Scalable PBWT Queries with Minimum-Length SMEM Constraints

Uwaise Ibna Islam[1], Davide Cozzi[2], Travis Gagie[3], Rahul Varki[1], Vincenza Colonna[4,5,6], Erik Garrison[4,5,6], Paola Bonizzoni[2], and Christina Boucher[1]

[1] Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL, USA
[2] Department of Informatics, Systems and Communication, University of Milano–Bicocca, Milan, Italy
[3] Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada
[4] Dept of Genetics, Genomics and Informatics, UTHSC, Memphis, TN, USA
[5] Institute of Genetics and Biophysics, National Research Council, Naples, 80111, Italy
[6] Dept of Pediatrics, UTHSC, Memphis, TN, USA
u.islam@ufl.edu, christinaboucher@ufl.edu

Abstract. Large, phase-resolved haplotype panels—now emerging from efforts such as UK Biobank, TOPMed, All of Us, and the Mexican Biobank—enable fine-grained analyses of admixture, local ancestry, and imputation. The Positional Burrows–Wheeler Transform (PBWT) is a natural index for these data, supporting efficient phase-aware matching, but current methods typically enumerate all set-maximal exact matches (SMEMs), incurring unnecessary computation when only long matches are of interest. We introduce PBML, a space- and time-efficient algorithm that enumerates all SMEMs of length at least L. Our approach adapts Boyer–Moore style skipping with a forward–backward traversal of the PBWT, yielding an implementation that avoids auxiliary data structures while restricting search to informative long matches. We evaluate PBML under two querying strategies on two haplotype resources. On the 1000 Genomes Project (1KGP), PBML achieves up to 38.1% faster index construction than μ-PBWT, 61.7% faster single-threaded queries (with 22% higher peak memory), and a 305.7% multithreaded speedup (with only 9.4% higher peak memory), demonstrating strong scalability at population scale. We further show that, when targeting long SMEMs ($L \geq 10{,}000$) on 1KGP, PBML reduces total runtime by 73.3% and peak memory usage by 20% relative to $L = 1$, while still reporting over 1.15 million long SMEMs genome-wide. Finally, we apply PBML to a biobank from the Tennessee Biorepository and Integrative Genomics (BIG) Initiative and show that, on chromosome 22, setting L between 500 and 1,000 preserves at least 95% single-query coverage (97–99.8% for batches of 5–20 queries) while reducing SMEM counts by 70–83.8% and cutting query time by 37.6%, with speedups up to 92.9% when $L = 5{,}000$ and only very long tracts are of interest. Together, these results establish PBML as an efficient and scalable tool to query PBWT-based haplotypes when biologically meaningful matches are the target.

1 Introduction

Preserving phase at the chromosome scale, haplotype-resolved assemblies allow precise mapping of admixture tracts [1,2], elucidation of disease genetics [3,4], estimation of homozygosity runs [5], and inference of local ancestry [6]. Given this, there are several ongoing projects that aim to collect large-scale haplotype-resolved assemblies, including the UK Biobank [7], the Mexican Biobank [8], TOPMed [9], and All of Us [10]. Collecting phased assemblies at scale creates shared haplotype panels that increase statistical power, enhance imputation accuracy, and support method development across diverse populations. To index and query these large panels efficiently, the Positional Burrows–Wheeler Transform (PBWT) provides an efficient phase-aware representation. Originally proposed by Durbin [11], the PBWT stores a set of h sequences that have w variation sites in a $h \times w$ binary matrix $S[1 \dots h][1 \dots w]$, where the rows are sorted in colexicographic order. For a given haplotype panel, it supports efficient querying and haplotype matching in $\mathcal{O}(hw)$ time. This has resulted in the application of PBWT in haplotype phasing [12,13], haplotype estimation [14], genotype imputation [15], genome database search [16], inferring ancestry [17] and Identity-By-Descent (IBD) segment detection [18,19,20,8].

Given the centrality of PBWT in haplotype indexing and matching, recent efforts have focused on compressing the structure, extending it in several ways, and accelerating the support for various queries. In this context, the trajectory of the advances is as follows. In 2017, Novak et al. [16] first generalized PBWT beyond linear references by encoding haplotypes on genome graphs, enabling efficient sub-haplotype matching across complex variation. Building on this, mPBWT [21] extended the framework to handle multi-allelic sites while keeping the time overhead close to the bi-allelic case, preserving PBWT's streaming, column-wise updates. GBWT [22] then targeted scalability by organizing paths with node-local structures, supports incremental construction, and exploits parallel merging, making very large haplotype panels tractable. Bi-directional PBWT [23] introduced coordinated forward and reverse sweeps, which sharpen long-range haplotype-block matching and help flag recombination or phasing errors. Syllable-PBWT [24] compresses panels by grouping haplotypes into syllables and uses polynomial hashing to accelerate long-match queries, reducing both memory and lookup time in practice. In parallel, μ-PBWT [25] applies run-length encoding to PBWT columns and enables efficient set maximal exact-match (SMEM) queries by computing and using matching statistics. SMEMs are the longest position-restricted substrings shared between an external query sequence and any other sequence of the same length that is represented in PBWT. Lastly, in 2025 Shakya et al. [26] devised a method to support updates to the PBWT without full reconstruction, trading additional memory for online insertions and maintenance.

In the PBWT, SMEMs mark the longest contiguous segments where a query haplotype is exactly identical to one or more references, and therefore capture shared inheritance, which is fundamental to several key applications. In genotype imputation, for example, SMEMs delineate regions of exact agreement between the query and selected donors, allowing missing or low-quality alleles to be inferred from the donors' consensus; the segment length provides a natural confidence signal, and the endpoints indicate where agreement breaks down. For IBD and local-ancestry inference, the distribution of SMEM lengths reflects the recency of shared ancestry (longer segments imply more recent common ancestors), and SMEM boundaries closely track historical recombination breakpoints, supporting precise, piecewise-constant ancestry labeling when aggregated across ancestry-annotated reference haplotypes. The seminal PBWT includes an algorithm for locating SMEM [11], and subsequent work has optimized SMEM discovery using run-length encoding, sampling, and parallelism to achieve fast, memory-efficient haplotype matching [25,26]. A remaining challenge is to restrict searches to SMEMs above a minimum length, as enumerating all SMEMs unnecessary increases both discovery time and downstream post-processing [27,28].

We introduce PBML, a space- and time-efficient procedure to enumerate all SMEMs whose length is at least a parameter L given at query time. When $L = 1$, the task reduces to the full (unrestricted) SMEM enumeration problem studied by Cozzi et al. [25]. In this case PBML is essentially just an adaptation to finding SMEMs with the PBWT, of Li's [29] forward-backward algorithm for finding MEMs with the BWT. Surprisingly, in our experiments it still turned out to be noticeably faster than Cozzi et al.'s approach. For larger values of L, PBML repeatedly skips to the right L values and then matches to the left, somewhat like the classic Boyer-Moore[30] string-matching algorithm from 1977. PBML skips right in the query haplotype and performs indexed pattern matching, however, while Boyer-Moore would skip right in the dataset and perform online pattern matching. Gagie [31] first proposed this idea for finding MEMs with the BWT, using

sophisticated data structures that support both longest common prefix and longest common suffix queries. Gagie later [27] simplified it to use the same data structures as Li's forward-backward algorithm and named it Boyer-Moore-Li (BML). Li [32] implemented BML in ropebwt3. With PBML, we extend BML to the PBWT for efficient haplotype indexing and querying.

We evaluate PBML using two querying strategies and two haplotype datasets to elucidate the trade-offs between query latency and peak memory usage. First, we show the efficiency of our method in identifying all SMEMs in the 1000 Genomes Project (1KGP) dataset, and compare our performance to μ-PBWT [25]. PBML outperforms μ-PBWT with up to 38.1% faster index construction, 61.7% faster single-threaded queries (using 22% more memory), and an impressive 305.7% multi-threaded speedup (only 9.4% more memory), demonstrating its efficiency and scalability for large-scale haplotype analysis. Next, we demonstrate our method for finding long SMEMs ($L \geq 10,000$) in the 1KGP dataset. We show that PBML reduces total runtime by 73.3% and peak memory usage by 20% relative to $L = 1$, while still identifying over 1.15 million long SMEMs. Lastly, we apply our method to the Biobank dataset from the Tennessee Biorepository and Integrative Genomics (BIG) Initiative, a multi-institution resource curated from a diverse Mid-South population, including African American participants from Memphis [33]. We demonstrate that PBML recovers long contiguous regions of shared local ancestry in this cohort: on chromosome 22, setting L between 500 and 1,000 preserves ≥95% single-query coverage (97–99.8% for 5–20 queries) while reducing the SMEM count by 70–83.8% and cutting query time by 37.6%, with even larger speedups (92.9%) at $L = 5,000$ when exclusively very long tracts are desired. The source code for PBML is publicly available at https://github.com/uwaiseibna/PBML.git.

2 Preliminaries

Representation of haplotypes. We represent phased haplotype data as a binary matrix $S \in \{0,1\}^{h \times w}$, where each of the h rows corresponds to a haplotype sequence and each of the w columns corresponds to a biallelic variant site ordered along the genome. An entry $S_{i,j}$ takes the value 0 if the haplotype i carries the reference allele at the site j, and 1 if it carries the alternate allele. Thus, each row encodes a haplotype as a sequence of binary states across sites, while each column reflects the allelic distribution at a particular site across the haplotype panel.

The positional Burrows–Wheeler transform (PBWT). The PBWT [11] is a data structure to efficiently store and query haplotype panels. We also refer to this as the *(forward) PBWT* for reasons that become clear in the next section. Given the haplotype panel $S \in \{0,1\}^{h \times w}$, the PBWT of S is another matrix $P \in \{0,1\}^{h \times w}$. For each column $j \in \{1, \ldots, w\}$, it maintains two arrays: the prefix array a_j and the divergence array d_j. In this work, we do not use the divergence array and, thus, restrict interest to the the prefix array a_j.

The prefix array a_j represents the ordering of the haplotypes in S induced by the co-lexicographic order of their prefixes up to column $j-1$, i.e., formally $a_j[i]$ is the index of the ith haplotype in the list of prefixes $\{S_{k,1..j-1} : 1 \leq k \leq h\}$ sorted co-lexicographically. The first column of P is identical to that of S, i.e. $P_{i,1} = S_{i,1}$ for all $1 \leq i \leq h$. For $j > 1$, the jth column of P is obtained by stably permuting the rows of $S_{*,j}$ according to the prefix order a_{j-1}. Equivalently, if we denote the jth column of S by $S_{*,j}$, then

$$P_{*,1} = S_{*,1}, \qquad P_{*,j} = S_{*,j}[a_{j-1}] \quad \text{for all } j \geq 2.$$

The reverse PBWT. In addition to this *(forward) PBWT*, we define another matrix *reverse PBWT*, $P^{rev} \in \{0,1\}^{h \times w}$. Here, the last column of the transformed matrix is identical to the last column of S, i.e., $P^{\text{rev}}_{*,w} = S_{*,w}$, and each preceding column is obtained by stably sorting haplotypes in co-lexicographic order with respect to their suffixes starting from that column. Formally, the ordering at column j is induced by sorting the suffixes $\{S_{k,j..w} : 1 \leq k \leq h\}$ co-lexicographically. We refer to this suffix-based construction as the *reverse PBWT*.

Set Maximal Exact Matches (SMEMs). Given this haplotype panel $S \in \{0,1\}^{h \times w}$ and a binary query sequence $Q \in \{0,1\}^{w}$, a *Set Maximal Exact Match (SMEM)* is defined as a substring $Q[i..j]$ with $1 \leq i \leq j \leq w$ such that:

1. There exists at least one haplotype $s \in S$ containing $Q[i..j]$ as a substring, i.e., $s[k..k+(j-i)] = Q[i..j]$ for some $1 \leq k \leq w-(j-i)$.
2. $Q[i..j]$ cannot be extended to the left, i.e. $Q[i-1..j] \neq s[k-1..k+(j-i)]$ for all $s \in S$ (when $i > 1$), nor extended to the right, i.e. $Q[i..j+1] \neq s[k..k+(j-i)+1]$ for all $s \in S$ (when $j < w$).
3. $Q[i..j]$ is not strictly contained in any longer exact match involving another sequence S.

$h \backslash w$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	0	0	1	0	0	0	0	0	0	0	1	1	0	1
1	1	0	0	1	1	0	0	1	0	0	0	0	0	1	1
2	1	0	0	1	1	0	0	1	0	0	0	1	0	0	1
3	1	0	0	1	1	0	0	1	0	0	0	1	0	0	1
4	0	1	0	1	0	1	0	0	0	0	0	1	0	0	1
5	0	1	0	1	0	1	0	0	0	0	0	1	0	0	1
6	0	1	0	1	0	1	0	0	0	0	0	1	0	0	1
7	0	1	0	1	0	1	0	0	0	0	0	0	1	0	1
8	0	1	0	0	1	0	0	0	0	1	1	1	0	0	1
9	0	1	0	1	0	0	0	0	1	0	0	0	0	1	1
10	0	1	0	1	0	0	0	0	1	0	0	0	0	1	1
11	0	1	0	0	1	0	0	0	0	0	1	1	0	0	0
12	0	1	0	0	1	0	0	0	1	0	1	1	0	0	1
13	0	1	0	0	1	0	0	0	1	0	1	1	0	0	1
14	0	1	0	0	0	0	0	0	1	0	0	0	1	0	1
15	0	1	0	0	0	0	0	0	1	0	0	0	1	0	1
16	0	1	0	1	0	0	0	0	0	0	0	1	1	0	1
17	1	1	0	0	0	1	0	0	0	0	0	1	1	0	1
18	0	1	1	0	1	0	0	0	0	0	0	1	0	0	1
19	0	1	1	0	1	0	1	0	0	0	0	0	1	0	1
Q	0	1	0	0	1	0	1	0	0	0	1	1	1	0	1

Table 1: Matrix S with 20 haplotypes and 15 variable sites where the SMEMs for query Q are highlighted in yellow.

3 Methods

In this section, we present our algorithm that, given a query haplotype and a haplotype panel, reports all SMEMs of length at least L and the panel haplotypes containing each one. To do this, we first show how to support longest common prefix (LCP) and longest common suffix (LCS) queries on the forward and reverse PBWT. We then adapt BML to operate directly on the PBWT using LCP and LCS queries to perform right-to-left and conditional left-to-right traversals that enumerate all SMEMs and their corresponding PBWT intervals. Finally, to recover the haplotypes contained in these intervals without storing full prefix arrays, we combine the Toehold Lemma [34,35] with the ϕ data structure.

3.1 LCP and LCS Queries on the PBWT

BML requires left-to-right and right-to-left matches which are realized by LCP queries on the forward PBWT and LCS queries on the reverse PBWT. Here, we begin by describing the former.

We store each column of the forward PBWT as a compressed sparse bitvector that supports constant-time rank queries and store the number of zeros in each column in a *zero_counts* array. Next, we let $Q[0\,..\,m-1]$ be the query haplotype. During an LCP query, we maintain the PBWT interval of haplotypes matching the current prefix of Q and extend this interval character-by-character in the forward direction. Formally, we let $[s, e]$ denote the current interval for PBWT column *col*, where the interval represents all haplotypes matching the current prefix. At the start of an LCP query, this interval is $[0, n]$, which means that all haplotypes are contained by the initial interval. We let $\text{rank}_1(col, x)$ return the number of ones in positions $[0, x)$ of column *col*, and let $zero_counts[col]$ be the total number of zeros in *col*. For a query bit $Q_{col} \in \{0, 1\}$, we compute the next interval $[s', e']$ as follows:

$$\text{if } Q_{col} = 0: \quad s' = s - \text{rank}_1(col, s),$$
$$e' = (e+1) - \text{rank}_1(col, e+1) - 1,$$

$$\text{if } Q_{col} = 1: \quad s' = zero_counts[col] + \text{rank}_1(col, s),$$
$$e' = zero_counts[col] + \text{rank}_1(col, e+1) - 1.$$

If $s' > e'$, the interval becomes empty and the procedure ends. Otherwise, the interval is updated to $[s', e']$, the match length is incremented, and we proceed to the next column. This process continues until either the interval becomes empty or we reach the end of the PBWT.

LCS queries apply the same procedure on the reverse PBWT, processing columns from right to left, and storing the reverse PBWT in an analogous manner as the forward PBWT. Both the LCP and LCS queries return the interval in the PBWT containing all haplotypes that share this longest prefix or longest suffix with the query.

3.2 Boyer-Moore-Li in the PBWT

We assume as input a query haplotype $Q[0\,..\,m-1]$ and a minimum length $L \geq 1$. We now describe how to compute all SMEMs of length at least L between the PBWT and the query haplotype Q. Any SMEM of length at least L whose start lies in $Q[0\,..\,L-1]$ must include position $Q[L-1]$. Consequently, we initiate the search at $Q[L-1]$ and match from right-to-left, updating the interval at each step until it becomes empty at position $Q[i]$. At this point, there are two cases depending on the position i at which the match ends.

Case $i = 0$: The match reaches the beginning of Q. We then match from left-to-right from $Q[0]$ until the interval becomes empty at position r, producing the first SMEM $Q[0\,..\,r-1]$. After reporting an SMEM ending at position $r-1$, BML finds the next SMEM by verifying whether it covers position r. We perform a right-to-left match from $Q[r]$ of length ℓ before the interval becomes empty. If $\ell \geq L$, we perform the corresponding left-to-right match to obtain the next SMEM and proceed. If $\ell < L$ then it follows that no SMEM of length at least L covers $Q[r]$. Since the first $L-\ell$ positions starting at r cannot begin an SMEM of length L, we recurse on the suffix $Q[r-L+\ell+1\,..\,m-1]$.

Case $i \geq 1$: No SMEM of length at least L begins in $Q[0\,..\,i-1]$. We recurse on the suffix $Q[i\,..\,m-1]$, starting at $Q[i+L-1]$.

The recursion ends once the remaining suffix has length $< L$. When $L = 1$, the procedure is simplified to the standard forward–backward algorithm.

The left-to-right and right-to-left matches in BML are then realized by LCP queries on the forward PBWT and LCS queries on the reverse PBWT. We described how to support these queries in Subsection 3.1.

3.3 Identifying Haplotypes Containing a SMEM

To report the actual matching haplotypes in a given interval $[s, e]$ returned by BML, we require access to the prefix array at each column. However, storing the complete prefix array for all columns is memory-prohibitive for large panels. Instead, PBML uses a similar approach as the r-index to support the finding of haplotypes in the PBWT in a manner that eliminates the necessity of storing the full prefix arrays. Briefly, the approach taken from the r-index is to use the Toehold Lemma to store one of the suffix array entries corresponding to a given interval in the BWT, and then all the entries in the suffix array are obtained by recursively applying the ϕ data structure. This allows a single suffix array entry—either the first or last—to be stored rather than the complete suffix array, while still allowing all entries to be recovered. Here, we will apply the same approach to the PBWT.

The Toehold Lemma for the PBWT. Using the *Toehold Lemma* [36], PBML maintains a representative haplotype from each nonempty interval throughout an LCP query. More precisely, for a given column *col* and interval end position e in the forward PBWT, we track the haplotype at row e as the interval is extended to the next column. If the bit at position e matches the query bit, we keep the same representative haplotype. Otherwise, the representative moves to the haplotype that ends the run immediately preceding the one containing current haplotype at row e in the updated interval of the PBWT permutation. This new representative haplotype is obtained via a ϕ query on the haplotype starting the run containing haplotype at position e, which we describe next.

The ϕ query operation on the PBWT. We get the haplotype at the end of an interval reported from LCP querying using the Toehold Lemma. However, we need the ϕ query both inside the Toehold Lemma and to enumerate the remaining haplotypes in the interval, as we now describe.

To support ϕ-queries, each haplotype maintains a *successor data structure* that stores: (i) the index of PBWT columns where the haplotype begins a run; (ii) the haplotype immediately preceding it in the prefix array at each such run start; and (iii) a pointer into that predecessor's successor array indicating the closest earlier column where the predecessor also began a run. For each PBWT run, we store: (i) its starting row offset in its column; (ii) the prefix-array index at the run start; and (iii) a pointer to the successor data structure with the haplotype that started the run and the corresponding index in that successor data structure. Finally, for each column, we store the index of the first run that begins in that column, allowing direct access from a column to its run-sampled data structures.

Building on this, ϕ allows us to traverse the implicit prefix array for an interval returned by LCP query. We get the interval ending haplotype, h from the Toehold lemma, and $\phi(h)$ returns the haplotype that precedes h in the prefix array (i.e., its co-lexicographic predecessor). By repeatedly applying ϕ starting from the haplotype h, PBML get all haplotypes in the interval and report SMEMs without ever storing the full prefix arrays. The detailed algorithm for run-based haplotype retrieval is provided in the appendix.

We conclude by making a note on the implementation of the ϕ query operation. There has been a significant amount of work in storing data structures for ϕ in a space- and time- efficient manner. Nishimoto and Tabei [37] showed how to support ϕ queries on the r-index in constant worst-case time through a careful run-splitting technique. Later, Zakeri et al [38] demonstrated that this splitting is not necessary in practice and provided a practical implementation based on table lookup. We adapt this approach to support ϕ queries on the PBWT, enabling sequential access to prefix arrays without materializing them explicitly.

3.4 PBML Algorithm

Algorithm 1 outlines the complete PBML querying procedure. Given a query sequence $Q[0\,..\,m-1]$, minimum SMEM length L, forward and reverse PBWTs, and the successor and run-sampled data structures for ϕ query operation, PBML enumerates all SMEMs of length at least L.

The algorithm maintains a position *left* initialized to 0. At each iteration, we set $col = left + L - 1$, the earliest position in which an SMEM of length L starting at *left* could end. We compute the longest common suffix (LCS) ending at *col* using the reverse PBWT, which gives the length ℓ. If $\ell < L$, no SMEM of length L can cover *col*, so we advance *left* by $L - \ell$ positions (as described in Section 3.1) and continue.

Otherwise, we compute the longest common prefix (LCP) starting at position $col - \ell + 1$ using the forward PBWT, producing length ℓ' and an interval $[s, e]$ of matching haplotypes. If $\ell' \geq L$, we report the SMEM $Q[col - \ell + 1\,..\,col - \ell + \ell']$ for all haplotypes in the interval. We retrieve the original haplotype indices from the interval using the Toehold Lemma and ϕ as described in Sections 3.3. Finally, we advance *left* to $(col - \ell + \ell') - L + 2$ to begin searching for the next SMEM after the current one.

Algorithm 1: PBML querying algorithm

```
Input: Query Q[0..m−1], minimum length L, forward PBWT, reverse PBWT
Output: Set of SMEMs of length ≥ L
left ← 0;
while left + L − 1 < m do
    col ← left + L − 1;
    (ℓ, s, e, h) ← LCS(Q, col) using reverse PBWT;
    if ℓ < L then
        left ← left + (L − ℓ);
        continue;
    start ← col − ℓ + 1;
    (ℓ′, s′, e′, h′) ← LCP(Q, start) using forward PBWT;
    if ℓ′ ≥ L then
        end ← start + ℓ′ − 1;
        Retrieve haplotype indices in interval [s′, e′] using toehold lemma and φ queries;
        Report SMEM Q[start..end] for all retrieved haplotypes;
    left ← (start + ℓ′) − L + 2;
```

3.5 Example: Finding SMEMs with $L = 4$

We demonstrate the PBML algorithm on matrix S on Table 1 using the forward and reverse PBWTs shown in Table 2, with minimum SMEM length $L = 4$.

First SMEM. We begin by computing an LCS on the reverse PBWT (Table 2b) at column $C3$, following Algorithm 1. Starting with all haplotypes in the interval $[0, 19]$, we iteratively update the interval extending backward to column $C0$. This results in an LCS from $C3 \rightarrow C0$ of length ≥ 4. The corresponding forward search begins with an LCP on the forward PBWT (Table 2a) at column $C0$ with interval $[0, 19]$, and continues until the interval becomes empty at column $C5$. At this point, haplotypes $[8, 11, 12, 13]$ are reported from the final interval $[11, 14]$, with $\text{LCP}(C0) = C0 \rightarrow C5 = 6$.

Second SMEM. Next, we compute an LCS at column $C6$ on the reverse PBWT, immediately after the end of the previous SMEM. This yields an LCS from $C6 \rightarrow C3 \geq 4$. The corresponding LCP begins at column $C3$ and extends to $C9$, until the interval becomes empty at $[10, 10]$. An SMEM is reported for haplotype 19, with $\text{LCP}(C3) = C3 \rightarrow C9 = 7$.

Third SMEM. The next LCS is computed at column $C10$ on the reverse PBWT, producing $C10 \rightarrow C7 \geq 4$. The corresponding LCP starts at $C7$ and extends to $C11$, until the interval becomes empty at $[16, 16]$. An SMEM is reported for haplotype 11, with $\text{LCP}(C7) = C7 \rightarrow C11 = 5$.

Failed LCS. We then attempt an LCS at column $C12$ on the reverse PBWT. This yields $C13 \rightarrow C11 < 4$, which is below the minimum SMEM length and therefore invalid. Such cases are highlighted in lavender and have no corresponding forward LCP.

Fourth SMEM. Finally, we continue with an LCS at column $C14$ based on the partial match length, which produces $C14 \rightarrow C11 \geq 4$. The corresponding LCP starts at $C11$ and extends to $C14$, reaching the end of the PBWT. The final interval $[14, 16]$ reports haplotypes $[0, 16, 17]$, with $\text{LCP}(C11) = C11 \rightarrow C14 = 4$. This terminates the PBML procedure.

4 Results

4.1 Experimental setup

PBML is implemented in `C++17` using the Succinct Data Structure Library (SDSL) [39] for compressed bitvector storage with rank support, enabling memory-efficient PBWT operations. OpenMP provides multi-threaded query processing, and htslib [40] handles VCF/BCF file parsing. We ran experiments on a machine with an AMD EPYC 75F3 32-Core Processor, 503 GB RAM, and 8 GB swap, running Red Hat Enterprise Linux 9 (64-bit, kernel 5.14.0) with compiler g++ version 11.5.0. The running time and maximum resident set size were measured using `/usr/bin/time -v`.

We evaluated PBML against μ-PBWT and Durbin's Algorithm 5 (`PBWT`) at $L = 1$. For each method, we recorded construction time, peak memory usage, and query time. Because neither μ-PBWT nor `PBWT` implements SMEM queries with a minimum-length constraint ($L > 1$), we report results for PBML alone in that setting. Although Shakya et al. [26] support SMEM queries, we do not include their method in our comparison because it supports dynamic-PBWT. As reported by the authors, dynamic maintenance incurs substantial overhead, with peak memory usage up to 33× higher than μ-PBWT, rendering a direct comparison inapposite. All other existing PBWT are unable to support SMEM finding.

4.2 Datasets

We evaluated PBML on two datasets. The first dataset consists of chromosome panels from the 1000 Genomes Project Phase 3. VCF files were obtained from the public repository (https://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/) and pre-processed to retain only biallelic SNPs using `bcftools view -m2 -M2 -v snps` [41]. Each chromosome panel contains 5,008 haplotypes with biallelic site counts ranging from approximately 1 million to 6 million variants per chromosome.

Row	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
0	1 (0)	1 (4)	0 (0)	1 (0)	1 (8)	0 (14)	0 (14)	0 (14)	1 (14)	0 (0)	0 (0)	1 (0)	1 (7)	1 (1)	1 (18)
1	1 (1)	1 (5)	0 (1)	1 (1)	1 (11)	0 (15)	0 (15)	0 (15)	1 (15)	0 (16)	0 (16)	1 (16)	1 (19)	1 (9)	1 (4)
2	1 (2)	1 (6)	0 (2)	1 (2)	1 (12)	1 (17)	0 (0)	0 (0)	0 (0)	1 (8)	1 (11)	1 (18)	0 (1)	1 (10)	1 (5)
3	1 (3)	1 (7)	0 (3)	1 (3)	1 (13)	0 (0)	0 (9)	0 (9)	1 (9)	0 (11)	0 (18)	1 (17)	1 (14)	0 (18)	1 (6)
4	0 (4)	1 (8)	0 (4)	1 (4)	0 (14)	1 (4)	0 (10)	0 (10)	1 (10)	0 (18)	0 (17)	1 (4)	1 (15)	0 (4)	1 (2)
5	0 (5)	1 (9)	0 (5)	1 (5)	0 (15)	1 (5)	0 (16)	0 (16)	0 (16)	0 (17)	0 (4)	1 (5)	0 (9)	0 (5)	1 (3)
6	0 (6)	1 (10)	0 (6)	1 (6)	0 (17)	1 (6)	0 (8)	0 (8)	0 (8)	0 (4)	0 (5)	1 (6)	0 (10)	0 (6)	0 (11)
7	0 (7)	1 (11)	0 (7)	1 (7)	1 (18)	1 (7)	0 (11)	0 (11)	0 (11)	0 (5)	0 (6)	0 (7)	1 (0)	0 (2)	1 (12)
8	0 (8)	1 (12)	0 (8)	0 (8)	1 (19)	0 (9)	0 (12)	0 (12)	1 (12)	0 (6)	0 (7)	0 (19)	1 (16)	0 (3)	1 (13)
9	0 (9)	1 (13)	0 (9)	1 (9)	0 (0)	0 (10)	0 (13)	0 (13)	1 (13)	0 (7)	0 (19)	0 (1)	0 (18)	0 (11)	1 (8)
10	0 (10)	1 (14)	0 (10)	1 (10)	1 (1)	0 (16)	0 (18)	0 (18)	0 (18)	0 (19)	0 (1)	1 (2)	1 (17)	0 (12)	1 (7)
11	0 (11)	1 (15)	0 (11)	0 (11)	1 (2)	0 (8)	1 (19)	1 (1)	0 (17)	0 (1)	0 (2)	1 (3)	0 (4)	0 (13)	1 (19)
12	0 (12)	1 (16)	0 (12)	0 (12)	1 (3)	0 (11)	0 (1)	1 (2)	0 (4)	0 (2)	0 (3)	0 (14)	0 (5)	0 (8)	1 (14)
13	0 (13)	1 (18)	0 (13)	0 (13)	0 (4)	0 (12)	0 (2)	1 (3)	0 (5)	0 (3)	0 (14)	0 (15)	0 (6)	0 (7)	1 (15)
14	0 (14)	1 (19)	0 (14)	0 (14)	0 (5)	0 (13)	0 (3)	0 (17)	0 (6)	0 (14)	0 (15)	0 (9)	0 (2)	0 (19)	1 (0)
15	0 (15)	0 (0)	0 (15)	0 (15)	0 (6)	0 (18)	0 (17)	0 (4)	0 (7)	0 (15)	0 (9)	0 (10)	0 (3)	0 (14)	1 (16)
16	0 (16)	0 (1)	0 (16)	1 (16)	0 (7)	0 (19)	0 (4)	0 (5)	0 (19)	0 (9)	0 (10)	1 (11)	0 (11)	0 (15)	1 (17)
17	1 (17)	0 (2)	1 (18)	0 (17)	0 (9)	0 (1)	0 (5)	0 (6)	0 (1)	0 (10)	1 (12)	1 (12)	0 (12)	0 (0)	1 (1)
18	0 (18)	0 (3)	1 (19)	0 (18)	0 (10)	0 (2)	0 (6)	0 (7)	0 (2)	0 (12)	1 (13)	1 (13)	0 (13)	0 (16)	1 (9)
19	0 (19)	1 (17)	0 (17)	0 (19)	0 (16)	0 (3)	0 (7)	0 (19)	0 (3)	0 (13)	1 (8)	1 (8)	0 (8)	0 (17)	1 (10)
Q	0	1	0	0	1	0	1	0	0	0	1	1	1	0	1
`zero_counts`	15	4	18	9	11	15	19	17	14	19	16	7	13	17	1
`LCP`	6	-	-	7	-	-	-	5	-	-	-	4	-	-	-

(a) forward PBWT on Matrix S

Row	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
0	1 (0)	1 (14)	0 (14)	1 (0)	1 (18)	1 (7)	0 (7)	1 (1)	0 (1)	0 (1)	0 (1)	1 (11)	0 (11)	0 (11)	1 (0)
1	1 (1)	1 (15)	0 (15)	1 (16)	0 (0)	1 (4)	1 (19)	0 (7)	1 (9)	0 (9)	0 (9)	1 (2)	1 (0)	0 (0)	1 (1)
2	1 (2)	1 (17)	0 (17)	1 (9)	0 (16)	1 (5)	0 (4)	0 (19)	1 (10)	0 (10)	0 (10)	1 (3)	0 (2)	1 (1)	1 (2)
3	1 (3)	1 (11)	1 (18)	1 (10)	1 (11)	1 (6)	0 (5)	1 (2)	0 (7)	0 (7)	0 (7)	1 (4)	0 (3)	0 (2)	1 (3)
4	0 (14)	1 (8)	0 (11)	0 (14)	1 (8)	0 (18)	0 (6)	1 (3)	1 (14)	0 (14)	0 (14)	1 (5)	0 (4)	0 (3)	1 (4)
5	0 (15)	1 (12)	0 (8)	0 (15)	0 (9)	0 (0)	0 (18)	0 (4)	1 (15)	0 (15)	0 (15)	1 (6)	0 (5)	0 (4)	1 (5)
6	1 (17)	1 (13)	0 (12)	1 (7)	0 (10)	0 (16)	0 (0)	0 (5)	0 (19)	0 (19)	0 (19)	1 (8)	0 (6)	0 (5)	1 (6)
7	0 (11)	0 (0)	0 (13)	1 (4)	0 (14)	1 (17)	0 (16)	0 (6)	0 (2)	0 (2)	1 (11)	1 (12)	1 (7)	0 (6)	1 (7)
8	0 (8)	1 (16)	1 (19)	1 (5)	0 (15)	0 (11)	0 (17)	0 (18)	0 (3)	0 (3)	0 (2)	1 (13)	0 (8)	0 (7)	1 (8)
9	0 (12)	1 (9)	0 (0)	1 (6)	1 (12)	0 (8)	0 (11)	0 (0)	0 (4)	0 (4)	0 (3)	1 (18)	0 (12)	0 (8)	1 (9)
10	0 (13)	1 (10)	0 (16)	0 (17)	1 (13)	0 (9)	0 (8)	0 (16)	0 (5)	0 (5)	0 (4)	0 (1)	0 (13)	1 (9)	1 (10)
11	0 (16)	1 (7)	0 (9)	0 (18)	1 (1)	0 (10)	0 (9)	0 (17)	0 (6)	0 (6)	0 (5)	0 (9)	1 (14)	1 (10)	0 (11)
12	0 (9)	1 (4)	0 (10)	0 (11)	1 (2)	0 (14)	0 (10)	0 (11)	0 (18)	0 (18)	0 (6)	0 (10)	1 (15)	0 (12)	1 (12)
13	0 (10)	1 (5)	0 (7)	0 (8)	1 (3)	0 (15)	0 (14)	0 (8)	0 (0)	0 (0)	1 (8)	1 (0)	1 (16)	0 (13)	1 (13)
14	0 (7)	1 (6)	0 (4)	0 (12)	1 (19)	0 (12)	0 (15)	0 (9)	0 (16)	0 (16)	1 (12)	0 (7)	1 (17)	0 (14)	1 (14)
15	0 (4)	0 (1)	0 (5)	0 (13)	0 (7)	0 (13)	0 (12)	0 (10)	0 (17)	0 (17)	1 (13)	0 (14)	0 (18)	0 (15)	1 (15)
16	0 (5)	0 (2)	0 (6)	1 (1)	0 (4)	0 (1)	0 (13)	0 (14)	0 (11)	0 (11)	0 (18)	0 (15)	1 (19)	0 (16)	1 (16)
17	0 (6)	0 (3)	0 (1)	1 (2)	0 (5)	0 (2)	0 (1)	0 (15)	1 (12)	1 (8)	0 (0)	1 (16)	0 (1)	0 (17)	1 (17)
18	0 (18)	1 (18)	0 (2)	1 (3)	0 (6)	0 (3)	0 (2)	0 (12)	1 (13)	0 (12)	0 (16)	1 (17)	0 (9)	0 (18)	1 (18)
19	0 (19)	1 (19)	0 (3)	0 (19)	0 (17)	0 (19)	0 (3)	0 (13)	0 (8)	0 (13)	0 (17)	0 (19)	0 (10)	0 (19)	1 (19)
Q	0	1	0	0	1	0	1	0	0	0	1	1	1	0	1
`zero_counts`	15	4	18	9	11	15	19	17	14	19	16	7	13	17	1
`LCS`	-	-	-	4	-	-	4	-	-	-	4	-	2	-	4

(b) Reverse PBWT on Matrix S

Table 2: Forward and reverse PBWTs on matrix S. Yellow and cyan highlight consecutive intervals from each alternating LCP (forward PBWT) and LCS (reverse PBWT) queries during SMEM finding; each SMEM query yields a color-matched pair of consecutive intervals across the two matrices. Lavender marks backward LCS searches that return a match length smaller than the minimum SMEM length L, which therefore lack a corresponding forward LCP interval. Overlapping subintervals are shown with a 50% blend of the respective interval colors.

The second dataset comprises biallelic single-nucleotide variants (indels excluded) from the Biorepository and Integrative Genomics (BIG) Initiative biobank [33]. The BIG cohort is a multi-institutional resource assembled from a diverse mid-south population, with substantial representation of African American participants from Memphis. We extract chromosome-specific variant sites from 5,500 individuals and compile them into chromosome-level datasets. From the merged dataset, we select chromosome 22 for comparison with 1KGP data and phase variable sites using ShapeIT5 [42]. There are 893,511 loci in chromosome 22. After retaining only biallelic sites, we divide the dataset into a panel of 10,000 samples and a query set of 1,000 samples for use with PBML.

4.3 Enumerating SMEMs in the 1KGP dataset

Using the 1KGP dataset, we first compared PBML with μ-PBWT and PBWT using 1,000 randomly selected haplotypes as queries. The remaining 4,008 haplotypes were used to construct the reference panels for all autosomes. For querying, each of the 1,000 haplotypes was used to identify all SMEMs, allowing comparison with μ-PBWT and PBWT.

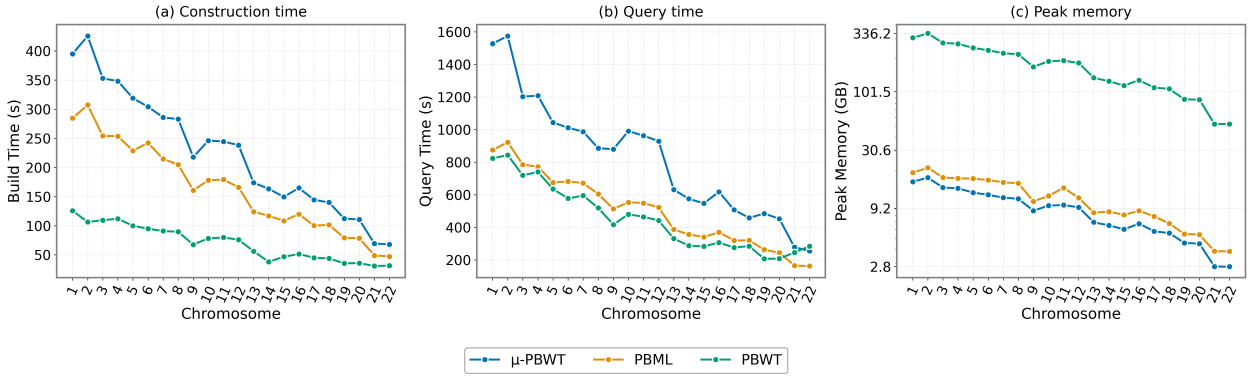

Fig. 1: Single-thread performance comparison across 22 chromosomes. PBML achieves substantially faster construction and query times than μ-PBWT with comparable memory usage. While PBML attains lower build times and competitive query performance, its memory footprint is prohibitively large, highlighting the superior efficiency and scalability of PBML.

Figure 1 shows the single-threaded construction, query performance, and peak memory usage across the 22 chromosomes. PBML constructs the PBWT index and auxiliary structures 38.6% faster than μ-PBWT, and PBWT has a 55.3% faster construction time compared to PBML. For 1,000 queries, PBML completes SMEM querying 63.8% faster than μ-PBWT and is only 6.4% slower than PBWT. This demonstrates that SMEM querying is performed more efficiently using the forward-backward algorithm rather than computing and using matching statistics. In terms of peak memory, PBML requires 22.0% more memory than μ-PBWT but compared to PBWT, it uses 14 times less memory, demonstrating that it delivers a highly efficient and scalable trade-off between construction time, query performance, and memory consumption.

With the thread count increasing to 16, we compare PBML only to μ-PBWT. We omit PBWT since it does not support multi-threading. On 16 threads, both PBML and μ-PBWT achieve faster query times on all chromosomes (Figure 2). Although PBML was already faster in the single-threaded mode, it had slightly higher memory usage compared to μ-PBWT. However, in multi-threaded execution PBML outperforms μ-PBWT by 305.7% in the mean query speed, while reducing the memory usage gap. Hence, PBML used only 9.4% more memory. In terms of scalability, PBML achieves a parallel speedup of 83.3% with near constant memory consumption (a decrease of 0.3%), while μ-PBWT reaches only a speedup of 57.9% with a 15.9% increase in memory usage. This spike in μ-PBWT memory results from storing all matching statistics during parallel queries.

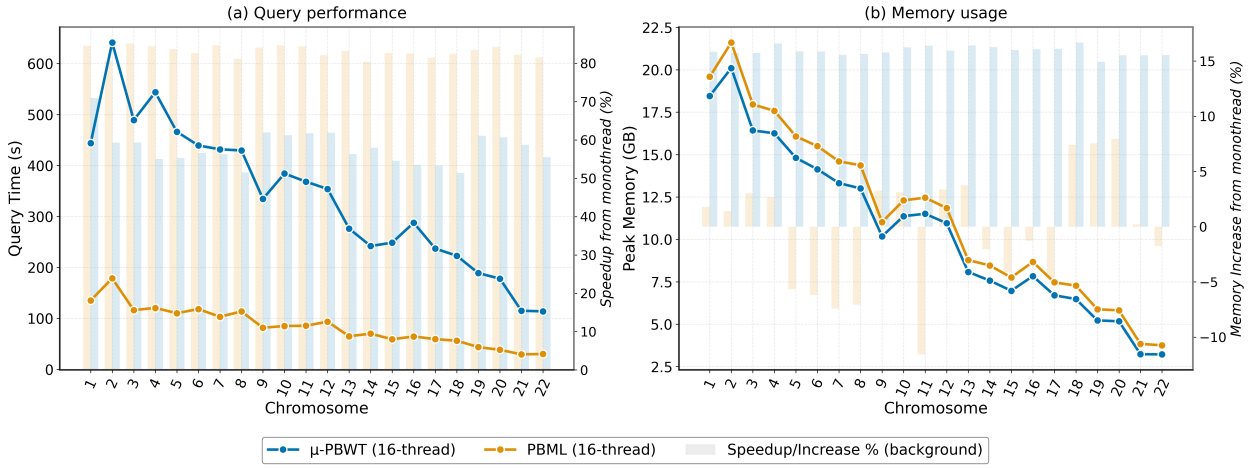

Fig. 2: Multi-threaded comparison of SMEM query performance and peak memory usage across 22 chromosomes under 16 threads. PBML demonstrates superior querying efficiency, better parallel scaling, and stable memory behavior compared to μ-PBWT.

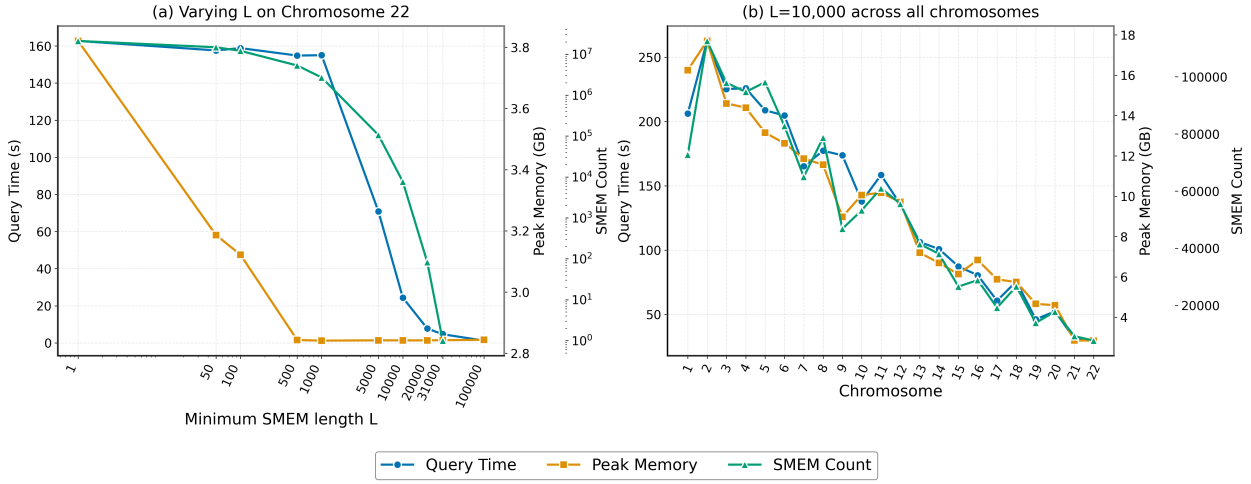

Fig. 3: Performance of PBML in large-scale SMEM discovery. (a) Effect of varying minimum SMEM length (L) on chromosome 22, showing sharp reductions in query time, peak memory, and SMEM count as L increases. (b) Genome-wide performance at $L = 10{,}000$ across all autosomes, illustrating consistent trends of reduced query time and memory usage proportional to SMEM count.

4.4 Finding long SMEMs in the 1KGP dataset

We evaluate the scalability of PBML on chromosome 22 as a function of the minimum SMEM length threshold L. When $L = 1$, PBML returns ∼21M SMEMs, yielding the largest query time and peak memory. As L increases, both metrics drop sharply and plateau by $L \approx 500$ (Fig. 3a). Even at an extreme threshold (i.e., when L is at least 10,000), PBML reports 7,674 long SMEMs while achieving a 6.67 times acceleration, and a 26% reduction in memory relative to $L = 1$.

To assess genome-wide performance, we ran PBML on all chromosomes at with $L = 10{,}000$ and recorded query time, peak memory, and the number of SMEMs returned (Figure 3b). By mean per-chromosome estimates, PBML ran 74.5% faster (saving 368.42 seconds per chromosome) and consumed 20% less memory (2.3 GB reduction per chromosome) compared to $L = 1$. When aggregated genome wide, the total runtime was reduced by 73.3% (8,105 s). The most pronounced per-chromosome acceleration was observed for chromosome 19 (82.4%), whereas the maximal reduction in peak memory occurred for chromosome 11 (27.8%).

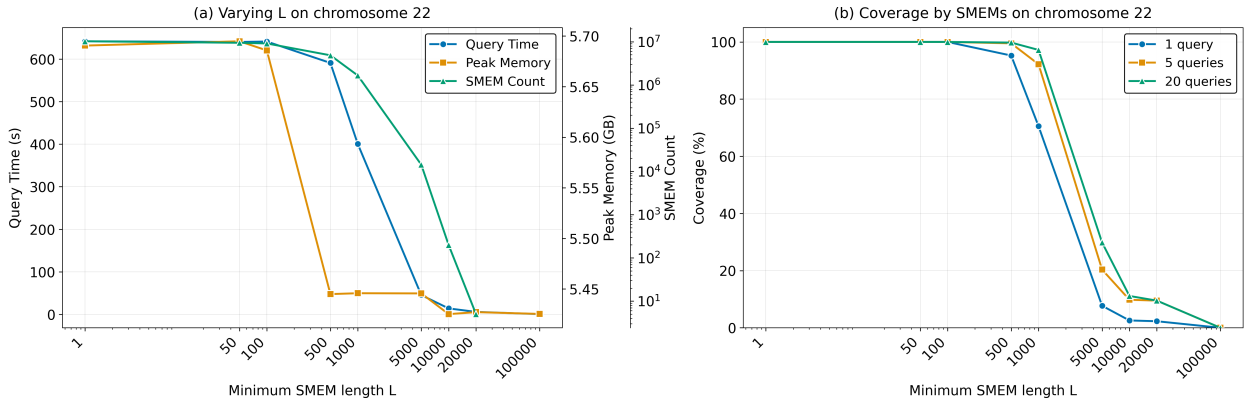

Fig. 4: Performance of PBML on the BIG dataset. (a) Increasing the minimum SMEM length (L) on chromosome 22 sharply reduces query time and SMEM count with minimal change in memory. (b) Coverage remains high for moderate thresholds ($L = 500$–1000) but drops rapidly at more aggressive settings.

In summary, PBML efficiently identified over 1.15 million long SMEMs genome-wide (per-chromosome mean of 52,202), demonstrating robust performance and substantial computational gains under stringent length thresholds.

4.5 Finding long SMEMs in the BIG data

We perform the same analysis by varying SMEM length on chromosome 22 of BIG dataset. At $L = 1$, PBML reports ∼10.2M SMEMs with the highest query time and memory usage. As L increases, query time and SMEM count decline sharply, while peak memory remains nearly constant and stabilizes by $L \approx 500$ (Fig. 4a). Large L values produce substantial acceleration: at $L = 1{,}000$, and $L = 5{,}000$, PBML queries 37.6%, 92.9% faster respectively than at $L = 1$.

Moderate thresholds provide the strongest balance between efficiency and coverage. At $L = 500$, single-query coverage remains high (95.2%), and coverage for 5 and 20 queries stays effectively complete (99.5% and 99.8%, respectively), even though the SMEM count drops from ∼10.2M to ∼3.1M (a 70% reduction). Increasing the threshold to $L = 1000$ yields a 37.6% speedup while still retaining 92.2% coverage for 5 queries and 97.2% for 20 queries, accompanied by a sharper decline in SMEM count to ∼1.66M (an 83.8% reduction from $L = 1$). These results show that PBML maintains strong biological coverage while delivering major computational gains for large L, whereas very large values, (e.g., $L \geq 5000$) are appropriate only when exclusively long matches are desired.

5 Conclusion

We introduced PBML, a PBWT querying algorithm that enumerates SMEMs of length at least L. Our method adapts BML to the PBWT, eliminating the need for auxiliary LCP/LCS structures while directly targeting informative long matches. On 1KGP, PBML achieves faster index construction than μ-PBWT and delivers substantial single- and multi-threaded speedups with modest memory overhead; it also efficiently retrieves long SMEMs ($L \geq 10{,}000$) that are most relevant for recent co-ancestry. Application to a diverse biobank cohort further illustrates how restricting to long matches sharpens downstream analyzes such as local-ancestry mapping. Building on these results, we will integrate PBML with local-ancestry pipelines on the BIG cohort to refine the boundaries of long exact-match regions and calibrate the length distribution of these regions against recombination maps; harmonize PBWT panels between BIG and 1KGP to assess cross-cohort transferability of long-match queries; and extend PBML to multi-allelic and graph-based PBWT variants. Lastly, the integration of distributed parallelism and incremental updates to accommodate rolling biobank releases warrants future investigation.

Acknowledgements

We gratefully acknowledge the Tennessee Biorepository and Integrative Genomics (BIG) Initiative for providing access to the biobank data from the Mid-South population, including African American participants from Memphis, which were analyzed in this study. We thank the participants, clinical staff, and investigators whose efforts made these data available.

References

1. Brian K. Maples, Simon Gravel, Eimear E. Kenny, and Carlos D. Bustamante. RFMix: A Discriminative Modeling Approach for Rapid and Robust Local-Ancestry Inference. *The American Journal of Human Genetics*, 93(2):278–288, 2013.
2. Sharon R. Browning, Ryan K. Waples, and Brian L. Browning. Fast, accurate local ancestry inference with flare. *The American Journal of Human Genetics*, 110(2):326–335, 2023.
3. Wen-Wei Liao, Mobin Asri, Jana Ebler, Daniel Doerr, Marina Haukness, Glenn Hickey, Shuangjia Lu, Julian K. Lucas, Jean Monlong, Haley J. Abel, et al. A draft human pangenome reference. *Nature*, 617(7960):312–324, 2023.
4. Peter Ebert, Peter A. Audano, Qihui Zhu, Bernardo Rodriguez-Martin, David Porubsky, Marc Jan Bonder, Arvis Sulovari, Jana Ebler, Weichen Zhou, Rebecca Serra Mari, et al. Haplotype-resolved diverse human genomes and integrated analysis of structural variation. *Science*, 372(6537):eabf7117, 2021.
5. Iona M. MacLeod, Denis M. Larkin, Harris A. Lewin, Ben J. Hayes, and Mike E. Goddard. Inferring Demography from Runs of Homozygosity in Whole-Genome Sequence, with Correction for Sequence Errors. *Molecular Biology and Evolution*, 30(9):2209–2223, 2013.
6. Jianqi Zhang and Daniel O Stram. The role of local ancestry adjustment in association studies using admixed populations. *Genetic epidemiology*, 38(6):502–515, 2014.
7. Bjarni V Halldorsson, Hannes P Eggertsson, Kristjan HS Moore, Hannes Hauswedell, Ogmundur Eiriksson, Magnus O Ulfarsson, Gunnar Palsson, Marteinn T Hardarson, Asmundur Oddsson, Brynjar O Jensson, et al. The sequences of 150,119 genomes in the uk biobank. *Nature*, 607(7920):732–740, 2022.
8. Mashaal Sohail, María J Palma-Martínez, Amanda Y Chong, Consuelo D Quinto-Cortés, Carmina Barberena-Jonas, Santiago G Medina-Muñoz, Aaron Ragsdale, Guadalupe Delgado-Sánchez, Luis Pablo Cruz-Hervert, Leticia Ferreyra-Reyes, et al. Mexican biobank advances population and medical genomics of diverse ancestries. *Nature*, 622(7984):775–783, 2023.
9. Daniel Taliun, Daniel N Harris, Michael D Kessler, Jedidiah Carlson, Zachary A Szpiech, Raul Torres, Sarah A Gagliano Taliun, André Corvelo, Stephanie M Gogarten, Hyun Min Kang, et al. Sequencing of 53,831 diverse genomes from the nhlbi topmed program. *Nature*, 590(7845):290–299, 2021.
10. All of Us Research Program Investigators. The "all of us" research program. *New England Journal of Medicine*, 381(7):668–676, 2019.
11. Richard Durbin. Efficient haplotype matching and storage using the positional burrows–wheeler transform (pbwt). *Bioinformatics*, 30(9):1266–1272, 2014.
12. Brian L Browning, Xiaowen Tian, Ying Zhou, and Sharon R Browning. Fast two-stage phasing of large-scale sequence data. *The American Journal of Human Genetics*, 108(10):1880–1890, 2021.
13. Robin J Hofmeister, Diogo M Ribeiro, Simone Rubinacci, and Olivier Delaneau. Accurate rare variant phasing of whole-genome and whole-exome sequencing data in the uk biobank. *Nature genetics*, 55(7):1243–1249, 2023.
14. O Delaneau, JF Zagury, MR Robinson, JL Marchini, and ET Dermitzakis. Accurate, scalable and integrative haplotype estimation. nat. commun. 10, 5436, 2019.
15. Simone Rubinacci, Robin J Hofmeister, Bárbara Sousa da Mota, and Olivier Delaneau. Imputation of low-coverage sequencing data from 150,119 uk biobank genomes. *Nature Genetics*, 55(7):1088–1090, 2023.
16. Adam M Novak, Erik Garrison, and Benedict Paten. A graph extension of the positional burrows–wheeler transform and its applications. *Algorithms for Molecular Biology*, 12(1):18, 2017.
17. Yaoling Yang, Richard Durbin, Astrid KN Iversen, and Daniel J Lawson. Sparse haplotype-based fine-scale local ancestry inference at scale reveals recent selection on immune responses. *Nature Communications*, 16(1):2742, 2025.
18. Ardalan Naseri, Erwin Holzhauser, Degui Zhi, and Shaojie Zhang. Efficient haplotype matching between a query and a panel for genealogical search. *Bioinformatics*, 35(14):i233–i241, 2019.
19. Ying Zhou, Sharon R Browning, and Brian L Browning. A fast and simple method for detecting identity-by-descent segments in large-scale data. *The American Journal of Human Genetics*, 106(4):426–437, 2020.
20. William A Freyman, Kimberly F McManus, Suyash S Shringarpure, Ethan M Jewett, Katarzyna Bryc, 23, Me Research Team, and Adam Auton. Fast and robust identity-by-descent inference with the templated positional burrows–wheeler transform. *Molecular Biology and Evolution*, 38(5):2131–2151, 2021.

21. Ardalan Naseri, Degui Zhi, and Shaojie Zhang. Multi-allelic positional burrows-wheeler transform. *BMC bioinformatics*, 20(Suppl 11):279, 2019.
22. Jouni Sirén, Erik Garrison, Adam M Novak, Benedict Paten, and Richard Durbin. Haplotype-aware graph indexes. *Bioinformatics*, 36(2):400–407, 2020.
23. Ardalan Naseri, William Yue, Shaojie Zhang, and Degui Zhi. Efficient haplotype block matching in bi-directional pbwt. In *21st International Workshop on Algorithms in Bioinformatics (WABI 2021)*, pages 19–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021.
24. Victor Wang, Ardalan Naseri, Shaojie Zhang, and Degui Zhi. Syllable-pbwt for space-efficient haplotype long-match query. *Bioinformatics*, 39(1):btac734, 2023.
25. Davide Cozzi, Massimiliano Rossi, Simone Rubinacci, Travis Gagie, Dominik Köppl, Christina Boucher, and Paola Bonizzoni. μ-pbwt: a lightweight r-indexing of the pbwt for storing and querying uk biobank data. *Bioinformatics*, 39(9):btad552, 2023.
26. Pramesh Shakya, Ahsan Sanaullah, Degui Zhi, and Shaojie Zhang. Dynamic μ-pbwt: Dynamic run-length compressed pbwt for biobank scale data. In *International Conference on Research in Computational Molecular Biology*, pages 209–226. Springer, 2025.
27. Travis Gagie. How to find long maximal exact matches and ignore short ones. In Joel D. Day and Florin Manea, editors, *In the Proceedings of the Developments in Language Theory (DLT)*, volume 14791 of *Lecture Notes in Computer Science*, pages 131–140, Cham, 2024. Springer. Göttingen, Germany, Aug 12–16, 2024.
28. Nathaniel K Brown, Lore Depuydt, Mohsen Zakeri, Anas Alhadi, Nour Allam, Dove Begleiter, Nithin Bharathi Kabilan Karpagavalli, Suchith Sridhar Khajjayam, Hamza Wahed, Travis Gagie, et al. KeBaB: k-mer Based Breaking for Finding Long MEMs. In *In Proceedings of the International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 10–17. Springer, 2025.
29. Heng Li. Exploring single-sample snp and indel calling with whole-genome de novo assembly. *Bioinformatics*, 28(14):1838–1844, 2012.
30. Robert S Boyer and J Strother Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.
31. Travis Gagie. How to find long maximal exact matches and ignore short ones. In *International Conference on Developments in Language Theory*, pages 131–140. Springer, 2024.
32. Heng Li. Bwt construction and search at the terabase scale. *Bioinformatics*, 40(12):btae717, 2024.
33. Silvia Buonaiuto, Franco Marsico, Akram Mohammed, Lokesh K Chinthala, Ernestine K Amos-Abanyie, Pjotr Prins, Khyobeni Mozhui, Robert J Rooney, Robert W Williams, et al. Insights from the Biorepository and Integrative Genomics pediatric resource. *Nature Communications*, 16:4750, 2025.
34. Alberto Policriti and Nicola Prezza. Lz77 computation based on the run-length encoded bwt. *Algorithmica*, 80(7):1986–2011, 2018.
35. Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-time text indexing in bwt-runs bounded space. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1459–1477. SIAM, 2018.
36. Alberto Policriti and Nicola Prezza. From lz77 to the run-length encoded burrows-wheeler transform, and back. *arXiv preprint arXiv:1702.01340*, 2017.
37. Takaaki Nishimoto and Yasuo Tabei. Optimal-time queries on bwt-runs compressed indexes. *arXiv preprint arXiv:2006.05104*, 2020.
38. Mohsen Zakeri, Nathaniel K Brown, Omar Y Ahmed, Travis Gagie, and Ben Langmead. Movi: a fast and cache-efficient full-text pangenome index. *Iscience*, 27(12), 2024.
39. Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pages 326–337, 2014.
40. James K Bonfield, John Marshall, Petr Danecek, Heng Li, Valeriu Ohan, Andrew Whitwham, Thomas Keane, and Robert M Davies. HTSlib: C library for reading/writing high-throughput sequencing data. *GigaScience*, 10(2), 02 2021. giab007.
41. Petr Danecek, James K Bonfield, Jennifer Liddle, John Marshall, Valeriu Ohan, Martin O Pollard, Andrew Whitwham, Thomas Keane, Shane A McCarthy, Robert M Davies, et al. Twelve years of samtools and bcftools. *Gigascience*, 10(2):giab008, 2021.
42. Robin J. Hofmeister, Diogo M. Ribeiro, Simone Rubinacci, and Olivier Delaneau. Accurate rare variant phasing of whole-genome and whole-exome sequencing data in the uk biobank. *Nature Genetics*, 55:1243–1249, 2023.