Phasing Data from Genotype Queries via the μ-PBWT

Davide Cozzi
Department of Computer Science, University of Milano-Bicocca, Italy

Paola Bonizzoni
Department of Computer Science, University of Milano-Bicocca, Italy

Christina Boucher
Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL, USA

Ben Langmead
Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA

Yuri Pirola
Department of Computer Science, University of Milano-Bicocca, Italy

Abstract

Genotype phasing – the process of reconstructing haplotypes from genotype data – is a fundamental problem in genomics with applications in ancestry inference, imputation, and disease association. Traditional phasing methods rely on statistical models or combinatorial approaches which can be computationally expensive, particularly when applied to large-scale reference panels.

In this paper, we present a first exploration of using the μ-PBWT (a run-length encoded Positional Burrows–Wheeler Transform) to solve the genotype phasing problem with a reference panel. Leveraging our previous results on positional substrings, we propose an approach that can explain a query genotype if the corresponding haplotype pair exists in the input panel. Moreover, our method is extended to cases where such a pair does not exist, even though some regions should remain unphased if they cannot be explicitly explained using the reference panel.

We implemented this method and compared it against Beagle, a state-of-the-art phasing tool, demonstrating that, in the absence of mutations and recombinations, our approach correctly identifies the haplotype pair that explains a genotype query while using seven times less memory than Beagle. However, we also observe that as mutation rates increase, the quality of the phasing decreases as a result of the growing difficulty of identifying consistent haplotype pairs in the presence of sequence variation.

These findings highlight the potential of μ-PBWT as an efficient alternative for genotype phasing, particularly in settings where computational resources are limited. The source code is publicly available at `https://github.com/dlcgold/muPBWT/tree/phase`.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Positional Burrows–Wheeler Transform, r-index, minimal position substring cover, set-maximal exact matches, genotype phasing

Digital Object Identifier 10.4230/OASIcs.Manzini.2025.10

Supplementary Material
Software (Source code): `https://github.com/dlcgold/muPBWT/tree/phase`
archived at `swh:1:dir:8131138a04fdd3a78c074d3b5707c43694bc7914`

Funding P.B., Y.P., and D.C. have received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement PANGAIA No. 872539. P.B. is also supported by the grant MIUR 2022YRB97K, PINC, Pangenome Informatics: from Theory to Applications, funded by the EU, Next-Generation EU, Mission 4 and ITN ALPACA N.956229. C.B. was supported by NIH/NIAID (Grant No. R01AI14180) and NSF:SCH:INT (Grant No. 2013998). C.B. and B.L were supported by NIH:NHGRI: (Grant No. R56HG013865).

1 Introduction

Within the field of haplotype analysis, the Positional Burrows-Wheeler Transform (PBWT) [9] has emerged as a fundamental data structure for efficiently encoding and searching large-scale haplotype data. Its ability to rapidly detect long shared substrings between a query sequence and a reference panel makes it a crucial tool for various genomic applications, including haplotype imputation [19, 18], ancestral inference [13, 25], and genotype phasing [15]. One of the key advantages of PBWT is its efficient representation of haplotype panels, allowing fast and memory-efficient computations, particularly compared to traditional sequence alignment-based approaches. Over the years, PBWT has been extended and refined to support more complex operations [24, 26, 17, 21], making it an essential component of modern haplotype analysis pipelines.

Sanaullah et al. [22], who formulated the haplotype threading problem, the task of representing a query haplotype using a minimal set of substrings derived from a reference panel, introduced a significant development in PBWT-based haplotype analysis. To address this problem, they introduced the Minimal Positional Substring Cover (MPSC) problem, which seeks to identify the smallest collection of substrings that fully cover a query while maintaining positional consistency with the reference haplotypes. Their work proposed an efficient PBWT-based solution by modeling the solution space as a graph, enabling computationally efficient haplotype threading. However, their approach focused primarily on haplotypes rather than genotypes, leaving an open question as to how these methods could be extended to genotype phasing, which requires handling heterozygous loci and potential recombination events.

Building on this work, in 2024, Bonizzoni et al. [3] demonstrated that the MPSC problem could be solved in sublinear space, significantly improving its scalability for large reference panels. That approach was based on efficiently computing k-Set Maximal Exact Matches (SMEMs) via k-Matching Statistics (k-MS), a technique that enabled the identification of minimal positional substring covers with reduced memory requirements. This advancement allowed for an efficient solution to the MPSC problem and its variations proposed by Sanaullah et al. The ability to solve these problems in sublinear space is particularly valuable for large-scale genomic datasets, where traditional PBWT-based methods may become memory-intensive.

In this paper, we extend the results of Bonizzoni et al. [3] to the problem of genotype phasing, adapting the MPSC framework to work with genotype queries instead of haplotypes. The main idea that inspires this work is the observation that large reference panels likely include a haplotype pair that explains a genotype query. However, we need methods that can scale to such large panels. Here, we introduce a novel approach using μ-PBWT [8], a refined version of run-length encoded PBWT (RLPBWT) [2], to efficiently phase genotypes while optimizing for minimal recombination events. Our method is designed to reconstruct haplotype pairs that explain a given genotype by leveraging positional substrings (PSs) and complementary positional substrings (CPSs) extracted from a reference panel. Unlike previously proposed combinatorial methods that assume a perfect haplotype match [14, 4, 1], our approach can handle cases where recombinations are required to fully explain the genotype, making it more adaptable to real-world data. In addition, current state-of-the-art tools such as Beagle [5, 6] are based on the Li and Stephens Hidden Markov Model [16], while our approach is deterministic in the absence of mutation and recombination.

Our study focuses on two primary scenarios: (1) an ideal case where the reference panel contains two haplotypes that fully explain the genotype without requiring recombination, and (2) a more realistic setting where recombinations across haplotype segments of the panel are needed due to the absence of a single haplotype pair that explains the genotype query. To address these challenges, we introduce a combinatorial approach that processes 2-positional substrings (2PSs) and identifies valid haplotype pairs that minimize recombination events. Furthermore, we implement and evaluate our method against Beagle [6, 5], a state-of-the-art statistical phasing tool. Our experimental results demonstrate that μ-PBWT achieves competitive accuracy while significantly reducing computational costs in scenarios with low mutation rates. On the other hand, introducing target mutations (and consequently recombinations not having the target parents in the reference panel) reduces the performance of our approach, in terms of switch error rate/mismatch rate. Our method requires little memory and can scale on very large reference panels, thanks to the μ-PBWT index. This makes μ-PBWT a promising alternative for applications where memory efficiency and processing speed are critical.

2 Related work

Due to the diploid nature of humans, chromosomes come in two copies, called haplotypes, each copy inherited from one of the two parents (see [4] for a survey of the main concepts). Sequencing an individual is a process that does not distinguish between the two haplotypes, as reads are fragments uniformly distributed over both chromosomal copies. Distinguishing the haplotypes from sequencing reads is biologically expensive, and thus computational methods are employed for this purpose.

A genotype is simply the conflated information of a pair of haplotypes and can be easily determined by sequencing. More precisely, a genotype is specified by a vector of allele pairs at each locus, where the pairs consist of homologous or heterozygous alleles. In a genotype, information on which heterozygous allele belongs to which haplotype is not available and must be determined.

Observe that each haplotype inherited from a parent is itself the result of a mosaic of the two parental haplotypes, due to *recombination* events, which are a typical evolutionary phenomenon occurring in human populations. Recombination events regulate Mendelian inheritance laws within a family trio and have been extensively studied in the context of combinatorial methods for the phasing or haplotyping of input genotype matrices [4].

From a combinatorial perspective, a haplotype at a set of biallelic loci, the genomic positions in which only two distinct alleles are observed in a population, is formalized as a binary string. In this representation, the i-th position of the string is assigned a 0 if the individual carries the reference allele at that locus (i.e. the allele matches the reference in that locus), and a 1 if the individual carries the alternative allele.

The *genotype* of an individual is the combined representation of the two haplotypes inherited from the two parents. Formally, a genotype is represented as a string on the alphabet $\{0, 1, 2\}$, where the i-th position is 0 if the individual has inherited the reference allele from both parents at the i-th locus, it is 1 if the individual has inherited the alternative allele from both parents at the i-th locus, or it is 2 if the individual, at the i-th locus, has inherited the reference allele from a parent and the alternative allele from the other parent. The positions where the genotype is 0 or 1 are called *homozygous*, while the others are called *heterozygous*. We say that a pair $\langle h_1, h_2 \rangle$ of w-length haplotypes explains a w-length genotype Q if and only if, for all $i \in [1, w]$ we have that $h_1[i] = h_2[i] = Q[i]$ if $Q[i] \neq 2$, or $h_1[i] \neq h_2[i]$ if $Q[i] = 2$. In simpler terms, a pair of haplotypes explains a genotype if they match the genotype at homozygous positions and differ at heterozygous positions.

Then, the *genotype phasing* problem aims at reconstructing an individual's haplotypes from their genotype. More specifically, determining the exact pair of haplotypes that correspond to the observed genotype. The problem is straightforward for homozygous positions, where both haplotypes match the genotype; however, at heterozygous positions, the phase is ambiguous because multiple pairs of haplotypes, differing only in the assignment of alleles to each parent, can equally explain the genotype. There is a vast literature on combinatorial methods for genotype phasing, mainly distinguished by how the problem is modeled and by the type of input data considered. For example, Gusfield proposed solving genotype phasing using the perfect phylogeny model [14, 1] when the input matrix consists of a collection of genotype vectors.

The availability of large panels of population haplotype data has shifted the focus of genotype phasing toward the use of statistical methods which, based on information from an input panel, can effectively explain genotype queries. An example of this is Beagle [5], which uses phased input data, that is, a known haplotype panel, to build a hidden Markov model specifically tailored to diploid data.

In this paper, genotype phasing is addressed through the so-called haplotype threading, which models the process of reconstructing a haplotype as a mosaic of haplotypes from a reference panel.

More precisely, haplotype threading aims to cover a haplotype query using segments from haplotypes in an input panel. The main idea is that a haplotype results from the accumulation of recombination events, combining chromosomal segments from various haplotypes in the panel. The Li and Stephens (LS) hidden Markov model (HMM) [16] produces a haplotype threading for a query haplotype using a panel of known haplotypes, requiring time that is linear in the size of the input panel. However, this running time becomes infeasible when the haplotype panel is large, and thus sublinear-time solutions have recently been investigated, mainly using the PBWT (Positional Burrows-Wheeler Transform) framework.

Moreover, modern haplotype panels typically exhibit much lower mismatch rates and switch error rates, two key measures commonly used in the probabilistic modeling of the LS framework. Observe that a zero mismatch rate between the query haplotype and the input panel corresponds to the situation where the query can be entirely covered by segments from the panel, with the number of such segments directly corresponding to the switch error. In this case, combinatorial approaches become particularly attractive, especially when they offer faster solutions than LS-based methods.

In particular, the solution proposed by Sanaullah et al. [20], the Minimum Positional Substring Cover (MPSC) with non-overlapping segments, corresponds exactly to haplotype threading under a zero mismatch error assumption. When overlapping segments are permitted, identifying the breakpoint between two matching positional substrings corresponds to determining the transition point between two distinct haplotypes in the panel.

Now, the input panel is not guaranteed to include the two haplotypes from which a query genotype may have been generated, nor do we know whether the two haplotypes of a parent are present in the panel when explaining a haplotype as the result of recombinations from that pair. This is why the mosaic of segments may involve multiple haplotypes of the panel.

The introduction of the PBWT has significantly improved the time efficiency of computational approaches to haplotype threading. The first major improvement concerns the time required to find coverage of a haplotype, which has been reduced from $O(N \cdot M)$ to $O(N)$, where N is the number of sites in a haplotype sequence [20]. This reduction in time complexity allows the methods to scale to a large number M of haplotype sequences, which was not feasible with the original LS-based method. Finally, since the memory required by these approaches can be extremely prohibitive, a further improvement was proposed in [3], where a compressed data structure is introduced to store and manage the input panel using the μ-PBWT [8]. In particular, space efficiency is crucial for handling large cohorts of haplotype data, helping to reduce mismatch errors and switch errors. In this paper, we will assume an ideal future framework where the large amount of data allows for the assumption that the two haplotypes that explain a genotype are part of the input data.

3 Background

3.1 The Minimal Positional Substring Cover Problems

Throughout this paper, we define a string X on a finite, ordered alphabet $\Sigma = \{c_1, \ldots, c_\sigma\}$ to be the concatenation of $|X| = w$ characters $X = X[1..w]$ from Σ. We denote the empty string as ε, the string that spans i through j as $X[i..j]$ (with $X[i..j] = \varepsilon$ if $i > j$), the i-th prefix of X as $X[1..i]$, and the i-th suffix as $X[i..|X|]$.

A *positional substring* of a string X, in short PS, is a triplet (i, j, X) with $1 \leq i, j \leq |X|$ and we say that the substring corresponding to (i, j, X) is $X[i..j]$. Two positional substrings (i, j, X) and (k, l, Y) are *equal* if and only if $i = k$, $j = l$, and $X[i..j] = Y[k..l]$. A positional substring (i, j, X) is *contained* in a string Y if and only if $X[i..j] = Y[i..j]$.

Given a set S of strings of length w (i.e., a panel), a positional substring (i, j, X) is a k-positional substring in S, in short kPS, if (i, j, X) is contained in at least k strings of S. A *k-positional substring cover* of a w-length string P by S is a set C of positional substrings of P such that: *(i)* each position $l \in [1, w]$ of P is covered by a $(i, j, X) \in C$ (i.e., $i \leq l \leq j$), *(ii)* each $(i, j, X) \in C$ is contained in P, and *(iii)* each $(i, j, X) \in C$ is contained in at least k distinct strings of S (i.e., they are kPS in S). The size of the cover is the number of elements in C, which we denote as $|C|$.

Given a set S of h strings of length w and a string P of length w, the *k-Minimal Positional Substring Cover problem* (k-MPSC) [20] seeks to compute, if it exists, a k-positional substring cover of P by S with the smallest size over all k-positional substring covers of P by S.

We note that the MPSC problem is the k-MPSC problem where k is equal to 1, i.e., each positional substring of the cover is contained in at least one string of the panel. It is easy to see that a solution to the problem exists if and only if for every i, with $1 \leq i \leq w$, the positional substrings (i, i, P) are contained in at least k distinct strings of S, i.e., they are kPS.

Given a panel S and a string P there can exist several distinct k-MPSC of the same size (hence, several solutions to the k-MPSC problem). Since the solution to k-MPSC can affect downstream applications, three variants have been proposed [20, 22] to constrain the solution.

Sanaullah et al. [22] defined the problem of computing Leftmost MPSC using 1-positional substring covers, while we generalize the method in [3] to compute Leftmost k-positional substring covers. In the problem definition, given a k-MPSC C, the i-th positional substring of C (for $1 \leq i \leq |C|$) refers to the i-th substring when listing all positional substrings of C in order of their starting positions.

Given a set S of h strings of length w and a string P of length w, the *Leftmost k-MPSC problem* asks to find a *leftmost* k-MPSC C of P by S, where a leftmost k-MPSC is a k-MPSC of P by S such that any i-th substring in C starts at least as early as the i-th substring of every other MPSC of P by S.

For the purposes of this manuscript, we relax the problem of computing MPSC to compute a positional Substring Cover of P by S, which is not necessarily minimal, and we denote this solution as a PSC.

3.2 Solving the k-MPSC problem variants using PBWT

The best known algorithm for computing a k-MPSC requires $O(w)$ time [22], assuming that an index of the haplotype panel S is given as input. This algorithm processes the haplotype panel S column-wise from left to right, extending the matches of the query string P that are shared with at least k strings of the panel S. In each column, if the current match cannot be extended, then a new match is started from the current column. Optimality is ensured by the property of k-MPSC modularity [20, Lemma 2]. Matches of P with at least h strings in S are efficiently computed and extended using the Positional Burrows-Wheeler Transform (PBWT) of S. Hereon, we denote $h \cdot w$ as n, which will be used throughout this paper to bind the space and time complexity. The k-MPSC algorithm of Sanaullah et al. [20] requires $O(n)$-space to ensure random access in constant time to the input panel and to the PBWT.

3.3 Positional Burrows–Wheeler Transform and μ-PBWT

In 2014, Durbin proposed the PBWT [9] as an efficient data structure to perform internal and external pattern matching on a set $S = \{S_1, \ldots, S_h\}$ of h binary sequences of length w. The PBWT consists of two arrays per column j:(i) the *prefix array* PA_j and (ii) the *divergence array* DA_j.

Since the PBWT of column j is based on the co-lexicographic ordering of prefixes of S up to column $j-1$, PA_j stores the permutation of the set of row indices, i.e., the set $\{1, \ldots, h\}$. Formally, $\mathsf{PA}_j[i] = k$ if and only if $S_k[1..j-1]$ is the i-th element in this sorting in all row prefixes up to column $j-1$. On the other hand, considering two consecutive prefixes in the reordering for column j, i.e., $S_{\mathsf{PA}_j[i]}[1..j-1]$ and $S_{\mathsf{PA}_j[i-1]}[1..j-1]$, $\mathsf{DA}_j[i] = \ell$ iff ℓ is the length of the longest common suffix between those two prefixes. Note that, by definition, $\mathsf{PA}_1 = \{1, \ldots, h\}$ and $\mathsf{DA}_1 = \{0, \ldots, 0\}$.

Given the entire set of prefix arrays, we define the PBWT matrix $\mathsf{PBWT}[1..h][1..w]$ as the matrix obtained by permuting each column of the input panel by the corresponding prefix array. Formally, denoting the j-th column of a matrix X by $\mathsf{col}(X)_j$, each column of the PBWT matrix is defined as $\mathsf{col}(\mathsf{PBWT})_j[i] = \mathsf{col}(\mathsf{M})_j[\mathsf{PA}_j[i]]$ for all $i = 1..h$ and $j = 1..w$.

Additionally, to solve the pattern matching problem of a string against a panel, the PBWT is augmented with an FM-index [12] like data structure, used to follow a certain row in the permutations induced by the prefix arrays.

Regarding complexity, Durbin [9] proposed a set of algorithms that compute the entire set of prefix arrays and the entire set of divergence arrays in $O(hw)$-time and $O(hw)$-space.

Durbin highlighted the potential of a run-length compression of the PBWT in his original paper on this data structure. Later, Bonizzoni et al. [2] and Cozzi et al. [8] proposed various combinations of data structures to efficiently store and query a run-length encoded PBWT (RLPBWT).

In this paper, we consider the implementation of the RLPBWT of Cozzi et al. [8], referred to as μ-PBWT. For a detailed explanation of its definition and its application in addressing the problem of identifying set-maximal exact matches between a reference panel and an external query, we direct readers to the original paper. Next, we recall one of the key ideas behind the μ-PBWT algorithm: Matching Statistics.

▶ **Definition 1** (Matching Statistics in the PBWT). *Given a set S of h binary w-long sequences $S = \{S_1, \ldots, S_h\}$ and a pattern $P[1..w]$, we define an array $MS[1..w]$ of* $(\mathsf{row}, \mathsf{len})$ *pairs as Matching Statistics of P with respect to S where, for each position $1 \le j \le w$:*

- *$MS[j].\mathsf{row}$ and P share a $MS[j].\mathsf{len}$-long match ending in j*
- *this match is left-maximal, i.e., $P[j - MS[j].\mathsf{len}..j]$ does not match any sequence in S*
- *we have a mismatch iff $P[j]$ does not occur in column j and we represent this event as $MS[j].\mathsf{row} = -$ and $MS[j].\mathsf{len} = 0$*

Using Matching Statistics, we can solve MPSC in $O(w)$-time and $O(r)$-space, where r represents the total number of runs in the PBWT [3]. We refer the reader to Cozzi et al. [8] and Bonizzoni et al. [3] for details about the algorithm used to compute Matching Statistics and SMEMs.

4 The genotype phasing problem via positional substrings

In this section, we extend the definition of positional substring cover to the case of a genotype and a panel S of strings. To accomplish this, we first define the concept of a *complementary positional substring* (CPS) for patterns P and P'. A CPS consists of two positional substrings (i, j, P) and (i, j, P'), which are bitwise complementary. For simplicity, a complementary positional substring will be simply identified by a triple (i, j, P).

Building on this concept, we let Q be a w-long genotype query and S a set of w-long strings, then a PSC$_g$ of genotype Q consists of a set C of positional substrings (i, j, Q) that occur in at least two rows of panel S, i.e., 2PS, and a CPS (k, l, S) of S, such that: (i) (k, l, Q) is a substring of Q of only 2's, and (ii) all positions t of Q are covered by C.

Next, we formally state the problem investigated in this paper. The core objective is to phase the query genotype Q by covering it with positional substrings (PSs) or complementary positional substrings (CPSs) derived from a panel of haplotypes S.

▶ **Problem 1.** Genotype-Positional-cover
INPUT: a reference panel S of haplotypes and a genotype query Q,
OUTPUT: a PSC$_g$ for the query Q and the panel S.

In Problem 1, a natural optimization goal is to minimize ℓ, the size of the PSC$_g$. However, to further reduce the number of recombinations, we focus on minimizing the number of distinct pairs of haplotype sequences from S that contribute to the positional substrings used to cover Q. In other words, our goal is to minimize the number of 2-PSs and CPSs that originate from different haplotype pairs in S. We refer to this variant of Problem 1 as minimum recombination genotype-positional phasing. The problem has no recombinations if all 2-PSs and CPSs are derived from the same pair of haplotypes. This criterion will be incorporated into the heuristic method we propose to solve the genotype-positional phasing problem. The primary case we investigate in this paper is when the two haplotypes that resolve Q are already present in the panel S, which means that no recombinations occur. As we will discuss in Section 5, the case of no recombination has a straightforward and efficient solution. Finally, we recall that each 2PS is associated with a set of rows in the reference panel, and we will explicitly use this set to solve Problem 1.

5 Methods

In this section, we present our approach to solving Problem 1 using μ-PBWT. From this point forward, we make the simplifying assumption that each column in the reference panel contains at least two 0's and two 1's. This ensures that every homozygous position in the query genotype shares a 2PS with the reference panel, simplifying the presentation of our method. To effectively apply μ-PBWT, we first refine how it identifies matches and 2-positional substrings (2PSs) shared between the target genotype and the haplotype panel. Specifically, we introduce a constraint that limits the length of these 2PSs to at most two. The rationale behind this constraint will be explained at the end of this section, where we demonstrate how the use of 2-MPSCs affects the number of phased heterozygous sites.

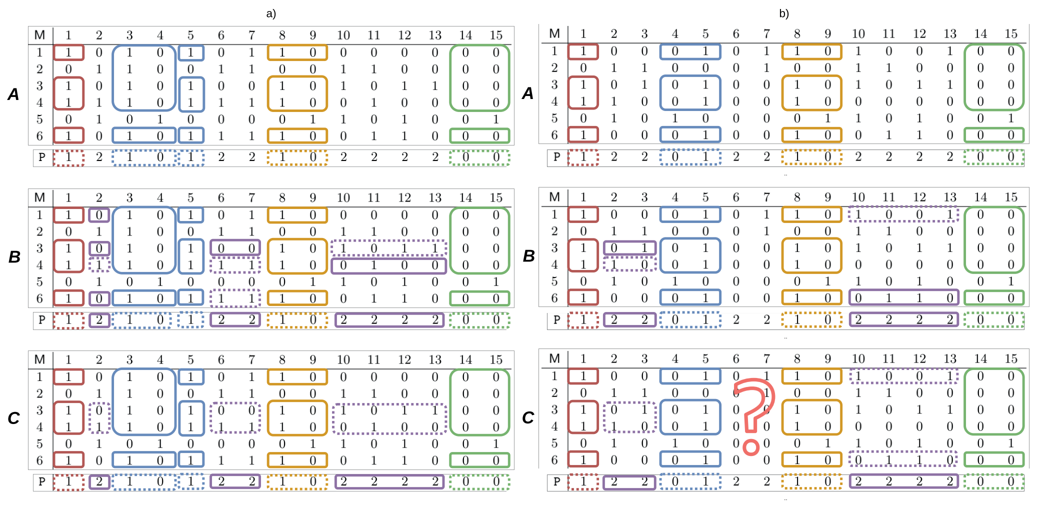

Figure 1 Example of solving the phasing problem a) assuming two haplotypes that fully explain the query genotype exist, i.e., in the absence of mutations and recombinations, and b) assuming this pair does not exist and we need to use different haplotype pairs for different genotype regions, i.e., with mutations and recombinations.

5.1 Computing 2PSs

First, we compute a set C of non-overlapping 2PSs of the genotype query with respect to the reference. The computation of 2PSs, which involves *at least* two rows in the reference panel, requires “resetting” the computation of two matching statistics each time $MS[i].len \geq 2$. This ensures that the computation restarts at $i + 1$, similar to handling a mismatch in column i. As a result, each entry in the matching statistics array now contains (likely not right-maximal) left-maximal matches with a length of at most two, involving at least two rows in the input panel. To prevent overlaps, we adopt a strategy similar to the one proposed in [3] for computing the leftmost MPSC set, thereby obtaining a set of non-overlapping 2PSs. Specifically, we modify that algorithm to incorporate 2-MSs, allowing us to skip matches shared by Q with only a single row in the panel, as well as positions corresponding to mismatches, which occur at heterozygous sites.

5.2 Solving the Haplotype Phasing Problem Without Recombinations

Now that we have computed the set C for query Q and reference panel S, we describe our approach for solving Problem 1. The key idea behind our approach is to analyze, for each iteration, two consecutive 2PSs of C to find possible haplotype pairs in S which can be used as anchors for phasing the heterozygous region of the query Q that lies between. In other words, given a heterozygous region in Q, we want to find a CPS that covers that region and shares at least a pair of rows with the previous 2PS and the next 2PS, i.e., the two anchors.

In practice, to solve a heterozygous region, we need to find two rows that complement each other bitwise. Given two consecutive 2PSs, we compute the intersection of the corresponding haplotype sets and use this intersection to extract the submatrix between them. Then, we naively find all the complementary haplotype pairs that contain the CPS explaining the substring of the genotype from the starting position of the first 2PS to the last position of the second 2PS. At each iteration, to avoid unnecessary computation and to consider previously computed pairs, we also need to store all the possible haplotype pairs which explain the genotype query up to the current position. To formalize this, we denote S_{pair} as the current set of row pairs. In other words, at each iteration, instead of simply considering two consecutive 2PSs in C, we refine the possible pairs using the temporary result, that is, the set of pairs that explain the current prefix of the genotype query. In other words, the left anchor results from the filtering procedure. A similar filtering process is applied when two consecutive 2PSs occur without any heterozygous region between them.

To simplify the explanation of this approach, we consider the ideal scenario in which the reference panel contains the two haplotypes that fully explain the genotype query. We iterate through the query Q and the set C of 2PSs from left to right, storing haplotype pairs that are compatible with the query for a certain prefix. Note that, due to the assumption of haplotype existence, at each filtering step, there is always at least one pair within the current S_{pair} compatible with the pairs associated with the right 2PS, even if we have a heterozygous gap. In other words, the left anchor, which results from the filtering procedure, always consists of at least a row pair. To better understand this case, we refer to Figure 1.a. In Figure 1.a.A, we depict the 2PS set C shared between a reference panel M and a genotype query P (respectively S and Q in the previous discussion). Our iterative approach, which filters incorrect row pairs, is shown in Figure 1.a.B. In this example, at first, we have several row pairs after considering the first two 2PS: $S_{pair} = \{(1, 4), (3, 4), (4, 6)\}$. Note that, for example, we exclude row 2 since it cannot be used as a left anchor, having a mismatch at position 1. Continuing the iteration, we have two consecutive 2PSs, without any heterozygous gaps. The unique rows in S_{pair} need to be filtered by the rows associated with this new right 2PS but, in this case, each pair in S_{pair} is also present in the pairs set related to the right 2PS, so we do not need to delete any pair. We now need to consider the fourth 2PS as the right anchor. Note that row 1, in the substring that spans positions 6 to 7, does not complement any of the other rows in $\{3, 4, 6\}$, which are the remaining rows obtained from the previous step. After this iteration, we get a new set of possible pairs which explain the genotype in the pattern prefix up to position 9: $S_{pair} = \{(3, 4)\}$. Despite rows 3 and 6 sharing a complementary substring between positions 6 and 7, they are not considered because they are not present in the first pair set (sharing the same symbol in position 2). Finally, in the last iteration, the pair $(3, 4)$ is confirmed to be the only row pair that explains the genotype, being present in the set of pairs associated with the last 2PS and sharing complementary rows between positions 10 and 13. Figure 1.a.C summarizes the solution phasing for the given input.

5.3 Handling Recombination Cases

The scenario with zero recombinations occurs when the sample belongs to an individual already present in the panel or the analysis is restricted to a region without recombinations within the panel. However, in practical cases where sequencing errors occur or the reference panel has a limited number of haplotypes, the introduction of mutations and recombinations becomes necessary. It is important to note that a mutation can create a small recombination event around the mutation site. In this context, recombination refers to different segments of the genotype being explained by different haplotype pairs from the reference panel.

We now describe how our method works in this scenario. Again, we proceed to scan the set of 2PSs from the left to the right, by processing consecutive 2PSs using also the current pairs set S_{pair}, according to the approach described above whenever we follow two haplotypes that are present in the panel.

When the above assumption (i.e. the assumption of having zero recombinations) does not hold, we need to manage two consecutive 2PSs differently (filtered by S_{pair}). Specifically, we need to account for two possible cases. Recall that each adjacent 2PS is associated with a list of row pairs, and we are considering the case where their intersection is empty. This leads to two possible scenarios: (1) there exist two row pairs – one from each 2PSs – that share at least one common haplotype but not both; or (2) no such two pairs exist. Recall that the row pairs associated with the 2PSs are filtered by considering the row pairs we have in S_{pair}. In the first scenario, only one haplotype recombines at this position between the two 2PSs, whereas in the second scenario, both haplotypes that explain the query undergo recombination at this position. In both cases, we store the temporary results up to the end of the first 2PS, then restart the iteration using all possible row pairs associated with the right 2PS. At this point, S_{pair} is stored and is updated to reflect the list of row pairs associated with the right 2PS.

In the second case, we encounter two consecutive 2PSs (after filtering by S_{pair}) separated by heterozygous regions that cannot be explained. This situation leads to two possible scenarios. First, a CPS covers the heterozygous region, but its associated list of row pairs does not overlap with any of the currently available pairs obtained from previous iterations. Second, the two anchor 2PSs do not share any associated row pairs in the reference panel. The latter case also involves any case in which there is no CPS in the heterozygous region. Both cases require different considerations when determining the compatible haplotype pairs to proceed.

In this situation, we can try to reset the current set S_{pair} by updating it with all the pairs associated with the left anchor, assuming a recombination. If we have pairs shared by this new anchor and the one related to the right 2PS that explain the underlying heterozygous region, we can continue to the next iteration using these pairs as S_{pair} after storing the previous S_{pair} in memory. On the other hand, if we do not have such pairs or simply if the CPS is not associated with any pair of rows in the panel, we skip those heterozygous sites and start the iteration again from the right 2PS. This implies that we need to update S_{pair} in the list of rows associated with the right 2PS but that these heterozygous sites will remain unphased. Observe that each time we need to reset the set of possible pairs and store the current S_{pair}, we add the last position in which it was consistent with the target genotype. In other words, each time we have a recombination, we store the current S_{pair} and the last position in which it explains a genotype substring.

After scanning the entire input genotype, we have a list of possible sets S_{pair} that cover various regions of the genotype, most likely all of them. Each genotype region can possibly be explained by more than one pair, so we must select the optimal haplotype pair. The selection is guided by an optimization function that aims to minimize the number of recombinant haplotypes. From a combinatorial perspective, we aim to find a list of haplotype pairs, one per region, that minimizes the number of pairs. This is the list of recombinant haplotypes that explain the input genotype. To compute this efficiently, we use a greedy algorithm to compute this list of pairs. Specifically, we sort all pairs by frequency and select the most frequent pair for each genotype region. In this way, for each genotype region, we get just one haplotype pair, which is the most common overall among the pairs that cover that region, i.e., the pair that covers more other regions. If some pairs have the same frequency, the selection is further weighted based on the pairs' choices made in previous regions to ensure consistency.

Figure 1.b illustrates the simplest case of recombination. In Figure 1.b.A, considering the first two 2PSs, we obtain $S_{pair} = \{(3, 4)\}$ as the current set of pairs. However, this pair cannot explain the genotype at positions 6 and 7, even resetting the left anchor not having any CPS in that region, so this region remains unphased. Using the previously described approach, as shown in Figure 1.b.B, we restart from the next 2PS pair and obtain

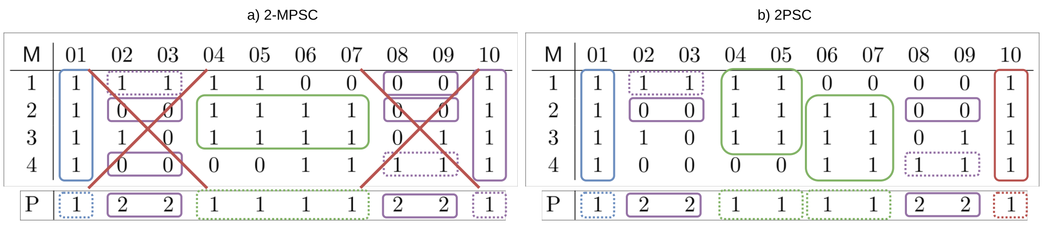

Figure 2 Example for advantages of using 2PS and CPS against MPSC. In a) we show that using MPSC we should not be able to phase the two heterozygous regions between sites 2-3 and 8-9, having a recombination inside the MPSC at sites 5-6, while in b) we show that using 2PS and CPS, those regions are correctly phased.

$S_{pair} = \{(1,6)\}$, which explains the last genotype region. At this point, we also store in memory $S_{pair} = \{(3,4)\}$ and the last consistent position with that set: 5. At the end of the iteration, not having other possible pairs, we store $S_{pair} = \{(1,6)\}$ and position 15. This results in a recombination event between $(3,4)$ and $(1,6)$ even though we cannot infer the exact haplotype configuration at positions 6 and 7, i.e. those sites remain unphased.

5.4 Advantages of 2PSs against 2-MPSCs

After explaining our method, it becomes clear why we needed to avoid using classical 2-MPSC in favor of 2PS. As shown in Figure 2.a, if we rely on the use of 2-MPSC, we require less flexibility in how we detect anchors. In detail, we should miss some recombinations that occur in a position inside a 2-MPSC. In fact, using 2-MPSCs as in the example, we have that the two heterozygous regions cannot be explained by any row pair despite having a CPS which covers that substring. The first and second MPSC do not share any associated row pair, which explains the heterozygous region in between positions 2 and 3. The same happens when restarting the iteration from the entire second 2-MPSC and considering the last 2-MPSC: the heterozygous region between positions 8 and 9 cannot be explained in any way.

However, using 2PS, as in Figure 2.b, splitting the central MPSC into two 2PS, we can detect the pair $S_{pair} = \{1,2\}$ that explains the genotype up to position 5, including $P[2..3]$, and the pair $S_{pair} = \{2,4\}$ that explains the genotype from position 6 to the end of the genotype query, including $P[8..9]$.

5.5 Time and Space Complexity

For the time and space complexity of MS and the leftmost MPSC, refer to [2, 3, 8]. Although computing 2PSs and MPSC share the same time complexity, the key difference lies in the number of rows that share the same match. Since 2PSs generally involve smaller matches, more rows tend to share them, increasing the computational overhead.

Additionally, our approach has a time complexity that is quadratic in the number of possible haplotype pairs shared between the two anchors. In terms of extra space requirements, we store the input panel along with a vector of sparse bitvectors, enabling efficient reconstruction of submatrices. This avoids recomputing them using μ-PBWT in $O(h \log \rho)$, with ρ as the total number of runs in the PBWT. Furthermore, we must consider that storing all 2PSs and the current set of haplotype pairs can become memory intensive, having in the worst case a total number of pairs which is quadratic in the number of haplotypes.

Table 1 Total number of heterozygous variation sites as the mutation rate varies.

Mutation rate	No. of het. sites
0 %	60 604
1 %	60 862
3 %	61 362
5 %	61 995
10 %	63 153
20 %	65 265

6 Results

In this section, we present some preliminary experimental results of our approach. Since this approach is a proof-of-concept, the current implementation is inefficient for most subtasks, from the extraction of complementary rows to the greedy algorithm used to select the best pairs of haplotypes. All of these steps could be further optimized in the future, both theoretically and practically.

To assess the applicability of our theoretical results, we integrate the genotype phasing algorithm into the μ-PBWT codebase and tested it in a simple genotype phasing scenario. For this experiment, we considered the HGSVC2 reference panel [7, 10] for human chromosome 2, with 68 haplotypes (34 samples) and 229 300 biallelic variation sites. We randomly selected a sample (*target sample*) from this reference panel as a query, combining its haplotypes into an unphased genotype target. The target sample we selected has 60,604 heterozygous sites ($\sim$ 26% of the total number of variants on chromosome 2).

We used VCF as input format where 0|0 and 1|1 represent phased homozygous sites, 0|1 and 1|0 phased heterozygous sites, 0/0 and 1/1 unphased homozygous sites, and 0/1 unphased heterozygous sites.

We compared μ-PBWT with Beagle [5, 6], a state-of-the-art phasing tool that is also based on the use of a reference panel. We ran Beagle with default parameters, using the HapMap [23] chromosome 2 genetic map for the GRCh38 reference genome.

We ran our experiments on a machine equipped with an Intel Xeon CPU E5-4610 v2 (2.30GHz), 256 GB of RAM, 8 GB of swap, and Ubuntu 20.04.6 LTS 64-bit with kernel 5.15.0. We measured execution times and peak memory usage using the Unix `time` command.

Moreover, we simulate mutations in the target genotype by replacing the genotype at each position with probability ε. We considered five different values for ε: 1%, 3%, 5%, 10%, and 20%. We refer to ε as *mutation rate*. Table 1 summarizes the total number of heterozygous sites after the mutation step. As explained above, mutations cause small spurious recombinations at nearby sites. Thus, this small experiment allows us to test more realistic scenarios.

The accuracy of the results was evaluated by considering *(i)* the *switch error rate*, i.e., the rate of phase changes between the predicted haplotypes w.r.t. the ground truth haplotypes, *(ii)* the *mismatch rate*, i.e., the rate at which two switch errors occur at consecutive positions, and *(iii)* the percentage of heterozygous sites that have been phased. The three metrics were computed using the utility script publicly shared by the developers of HapCUT2 [11][1], considering the two haplotypes of the target sample as ground truth.

[1] `https://github.com/vibansal/HapCUT2/tree/master/utilities`

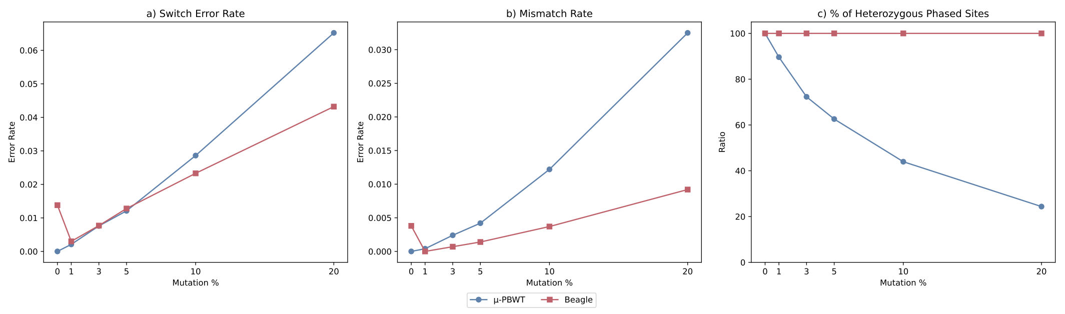

Figure 3 Various statistical results of μ-PBWT and Beagle at different genotype mutation rates. In a) we have the switch error rate, in b) the mismatch rate, and in c) the percentage of heterozygous sites phased over the total number of heterozygous sites.

Figure 3 presents the accuracy obtained by Beagle and μ-PBWT at different mutation rates (detailed results are available in Table 2). The results obtained in the scenario without mutations (i.e., with a mutation rate equal to 0%) are consistent with how the two tools approach the solution of the phasing problem. Beagle is a statistical approach that is not expected to find the exact solution even if the target genotype can be perfectly explained by two haplotypes in the reference panel. In contrast, μ-PBWT implements a combinatorial approach which is designed to always find a haplotype pair that explains the target, if such a pair exists in the reference panel. In fact, in this scenario μ-PBWT correctly identifies the two haplotypes of the target sample, so both *switch error rate* and *mismatch rate* are equal to 0.0%, while Beagle exhibits some fluctuations, with both metrics greater than zero.

As the mutation rate increases, the accuracy of μ-PBWT decreases dramatically. In fact, as we can see in Figure 3c, even with a mutation rate of 5%, μ-PBWT is able to phase only 62.6% of heterozygous sites. We recall that μ-PBWT does not phase heterozygous sites that are not explained by any haplotype pair that can be used as both a left and a right anchor. As a consequence, the higher the mutation rate, the larger the heterozygous genotype regions that do not have valid anchors and thus remain unphased. This fact further demonstrates the dependency of our approach on the data available in the reference panel, which benefits from panels including a large number of haplotypes. In addition to this strong dependency on the reference panel, our approach is unable to estimate haplotype pairs that differ from the ones obtained using the positional cover. Thus, around each mutation site, μ-PBWT will report pairs of haplotypes that differ from the ones of the target sample. However, a manual inspection of the results revealed that one of the two reported haplotypes often coincides with a haplotype of the target sample. However, these cases are handled better by Beagle, again due to its probabilistic approach, which seems to be more robust against these mutations and successfully phases each genotype site.

Figure 4 presents the computational resources used by the two approaches. The running times of the two approaches are comparable. However, the running time of μ-PBWT increases as the mutation rate increases, since, as expected, the number of haplotype switches that μ-PBWT has to compute (instead of simply continuing the match) also increases. Each time the algorithm cannot proceed with any pair, it has to start again, considering larger sets of possible pairs whose analysis increases both the overall execution time and RAM usage. Note that enlarging the number of reference samples would worsen performance in terms of indexing and storing the reference panel, but it would likely reduce the number of possible

Table 2 Comparison of various genotype phasing accuracy metrics (such as the switch error rate and the mismatch rate) of Beagle and μ-PBWT as the mutation rate varies.

(a) Results with 0% and 1% mutation rate.

Mutation rate	0 %		1 %	
	Beagle	μ-PBWT	Beagle	μ-PBWT
Switch error rate	0.0138	0.0	0.003	0.0021
Mismatch rate	0.0038	0.0	0.0	0.0004
Flat rate	0.491	0.0	0.454	0.0121
Phased count	60 604	60 604	60 862	54 547
N50	242 130 407	242 130 407	242 130 407	242 130 407
No. of SNPs max block	60 604	60 604	60 862	54 547

(b) Results with 3% and 5% mutation rate.

Mutation rate	3 %		5 %	
	Beagle	μ-PBWT	Beagle	μ-PBWT
Switch error rate	0.0077	0.0076	0.0128	0.0121
Mismatch rate	0.0007	0.0024	0.0014	0.0042
Flat rate	0.4721	0.0272	0.4754	0.0397
Phased count	61 362	44 365	61 995	38 803
N50	242 130 407	242 130 407	242 065 471	242 065 471
No. of SNPs max block	61 362	44 365	61 995	38 803

(c) Results with 10% and 20% mutation rate.

Mutation rate	10 %		20 %	
	Beagle	μ-PBWT	Beagle	μ-PBWT
Switch error rate	0.0233	0.0286	0.0432	0.0652
Mismatch rate	0.0037	0.0122	0.0092	0.0325
Flat rate	0.4878	0.0745	0.4920	0.1638
Phased count	63 153	27 760	65 265	15 903
N50	242 130 407	242 046 972	242 133 632	242 088 948
No. of SNPs max block	63 153	27 760	65 265	15 903

pairs to consider at each iteration since longer matches would become more likely. On the other hand, the running time of Beagle remains constant as the mutation rate increases since it does not depend on the number of haplotype switches in the reported solution.

As expected, μ-PBWT is very memory-efficient, requiring approximately 7 times less memory than Beagle. These results show that μ-PBWT could be run on low-end hardware or, on the public cloud, at a fraction of the cost of running Beagle. We argue that further refinements and improvements to the model and algorithm would not significantly impact memory requirements, since memory usage in genotype phasing approaches is often dominated by the space required to store the reference panel and μ-PBWT employs a very compact and efficient index representation of the panel.

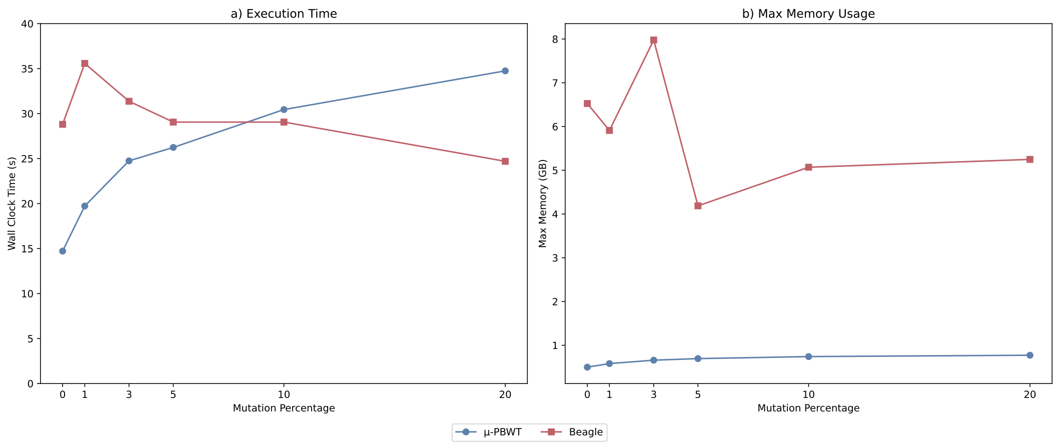

Figure 4 a) Running time and b) peak memory usage (right) of μ-PBWT and Beagle at different mutation rates.

7 Conclusion

In this paper, we present a proof of concept showing how μ-PBWT can be used to solve the genotype phasing problem. First, we outline the theoretical framework underlying our method, based on the foundations presented in [3]. We then introduce our combinatorial approach to tackling the phasing problem, considering both scenarios: (i) when a haplotype pair fully explains the target genotype and (ii) when multiple haplotype pairs are required to cover different genotype regions. Experimental results indicate that our approach is sensitive to the size of the reference panel, particularly when the mutation rate increases.

Improving the resolution of haplotype reconstruction in complex genomic regions remains an important direction for future work. In particular, anchoring heterozygous regions using flanking homozygous segments may enhance stability during phasing, while incorporating methods to detect recombination events within heterozygous regions could improve accuracy in the presence of structural variation and mutation.

These enhancements build on the current strengths of the proposed method, which, in mutation-free settings, reliably identifies the correct haplotype pair. Furthermore, consistent with the evaluations of μ-PBWT, the method achieves high efficiency with minimal memory requirements, making it suitable for execution on commodity hardware. Taken together, these characteristics provide a strong foundation for future comparisons with state-of-the-art phasing tools, particularly in real-world scenarios involving complex variant patterns and sequencing noise.

References

1 Paola Bonizzoni. A linear-time algorithm for the perfect phylogeny haplotype problem. *Algorithmica*, 48:267–285, 2007. doi:10.1007/S00453-007-0094-3.

2 Paola Bonizzoni, Christina Boucher, Davide Cozzi, Travis Gagie, Dominik Köppl, and Massimiliano Rossi. Data Structures for SMEM-Finding in the PBWT. In *International Symposium on String Processing and Information Retrieval*, pages 89–101. Springer, 2023. doi:10.1007/978-3-031-43980-3_8.

3 Paola Bonizzoni, Christina Boucher, Davide Cozzi, Travis Gagie, and Yuri Pirola. Solving the minimal positional substring cover problem in sublinear space. In *35th Annual Symposium*

on Combinatorial Pattern Matching (CPM 2024). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.

4 Paola Bonizzoni, Gianluca Della Vedova, Riccardo Dondi, and Jing Li. The haplotyping problem: an overview of computational models and solutions. *Journal of Computer Science and Technology*, 18:675–688, 2003. doi:10.1007/BF02945456.

5 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.

6 Brian L Browning, Ying Zhou, and Sharon R Browning. A one-penny imputed genome from next-generation reference panels. *The American Journal of Human Genetics*, 103(3):338–348, 2018.

7 Mark JP Chaisson, Ashley D Sanders, Xuefang Zhao, Ankit Malhotra, David Porubsky, Tobias Rausch, Eugene J Gardner, Oscar L Rodriguez, Li Guo, Ryan L Collins, et al. Multi-platform discovery of haplotype-resolved structural variation in human genomes. *Nature communications*, 10(1):1784, 2019.

8 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. doi:10.1093/BIOINFORMATICS/BTAD552.

9 Richard Durbin. Efficient haplotype matching and storage using the positional Burrows–Wheeler transform (PBWT). *Bioinformatics*, 30(9):1266–1272, 2014. doi:10.1093/BIOINFORMATICS/BTU014.

10 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.

11 Peter Edge, Vineet Bafna, and Vikas Bansal. Hapcut2: robust and accurate haplotype assembly for diverse sequencing technologies. *Genome research*, 27(5):801–812, 2017.

12 Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings 41st annual symposium on foundations of computer science*, pages 390–398. IEEE, 2000. doi:10.1109/SFCS.2000.892127.

13 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.

14 Dan Gusfield. Haplotyping as perfect phylogeny: conceptual framework and efficient solutions. In *Proceedings of the sixth annual international conference on Computational biology*, pages 166–175, 2002. doi:10.1145/565196.565218.

15 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.

16 Na Li and Matthew Stephens. Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data. *Genetics*, 165(4):2213–2233, 2003.

17 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. doi:10.1093/BIOINFORMATICS/BTZ347.

18 Simone Rubinacci, Olivier Delaneau, and Jonathan Marchini. Genotype Imputation using the Positional Burrows Wheeler Transform. *PLoS Genetics*, 16(11):e1009049, 2020.

19 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.

20 Ahsan Sanaullah, Degui Zhi, and Shaoije Zhang. Haplotype threading using the positional Burrows-Wheeler transform. In *22nd International Workshop on Algorithms in Bioinformatics (WABI 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

21 Ahsan Sanaullah, Degui Zhi, and Shaojie Zhang. d-pbwt: dynamic positional burrows–wheeler transform. *Bioinformatics*, 37(16):2390–2397, 2021. doi:10.1093/BIOINFORMATICS/BTAB117.

22 Ahsan Sanaullah, Degui Zhi, and Shaojie Zhang. Minimal positional substring cover is a haplotype threading alternative to Li and Stephens Model. *Genome Research*, 33(7):1007–1014, 2023. doi:10.1101/gr.277673.123.

23 The International Hapmap 3 Consortium. Integrating common and rare genetic variation in diverse human populations. *Nature*, 467(7311):52–58, September 2010. doi:10.1038/nature09298.

24 Victor Wang, Ardalan Naseri, Shaojie Zhang, and Degui Zhi. Syllable-PBWT for space-efficient haplotype long-match query. *Bioinformatics*, 39(1):btac734, 2023. doi:10.1093/BIOINFORMATICS/BTAC734.

25 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. *medRxiv*, pages 2024–03, 2024.

26 William Yue, Ardalan Naseri, Victor Wang, Pramesh Shakya, Shaojie Zhang, and Degui Zhi. P-smoother: efficient PBWT smoothing of large haplotype panels. *Bioinformatics Advances*, 2(1):vbac045, 2022.