$\begin{aligned} P[i_1] &\neq T[\mathrm{MS}[i_1+1].\mathrm{pos}-1] \\ P[i_2] &\neq T[\mathrm{MS}[i_2+1].\mathrm{pos}-1] \\ P[i_3] &\neq T[\mathrm{MS}[i_3+1].\mathrm{pos}-1] \end{aligned}$$

but $P[i'] = T[\mathrm{MS}[i'+1].\mathrm{pos}-1]$ for all $i' \neq i_2$ strictly between i_1 and i_3. If we perform LCE queries when processing $P[i_1]$ and $P[i_3]$ but not when processing $P[i_2]$, then in general we do not see how to continue and compute $\mathrm{MS}[1..i_2].\mathrm{len}$.

On the other hand, if we know $\mathrm{MS}[i_1].\mathrm{pos}$ then we can compute $\mathrm{MS}[i_1].\mathrm{len} = \mathrm{LCP}(P[i_1..m], T[\mathrm{MS}[i_1].\mathrm{pos}..n])$ even without knowing matching-statistics lengths further to the right. This could be useful for parallelizing MONI and, more intriguingly, for reducing the number of LCP queries we evaluate. To see why, suppose $P[i_1..i_1+\mathrm{MS}[i_1].\mathrm{len}-1]$, $P[i_2..i_2+\mathrm{MS}[i_2].\mathrm{len}-1]$ and $P[i_3..i_3+\mathrm{MS}[i_3].\mathrm{len}-1]$ are all suffixes of the same MEM, which ends at position

$$\begin{aligned} & i_1+\mathrm{MS}[i_1].\mathrm{len}-1 = (i_1+1)+\mathrm{MS}[i_1+1].\mathrm{len}-1 = \cdots \\ = \ & (i_3-1)+\mathrm{MS}[i_3-1].\mathrm{len}-1 = i_3+\mathrm{MS}[i_3].\mathrm{len}-1 \end{aligned}$$

in P. Then, once we know $\mathrm{MS}[i_1].\mathrm{len}$ and $\mathrm{MS}[i_3].\mathrm{len}$, we can infer $\mathrm{MS}[i_2].\mathrm{len}$ — and all of $\mathrm{MS}[i_1..i_3].\mathrm{len}$ — without evaluating $\mathrm{LCP}(P[i_2..m], T[\mathrm{MS}[i_2].\mathrm{pos}])$ directly.

Working right to left, we can compute $\mathrm{MS}[1..m].\mathrm{pos}$ without LCP queries, and then find the start of each MEM in P using exponential search (still only evaluating LCP queries when $P[i] \neq T[\mathrm{MS}[i+1].\mathrm{pos}-1]$). This way, we can compute the matching statistics of P with respect to T using $O(\mu \log(m/\mu))$ LCP queries, in a total of $O(m + \mu \log(m/\mu) \log n) \subseteq O(m \log n)$ time, where $\mu \leq m$ is again the number of MEMs. It does not change our asymptotic time bounds but, since each step in the exponential search for the start of a MEM requires testing only whether the LCP of a suffix of P and a suffix of T extends at least to the end of that MEM, we can use substring-equality checks — which are easier to implement in practice — rather than full LCP queries.

This has low latency when all MEMs are fairly short and we compute $\mathrm{MS}[i].\mathrm{pos}$ and $\mathrm{MS}[i].\mathrm{len}$ entries more or less simultaneously. In other words, when all MEMs are fairly short we can always compute both $\mathrm{MS}[i].\mathrm{pos}$ and $\mathrm{MS}[i].\mathrm{len}$ fairly quickly after processing $P[i]$. However, suppose there is a long MEM $P[i..i+2^k+1]$ and, in our exponential search for the start of that MEM, we perform LCP queries when processing $P[i+2^k+1], P[i+2^k], P[i+2^k-1], P[i+2^k-3], P[i+2^k-7], \ldots, P[(i+2^k+1)-2^k = i+1]$. We may not perform another LCP query until we process $P[(i+2^k+1)-2^{k+1} = i-2^k+1]$, so we do not learn $\mathrm{MS}[i].\mathrm{len}$ until we have processed 2^k-1 characters after processing $P[i]$.

Since an LCP query takes $O(\log n)$ time, however, we can perform an additional one (again, when $P[i] \neq T[\mathrm{MS}[i+1].\mathrm{pos}-1]$) after processing every $\log n$ characters of P while using only $O(m)$ extra time. Done carefully, this guarantees we use

$O(m/\log n + \mu \log(m/\mu))$ LCP queries and $O(m + \mu \log(m/\mu) \log n)$ time while computing the matching statistics, and we always know the matching-statistics lengths for all but the $\log n$ characters of P we have processed most recently.

When evaluating the LCP queries lazily, we cannot be sure that the substrings of P we pass as arguments are all substrings of T, and thus we cannot rule out the possibility of hash values colliding. This would mean we obtain the correct matching-statistics lengths of P with respect to T only with high probability, but we can ensure their correctness if we weaken our time bound from holding in the worst case to holding with high probability. To do this, we first compute the (probably correct) matching statistics, then compute the (probably correct) MEMs, and finally, for each MEM from left to right in P, use Bille et al.'s structure to extract from T the suffix of that MEM that does not overlap any MEM further to the left in P and compare the characters in that suffix to the corresponding ones in P.

If all the pairs of corresponding characters in all suffixes are equal, then we verify the matching statistics in a total of $O(m+\mu \log n)$ time. We note that this verification process lets us check for supposed MEMs that extend too far to the right, but not for ones that do not extend far enough to the right. Fortunately, by inspection of how we compute an LCP and the fact that Karp-Rabin hashing can indicate false-positive matches but not false-negative mismatches, we can never underestimate an LCP and so we can never underestimate how far a MEM extends to the right.

If any of the pairs or corresponding characters are not equal — which happens with low probability — then we detect a hash collision in $O(m \log n)$ time. In this case, we can compute the matching statistics naïvely in $O(m(\log n + m))$ time by extracting and scanning $T[\mathrm{MS}[1].\mathrm{pos}..\mathrm{MS}[1].\mathrm{pos} + m - 1], T[\mathrm{MS}[2].\mathrm{pos}..\mathrm{MS}[2].\mathrm{pos} + m - 2], \ldots, T[\mathrm{MS}[m].\mathrm{pos}]$. Because this case happens with low probability, we still use $O(m + \mu \log(m/\mu) \log n)$ time overall with high probability. This approach does not require the $O(n \log n)$ expected-time preprocessing Bille et al. use for derandomization.

Theorem 2 *We can store a text $T[1..n]$ in $O(r + g)$ space, where r is the number of runs in the BWT of T and g is the number of rules in a given SLP for T, such that later, given $P[1..m]$, we can compute the matching statistics of P with respect to T in $O(m + \mu \log(m/\mu) \log n) \subseteq O(m \log n)$ time with high probability, where μ is the number of MEMs of P with respect to T. We work right to left and always know the matching-statistics positions for all the characters of P we have processed and the matching-statistics lengths for all but the $\log n$ characters that we have processed most recently.*

4 Practical Results

In this section we first show how Cobas et al.'s [20] scheme for subsampling the SA can be simplified and slighly improved when used with MONI instead of with the original r-index. Although they showed that their scheme significantly reduces the space of the r-index without significantly affecting its query time, this is the first time it has been used with MONI, with or without our further optimization. Because short MEMs are poor seeds and not interesting for many applications, we then give a simple algorithm that uses LCPs to find either all MEMs of P with respect to T or all LCSs of P and T. Despite the simplicity of our modified subsampling scheme and of our algorithm, we have found no good theoretical bounds for them, so we give some preliminary experimental results for them. Due to space constraints, however, we leave a complete evaluation for the full version of this paper.

4.1 Subsampling

Cobas et al. noticed that, in practice, if $\mathrm{BWT}[i]$ is at the beginning or end of a run in the BWT, then $\mathrm{BWT}[\mathrm{LF}(i)]$ is often at the beginning or end of a run as well. In such cases, if we store $\mathrm{SA}[\mathrm{LF}(i)] - 1$ corresponding to $\mathrm{BWT}[\mathrm{LF}(i)]$, then we need not store $\mathrm{SA}[i] - 1$ for $\mathrm{BWT}[i]$ as well. They devised a scheme that takes a parameter s and subsamples the SA entries at boundaries of run in the BWT such that

- if $\mathrm{BWT}[i]$ is at the beginning or end of a run, then one of $\mathrm{SA}[i]-1, \mathrm{SA}[\mathrm{LF}(i)]-1 = \mathrm{SA}[i]-2, \ldots, \mathrm{SA}[\mathrm{LF}^{s-1}(i)]-1 = \mathrm{SA}[i]-s$ is subsampled;
- three SA samples $\mathrm{SA}[i]-1$, $\mathrm{SA}[j]-1$ and $\mathrm{SA}[k]-1$ with $\mathrm{SA}[i] < \mathrm{SA}[j] < \mathrm{SA}[k] \leq \mathrm{SA}[i]+s$ are never all subsampled;
- if $\mathrm{BWT}[j]$ is at the beginning or end of a run and sample $(\mathrm{SA}[j]-1)$'s predecessor and successor among the sorted sampled SA values are $\mathrm{SA}[i]-1$ and $\mathrm{SA}[k]-1$ with $\mathrm{SA}[k]-\mathrm{SA}[i] > s$, then $\mathrm{SA}[j]$ is always subsampled.

The third constraint guarantees that ϕ queries [23] can be evaluated with at most $s-1$ LF-steps. Although MONI uses ϕ queries when enumerating MEMs' occurrences in T, it does not use them when computing matching statistics, so here we can omit that constraint and obtain even smaller SA subsamples.

4.2 Finding Long MEMs

If we are interested only in MEMs of length at least d, where d is a reasonably large parameter, then we can use faster version of Theorem 1 that is simpler than Theorem 2. Assume we have already found all the MEMs of length at least d whose left endpoints are strictly to the right of $P[j]$, and just computed

$$\mathrm{MS}[j].\mathrm{len} = \mathrm{LCP}\left(P[j..m], T\left[\mathrm{MS}[j].\mathrm{pos}..n\right]\right) < d$$

while processing $P[j]$. Because $\mathrm{MS}[i].\mathrm{len} \leq \mathrm{MS}[i{+}1].\mathrm{len}{+}1$, it follows that $\mathrm{MS}[i].\mathrm{len} < d$ for $j - d + \mathrm{MS}[j].\mathrm{len} < i \leq j$ and so, even if $P[i] \neq T[\mathrm{MS}[i+1].\mathrm{pos}-1]$, we need not perform another LCP query for processing $P[i]$ while $j - d + \mathrm{MS}[j].\mathrm{len} < i \leq j$.

A MEM cannot start at $P[i+1]$ when $P[i] = T[\mathrm{MS}[i+1].\mathrm{pos}-1]$, so we can safely wait to perform our next LCP query until we reach a character $P[i]$ with both $i \leq j - d + \mathrm{MS}[j].\mathrm{len}$ and $P[i] \neq T[\mathrm{MS}[i+1].\mathrm{pos}-1]$. At that point, we compute

$$\mathrm{MS}[i+1].\mathrm{len} = \mathrm{LCP}\left(P[i+1..m], T\left[\mathrm{MS}[i+1].\mathrm{pos}..n\right]\right) .$$

If $\mathrm{MS}[i+1].\mathrm{len} < d$ then, again, we need not perform another LCP query until we reach $P[i-d+\mathrm{MS}[i+1].\mathrm{len}]$. Otherwise, we compute

$$\mathrm{MS}[h+1].\mathrm{len} = \mathrm{LCP}(P[h+1..m], T[\mathrm{MS}[h+1].\mathrm{pos}..n])$$

whenever $P[h] \neq T[\mathrm{MS}[h+1].\mathrm{pos}]$ until one of those LCP queries returns a value less than d.

This algorithm can be summed up simply as follows: we perform LCP queries only when we reach characters $P[i] \neq T[\mathrm{MS}[i+1].\mathrm{pos}]$; an LCP query that returns a value less than d gives us a lower bound on how long we can safely wait before performing another query; an LCP query that returns a value at least d tells us to perform the next query as well.

We note that d can be given at query time, and even modified during the query. For example, if we keep d equal to the length of the longest match we have found so far, then we will find all the LCSs of P and T.

4.3 Experiments

We ran all our experiments on a server[1] with an Intel(R) Xeon(R) Gold 6248R CPU running at 3.00 GHz with 24 cores and 1.5TB of memory. For P we used the 600 MB concatenation of ten distinct copies of chromosome 19 from the 1000 Genomes dataset, and for T we used the 60 GB concatenation of 1000 other distinct copies of chromosome 19 from the same dataset.

We first modified Martínez-Guardiola et al.'s `Aug-1` index (their most competitive version) to use our subsampling from Subsection 4.1 with $s = 1$ (no subsampling), 2, 5 and 10, and computed the matching statistics of P with respect to T with each version of the index. The versions of the index occupied 850, 749, 639 and 594 MB — all at most 1.5% the size of the uncompressed dataset — and took 1650.78, 1722.51, 1755.75 and 2280.10 seconds, respectively, to compute the matching statistics. This means, for example, that with $s = 5$ the index took about three quarters as much space as without subsampling and used only 6% more query time.

We then further modified Martínez-Guardiola et al.'s index (with subsampling parameter $s = 5$) to use LCPs instead of LCEs and to find LCSs with our algorithm from Subsection 4.2. We found the LCSs of P and T in 1155.37 seconds, which is significantly faster — $1155.37/1650.78 < 70\%$ — than computing the LCSs by computing the matching statistics with any version of the index. As far as we know, this is the fastest way to compute LCSs in comparably compressed space.

Acknowledgements

This research was supported by the European Union's Horizon 2020 research and innovation program under Marie Skłodowska-Curie grant agreement No 956229 (ALPACA). It was also supported by grants 1/0463/20 and 1/0538/22 from the Scientific

[1]This server is part of the Advanced Research Computing at Hopkins (ARCH) core facility (rockfish.jhu.edu), supported by NSF grant OAC 1920103.

Grant Agency of the Ministry of Education, Science, Research, and Sport of the Slovak Republic and Slovak Academy of Sciences (VEGA) and grant APVV-22-0143 from the Slovak Research and Development Agency. L.D. funded by the PhD Fellowship FR (1117322N) by the Research Foundation - Flanders (FWO). T.G. and G.N. also funded by Basal Funds FB0001, ANID, Chile.

References

[1] Paolo Ferragina and Giovanni Manzini, "Indexing compressed text," *Journal of the ACM*, 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] 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.

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

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

[8] Heng Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM," *arXiv.org*, 2013.

[9] 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.

[10] 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.

[11] Omar Y Ahmed, Massimiliano Rossi, Travis Gagie, Christina Boucher, and Ben Langmead, "SPUMONI 2: improved classification using a pangenome index of minimizer digests," *Genome Biology*, vol. 24, no. 1, pp. 122, 2023.

[12] Vikram S Shivakumar, Omar Y Ahmed, Sam Kovaka, Mohsen Zakeri, and Ben Langmead, "Sigmoni: classification of nanopore signal with a compressed pangenome index," *bioRxiv.org*, 2023.

[13] Takaaki Nishimoto and Yasuo Tabei, "Optimal-time queries on BWT-runs compressed indexes," in *Proc. ICALP*, 2021.

[14] Philip Bille, Anders Roy Christiansen, Patrick Hagge Cording, and Inge Li Gørtz, "Finger search in grammar-compressed strings," *Theory of Computing Systems*, vol. 62, pp. 1715–1735, 2018.

[15] Tomohiro I, "Longest common extensions with recompression," in *Proc. CPM*, 2017.

[16] Artur Jeż, "Approximation of grammar-based compression via recompression," *Theoretical Computer Science*, vol. 592, pp. 115–134, 2015.

[17] Moses Ganardi, Artur Jeż, and Markus Lohrey, "Balancing straight-line programs," *Journal of the ACM*, vol. 68, no. 4, pp. 1–40, 2021.

[18] César Martínez-Guardiola, Nathaniel K Brown, Fernando Silva-Coira, Dominik Köppl, Travis Gagie, and Susana Ladra, "Augmented thresholds for moni," in *Proc. DCC*, 2023.

[19] Andrej Baláz, Travis Gagie, Adrián Goga, Simon Heumos, Gonzalo Navarro, Alessia Petescia, and Jouni Sirén, "Wheeler maps," *arXiv.org*, 2023.

[20] Dustin Cobas, Travis Gagie, and Gonzalo Navarro, "A fast and small subsampled r-index," in *Proc. CPM*, 2021.

[21] Peter Weiner, "Linear pattern matching algorithms," in *Proc. SWAT*, 1973.

[22] Philip Bille, Inge Li Gørtz, Patrick Hagge Cording, Benjamin Sach, Hjalte Wedel Vildhøj, and Søren Vind, "Fingerprints in compressed strings," *Journal of Computer and System Sciences*, vol. 86, pp. 171–180, 2017.

[23] Juha Kärkkäinen, Giovanni Manzini, and Simon J. Puglisi, "Permuted longest-common-prefix array," in *Proc. CPM*, 2009.