Bioinformatics, 35(5), 2019, 798–806
doi: 10.1093/bioinformatics/bty735
Advance Access Publication Date: 25 August 2018
Original Paper

OXFORD

Genetics and population analysis

Haplotype matching in large cohorts using the Li and Stephens model

Gerton Lunter

University of Oxford, Wellcome Centre for Human Genetics, Oxford OX3 7BN, UK

Associate Editor: Oliver Stegle

Received on March 7, 2018; revised on May 16, 2018; editorial decision on August 18, 2018; accepted on August 23, 2018

Abstract

Motivation: The Li and Stephens model, which approximates the coalescent describing the pattern of variation in a population, underpins a range of key tools and results in genetics. Although highly efficient compared to the coalescent, standard implementations of this model still cannot deal with the very large reference cohorts that are starting to become available, and practical implementations use heuristics to achieve reasonable runtimes.

Results: Here I describe a new, exact algorithm ('fastLS') that implements the Li and Stephens model and achieves runtimes independent of the size of the reference cohort. Key to achieving this runtime is the use of the Burrows-Wheeler transform, allowing the algorithm to efficiently identify partial haplotype matches across a cohort. I show that the proposed data structure is very similar to, and generalizes, Durbin's positional Burrows-Wheeler transform.

Contact: gerton.lunter@well.ox.ac.uk

1 Introduction

The genetic variation in a population of interbreeding individuals is highly structured. Kingman (1982) introduced the canonical model that describes this structure mathematically, known as Kingman's coalescent, later extended by Hudson (1983) and Griffiths and Marjoram (1997) to include recombination. Although mathematically elegant, it is challenging to use these models directly for statistical inference. Li and Stephens (2003) introduced a model (LS) that is both a good approximation to the coalescent with recombination, and computationally tractable. As a result, LS now underpins a large range of key tools and scientific findings (Beaumont, 2010; Howie *et al.*, 2009; The International HapMap Consortium, 2005; The Wellcome Trust Case Control Consortium, 2007). Depending on whether the input sequence is haploid or diploid, LS in its straightforward implementation as a hidden Markov model (HMM) runs in linear or quadratic time in the number of reference haplotypes. While this is orders of magnitude more efficient than algorithms based directly on Kingman's coalescent or the ARG, the recent availability of affordable DNA sequencing technology has provided access to very large reference sets, on which even the LS model is intractable in its standard implementation, so that current implementations of LS use heuristics to cope with datasets encountered in practice (Howie *et al.*, 2009).

A very different algorithm that is making an impact in genomics was introduced by Burrows and Wheeler (1994). Known as the Burrows-Wheeler transform (BWT), it permutes an arbitrary text in such a way that the original text can be recovered, while at the same time improving the compressibility of the transformed text by increasing simple repetitions. In addition, the transformed text, even in compressed form, serves as an index that allows rapid searching in the original text. In genomics this idea has so far been used mainly for fast alignment of short reads against a large and relatively repetitive reference genome (Langmead *et al.*, 2009; Li and Durbin, 2009). More recently, Durbin (2014) introduced a variant of the BWT, termed the Positional Burrows-Wheeler Transform (PBWT), that exploits the additional structure that exists in a set of haplotypes in a population sample. These data, which are usually encoded as a series of 0 and 1 s representing the absence or presence in a sample of particular genetic variants along a reference sequence, have a natural representation as a matrix, where rows represent samples and columns represent the particular positions in a reference. Local matches between samples are only relevant at matching positions, and exploiting this restriction leads to improvements over a standard application of the BWT. The resulting data structure again allows for fast haplotype searches against a database, and achieves very high compression ratios.

2 Approach

There are two main results in this paper. First, I establish a formal connection between the standard and positional BWT, showing how the PBWT as introduced in Durbin (2014) is a special case of the BWT. This connection also shows how the PBWT can be slightly generalized to cope with the multiallelic case. Besides providing an additional perspective on the positional BWT algorithms, which helps to better understand them, it also provides a mechanical way to 'lift' existing algorithms operating on the BWT data structure to their positional equivalent, allowing the large literature on BWT algorithms to be applied to the current data structure. I show how this works by deriving the haplotype search algorithm from the equivalent BWT algorithm.

The second contribution consist of algorithms that implement the LS model on top of the BWT. More precisely, I present algorithms that compute maximum-likelihood ('Viterbi') paths through the LS hidden Markov model, providing a parsimonious description of a given sequence as an imperfect mosaic of reference haplotypes. The ability to efficiently identify matches in the database of reference haplotypes result in considerable improvements in runtime over the standard implementation, reducing the linear and quadratic asymptotic runtime to empirical constant time, independent of the number of reference haplotypes. More precisely, for H samples of n loci each, the standard implementation runs in $O(Hn)$ time for a haploid input sequence, and $O(H^2n)$ for a diploid input sequence, while the proposed algorithms run in empirical $O(n)$ time in both cases. This allows the Li and Stephens model to be used on very large reference panels, without recourse to approximations.

3 Materials and methods

3.1 Haplotype matching using the BWT

Let $x_0,\dots,x_{H-1}$ be H haplotype sequences, each consisting of n symbols from the alphabet A representing the possible allelic states at a locus; for simplicity I will often use $A=\{0,1\}$ in this paper. A straightforward way of identifying haplotype matches would be to use the BWT on the concatenation $x_0x_1\cdots x_{H-1}$ of haplotype sequences. It turns out that a more efficient algorithm is obtained, in terms of time and memory use, by embedding this sequence of Hn characters into a sequence of $2Hn$ characters taken from a much larger alphabet. The increase in sequence length and alphabet size is offset by the additional structure in the BWT that results from the chosen embedding. This in turn translates into better compression and a streamlined search algorithm.

I will write $x[j]$ for the jth symbol in the sequence x, and $x[j,k)$ for the subsequence starting at position j and ending at $k-1$. I will also use $[i,j)$ to denote the half-open interval $\{i,i+1,\dots,j-1\}$, and if M_{ij} is a matrix, $M_k[i,j)$ is the subsequence $M_{k,i},M_{k,i+1},\dots,M_{k,j-1}$ of the kth row of the matrix. Throughout this paper, all indices start at 0.

Let $\mathbf{p}_0,\dots,\mathbf{p}_{n-1}$ be n additional symbols in the alphabet, ordered such that $\mathbf{p}_0<\cdots<\mathbf{p}_{n-1}<0<1$. Introduce a new sequence X of length $2Hn$ by inserting a symbol $\mathbf{p}_j$ after each symbol $x_i[j]$ and concatenating the resulting sequences into a single sequence of the form

$$X=\begin{array}{cccccccc} x_0[0] & \mathbf{p}_0 & x_0[1] & \mathbf{p}_1 & \cdots & x_0[n-1] & \mathbf{p}_{n-1} \\ x_1[0] & \mathbf{p}_0 & x_1[1] & \mathbf{p}_1 & \cdots & x_1[n-1] & \mathbf{p}_{n-1} \\ \vdots & & & & & & \vdots \\ x_{H-1}[0] & \mathbf{p}_0 & \cdots & & & x_{H-1}[n-1] & \mathbf{p}_{n-1} \end{array} \quad (1)$$

Algorithm 1 Calculating BWT(X)

Input: sequences $x_0,\dots,x_{H-1}$, each of length n; alphabet A
Output: Block permutations $j\mapsto a_j^i$, $i=0,\dots,n-1$.
$i\leftarrow n$; $a_j^{n-1}\leftarrow j$ for $j\in[0,H)$
While $i>0$:
  $i\leftarrow i-1$; $t_c=\sum_{u<c}f_i^u \quad (c\in A)$
  For j in $[0,H)$:
    $c\leftarrow x_{a_j^i}[i]$
    $a_{t_c}^{i-1}\leftarrow a_j^i$; $t_c\leftarrow t_c+1$

Input: sequences $x_0,\dots,x_{H-1}$, each of length n; alphabet $A=\{0,1\}$
Output: Block permutations $j\mapsto a_j^i$, $i=0,\dots,n-1$.
$i\leftarrow n$; $a_j^{n-1}\leftarrow j$ for $j\in[0,H)$
while $i>0$:
  $i\leftarrow i-1; t\leftarrow 0; u\leftarrow f_i^0$
  For j in $[0,H)$:
    If $x_{a_j^i}[i]=0$:
      $a_t^{i-1}\leftarrow a_j^i$; $t\leftarrow t+1$
    Else:
      $a_u^{i-1}\leftarrow a_j^i$; $u\leftarrow u+1$

(To impose a particular initial ordering I will later on replace the last symbol $\mathbf{p}_{n-1}$ by H symbols $\mathbf{p}_{n-1}^0<\cdots<\mathbf{p}_{n-1}^{H-1}$, but to avoid cluttering the notation I ignore this detail for now.) Consider all cyclic shifts $X^k=X[k]X[k+1]\cdots X[2Hn-1]X[0]\cdots X[k-1]$ of X. Let M be the matrix obtained by writing X^k on the kth row of a square matrix, and sorting the resulting rows lexicographically. Let π be the permutation that sorts the rows, so that $X^{\pi(0)}<X^{\pi(1)}<\cdots<X^{\pi(2Hn-1)}$, and $M_{ij}=X^{\pi(i)}[j]$. The Burrows-Wheeler transform of X is the last column of this matrix: $BWT(X)[i]=X^{\pi(i)}[2Hn-1]$. Note that this is almost the traditional BWT of the sequence X, except that there is no special 'end' character. This character is used to identify the start of the sequence; here, the special structure of X is sufficient to navigate $BWT(X)$.

Now consider how the matrix M may be constructed. The position symbols $\mathbf{p}_i$ determine the coarse structure of M, which is independent of the data x_i apart from the haplotype frequencies f_i^0 and f_i^1 (see Fig. 1). The fine-scale structure of M within each 'block' of H rows is determined by the data. More precisely, rows in the block starting at index iH are those cyclic shifts of X that start with symbol $\mathbf{p}_i$ and end with $x_k[i]$ for some $k\in[0,H)$, such that these rows are ordered lexicographically within the block. Let $j\mapsto a_j^i$ denote the permutation of $[0,H)$ that describes this order within block i, so that row $iH+j$ ends with symbol $x_{a_j^i}[i]$. Determining M therefore boils down to determining the n permutations a_j^i for $i\in[0,n)$, since these determine the top half of M, and those in turn determine the remaining rows (see Fig. 1 and the explanation).

The permutations a_j^i are determined recursively, working from $i=n-1$ backwards. Because we imposed the special ordering $\mathbf{p}_{n-1}^0<\cdots<\mathbf{p}_{n-1}^{H-1}$ on the final position symbols, the permutation for block $n-1$ is given by the identity permutation $j\mapsto a_j^{n-1}=j$. Now suppose the permutation a_j^i for block i has been determined. The sequences in block $i-1$ are formed from those in block i by moving two characters from the end to the front. The first character in any sequence of this new block is $\mathbf{p}_{i-1}$, which does not influence the ordering within the block. The second character is an allele marker $x_{a_j^i}[i]$. To sort the sequences in block $i-1$ in lexicographic

Algorithm 2 General subsequence search

Input: Sequence $w[0,j)$, BWT(X) of sequence $X[0,n)$
Output: Indices s, e such that $M_k[0,j) = w$ for $k \in [s,e)$
1: $s \leftarrow 0$, $e \leftarrow n$, $i \leftarrow j$
2: While $s < e$ and $i > 0$: ▷ $w[i,j)$ matches $M_k[0,j-i)]$ for $k \in [s,e)$
3: $\quad i \leftarrow i-1$
4: $\quad s \leftarrow C(w[i]) + R^{w[i]}(s)$
5: $\quad e \leftarrow C(w[i]) + R^{w[i]}(e)$

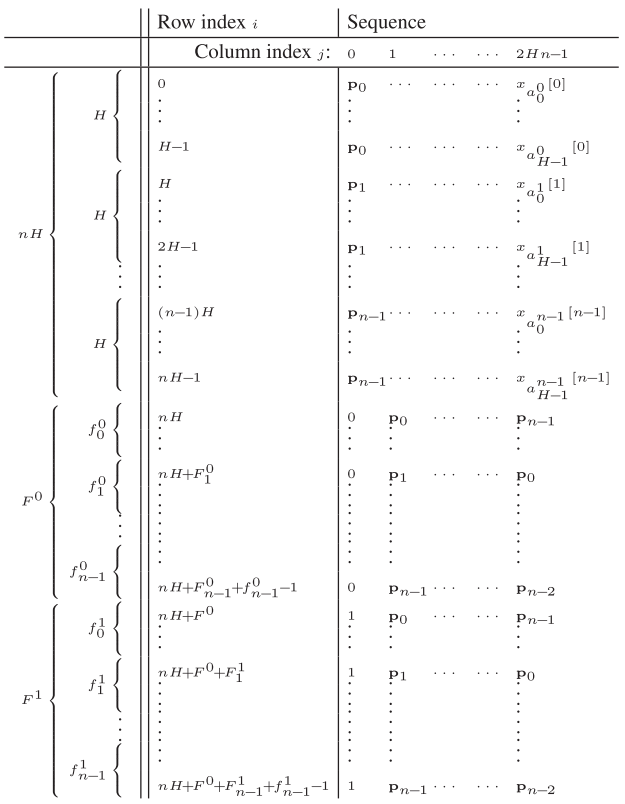

Fig. 1. Structure of the matrix M_{ij}. The rows M_i are sorted lexicographically; in particular $\mathrm{p}_0 < \mathrm{p}_1 < \cdots < 0 < 1$. The Burrows-Wheeler transform of X (see text) is the rightmost column of M, while the positional BWT of the sequences $x_0, \ldots, x_{H-1}$ is the upper half of the same column (see text). The column indices are determined by f_i^a, the allele frequency of symbol a at locus i, and $F_i^a := \sum_{j=0}^{i} f_i^a$, the cumulative frequency of symbol a across loci $0, \ldots, i$. Note that ordering of rows $(n-1)H$ to $Hn-1$ is determined by the special position symbols $\mathrm{p}_{n-1}^0 < \cdots < \mathrm{p}_{n-1}^{H-1}$, but to avoid cluttering the notation these are all written as p_{n-1}

order, it is therefore sufficient to list those sequences that start a 0 symbol first, followed by those starting with a 1 symbol (followed by other symbols if the locus is multiallelic), and otherwise leave the original order undisturbed. Doing this results in Algorithm 1.

To show that the proposed construction is equivalent to the positional Burrows-Wheeler transform, Algorithm 1 is given both for general alphabets A and specialized for the case $A = \{0,1\}$, since that in that case the inner loop is precisely Algorithm 1 in Durbin (2014) (except that the proposed algorithm runs back-to-front, as is usual for BWT algorithms). As in the PBWT algorithm, the permutations a_j^i play the role of the suffix array in the ordinary BWT algorithm. Note that the output includes a permutation a_j^{-1}, which encodes how the very first characters $x_j[0]$ influence the permutation of the cyclic shifts X^k; this permutation is used in Algorithm 5. Following Durbin (2014) I now define the PBWT of $x_0, \ldots, x_{H-1}$ as the first half of $BWT(X)$, which is availably implicitly as $BWT(X)[Hi+j] = x_{a_j^i}[i]$. Figure 1 shows that the second half of $BWT(X)$ is determined by the allele frequencies f_i^c, $i \in [0,n)$, which can be computed easily from the relevant block in the first half of $BWT(X)$, so that the PBWT of $x_0, \ldots, x_{H-1}$ is in fact equivalent to $BWT(X)$.

3.2 Substring searching

Algorithm 1 calculates $BWT(X)$ in linear time by exploiting the special structure of X, and is not a specialization of an existing, general algorithm to calculate the BWT. By contrast, Algorithm 2, which performs a substring search, can be derived directly from its analogous algorithm for a general BWT.

To describe the algorithm, let M be the sorted matrix of cyclic shifts of an arbitrary sequence X of length n, so that $BWT(X)[i] = M_i[n-1]$, and let $R^a(i)$ (the 'a-rank' for row i) be the number of times that a appears in $BWT(X)[0,i)$. This function can be calculated efficiently from $BWT(X)$, particular if the data is stored in compressed form. Finally, let $C(a)$ (the cumulative symbol frequency) be the number of symbols in X that are less than a. This notation makes it possible to write down Algorithm 2, for substring searching. (The symbol ▷ is used throughout to mark comments and invariants in the algorithms.)

To understand the algorithm, consider all rows of M that end with a symbol a. If these rows are cyclically shifted rightward, so that the last symbol becomes the first and all others are moved one position to the right, all rows will now start with a, and the relative order in which they appear in M (which they must as M contains all cyclic shifts of X) is the same as before the shift since they were ordered lexicographically to start with. Suppose that M_k is a row that ends with a, and that after right-shifting it ends up as row $M_{k'}$; then the above observation means that the rank $R^a(k)$ of the symbol a in M_k in the last column of M, is the same as the rank in the first column of M of the symbol a in $M_{k'}$. Because M is sorted lexicographically, the rows that start with a form a contiguous block in M, so that the first-column rank of the symbol a in row $M_{k'}$ is $k' - C(a)$, so that $R^a(k) = k' - C(a)$ or

$$LF(k,a) := k' = C(a) + R^a(k) \tag{2}$$

The function $k \mapsto LF(k,a)$, mapping row k to the row corresponding to its right-shifted counterpart k', is called the *last-to-first* mapping because it maps the last (rightmost) symbol of M_k to the corresponding symbol in the first (leftmost) position of $M_{k'}$. It is repeatedly used to identify the interval of rows corresponding to sequences that match one additional character of w.

Note that the mapping is well-defined whether or not $M_k[n-1] = a$. This makes it possible to think of k as representing a possible location between two entries (k and $k-1$) in M where a sequence (or sequence prefix) x not necessarily represented in M would be inserted; this is the view taken in the search algorithm. Alternatively, when k is thought of as a particular row in M, that row's initial character a can be obtained from the $C(\cdot)$ function, and since the mapping (2) is invertible when restricted to the set of rows k ending in a, this makes the mapping $k \mapsto LF(k, M_k[n-1])$ invertible for all k. The existence of this inverse mapping also follows

Algorithm 3 PBWT subsequence search

Input: Sequence $w[0,j)$, PBWT of $x_0,\ldots,x_{H-1}$
Output: Indices s, e such that $x_{a_k^0}[0,j) = w$ for $k \in [s,e)$
1: $s \leftarrow 0$, $e \leftarrow H$, $i \leftarrow j$
2: While $s < e$ and $i > 0$: ▷ $w[i,j)$ matches $x_{a_k^i}[i,j)$ for $k \in [s,e)$
3: $i \leftarrow i-1$
4: $s \leftarrow LF(s, w[i], i)$ ▷ see equation (3)
5: $e \leftarrow LF(e, w[i], i)$

directly from the observation that it corresponds to rotating the sequence one position leftward; it could be called the *first-to-last mapping*, $k \mapsto FL(k)$, and is used in Algorithm 5.

To derive the corresponding algorithm for matching a sequence in the PBWT data structure, it is enough to track the bounding variables for two steps through the standard BWT algorithm acting on the 'lifted' sequence X, matching a haplotype character and a position character. The first step identifies the new range depending on the haplotype character to be matched, and points these variables to the second half of the matrix. The next step moves the bounding variable back into the first half by moving a position character in front. Because of the regular form of $BWT(X)$ (see Fig. 1), these two steps can be followed algebraically and combined into a single update step. The derivation, which is straightforward but requires additional notation, is presented in the Appendix. The resulting combined update step is given by a modified last-to-first mapping function, which now additionally depends on the current position i:

$$LF(k,a,i) := k' = \begin{cases} r_i^0(k) & \text{if } a = 0 \\ f_i^0 + k - r_i^0(k) & \text{if } a = 1, \end{cases} \tag{3}$$

or for an arbitrary alphabet, $LF(k,a,i) = r_i^a(k) + \sum_{c<a} f_i^c$. Here $r_i^a(k)$ is the positional analogue of $R^a(i)$, and counts how often a appears in the first k rows of the ith block of $PBWT(x_0,\ldots,x_{H-1})$, or equivalently, in $BWT(X)[Hi, Hi+k) = x_{a_0^i}[i],\ldots,x_{a_{k-1}^i}[i]$, and f_i^a is the (haplotype) frequency of a at position i. This leads to Algorithm 3.

3.3 Haploid Li and Stephens

The Li and Stephens (2003) model approximates the coalescent model describing the relationship between DNA sequences in a population, by generating a new sequence as a mosaic of imperfect copies of existing sequences The popularity of the model stems from the fact that it is both a good approximation to the full coalescent model with recombination, as well as fast to compute in its natural implementation as a hidden Markov model, running in $O(Hn)$ time for H sequences of length n. However, for very large population samples this is still too slow in practice.

Here I describe an algorithm to compute the maximum likelihood path through the LS hidden Markov model (HMM) in empirical $O(n)$ time. The approach is not to consider single sequences to copy from, but *groups* of sequences that share a common subsequence. Like the Viterbi algorithm for HMMs, the proposed algorithm traverses the sequence to be explained, but rather than using a dynamic programming approach, it uses a branch-and-bound approach considering (groups of) potential path prefixes to a maximum likelihood path. Where at each iteration the Viterbi algorithm must consider all possible sequences that a potential path prefix

Algorithm 4 Haploid Burrows-Wheeler Li and Stephens

Input: Sequence $x[0,n)$, PBWT of $x_0,\ldots,x_{H-1}$, scores $\mu \geq 0$, $\rho \geq 0$.
Output: Minimum path score under the Li and Stephens model
1: $i \leftarrow n$; $st \leftarrow [(0,H,0)]$; $gm \leftarrow 0$;
$traceback \leftarrow [(n-1,-1,-1)]$
2: While $i > 0$: ▷ st represent states of paths in full suffix set for $x[i,n)$
3: $i \leftarrow i-1$; $st' \leftarrow []$; $gm' \leftarrow gm + \mu$; $extended \leftarrow False$
4: For (s, e, $score$) in st:
5: If $score < gm + \rho$:
6: $s' \leftarrow LF(s, x[i], i)$; $e' \leftarrow LF(e, x[i], i)$
7: If $s' < e'$:
8: $st'.append((s', e', score))$
9: $gm' \leftarrow \min(gm', score)$
10: If $score = gm$: $extended \leftarrow True$
11: If $score + \mu < gm' + \rho$:
12: $s' \leftarrow LF(s, 1-x[i], i)$; $e' \leftarrow LF(e, 1-x[i], i)$
13: If $s' < e'$: $st'.append((s', e', score + \mu))$
14: $s' \leftarrow LF(0, x[i], i)$; $e' \leftarrow LF(H, x[i], i)$
15: If $s' < e'$ and $extended = False$: ▷ Never true on 1st iteration
16: $st'.append((s', e', gm + \rho))$
17: $traceback.append((i, gm\_idx, gm + \rho))$
18: $gm \leftarrow gm'$; $st \leftarrow st'$
19: $gm\_idx \leftarrow$ any of $\{s | (s,e,score) \in st$ and $score = gm\}$
20: Return gm, gm_idx, $traceback$

Algorithm 5 Haploid traceback

Input: Sequence $x[0,n)$, PBWT of $x_0,\ldots,x_{H-1}$, scores $\mu \geq 0$, $\rho \geq 0$, minimum score gm, corresponding index gm_idx, traceback list $traceback$.
Output: Representation $path$ of a minimum-scoring path
1: Function $FL(k, i)$: ▷ "First-to-last" mapping
2: $lo \leftarrow 0$; $hi \leftarrow H$; $a \leftarrow 0$ if $k < f_i^0$ else 1
3: While $lo < hi$: ▷ $LF(j,a,i) \leq k\ \forall j < lo$ and $LF(j,a,i) > k\ \forall j \geq hi$
4: $mid \leftarrow \lfloor (lo + hi)/2 \rfloor$
5: If $LF(mid, a, i) \leq k$: $lo \leftarrow mid + 1$
6: Else: $hi \leftarrow mid$
7: Return $a, lo - 1$
8: $i \leftarrow 0$; $path \leftarrow [(i, a_{gm\_idx}^{i-1})]$
9: For $(t\_locus, t\_idx, t\_score)$ in $reverse(traceback)$:
10: While $i \leq t\_locus$:
11: $a, gm\_idx \leftarrow FL(gm\_idx, i)$
12: If $a \neq x[i]$: $gm \leftarrow gm - \mu$
13: $i \leftarrow i + 1$
14: If $gm = t\_score$:
15: $gm\_idx \leftarrow t\_idx$; $gm \leftarrow gm - \rho$;
$path.append((i, a_{gm\_idx}^{i-1}))$
16: Return $path$

could end with, the proposed algorithm in principle considers all extensions of the current potential path prefixes (the 'branch' part), but ignores prefixes that cannot be part of an optimal path (the 'bound' part). For instance, if a prefix can be extended with a matching nucleotide, a recombination does not have to be considered, since the recombination can be postponed at no cost. Below I will show this more formally. This formal approach is perhaps not necessary (or even helpful) for the haploid case, but becomes useful when I introduce the diploid Li and Stephens algorithm.

First some definitions. A *placed character* is a character c at a sequence position i; it is equivalent to a pair $c\mathbf{p}_i$ where $\mathbf{p}_i$ is the position symbol introduced before. Two placed characters are *contiguous* if they occupy neighbouring positions; subsequences of placed characters are contiguous if every pair of neighbouring characters is; and two or more subsequences are contiguous if their concatenation is. A *path* π of m *parts* through a set of sequences $\Omega = \{x_0, \ldots, x_{H-1}\}$ is a contiguous sequence of m subsequences $s_0, \ldots, s_{m-1}$ such that each s_i is a subsequence of some x_j. I will write a path as

$$\pi = (c_0 c_1 \cdots c_{k_0-1} R c_{k_0} \cdots c_{k_1-1} R c_{k_1} \cdots\cdots R c_{k_{m-2}} \cdots c_{l-1})$$

where c_i is a character placed at position i, and $k_0, k_1, \ldots, k_{m-2}$ are the *recombination breakpoints* identified by the symbol R (which is not part of the alphabet), and l is the *length* of the path. The (sequence) *group* associated with π is the set $G(\pi)$ of all sequences $x \in \Omega$ for which the subsequences $x[k_{m-2}, l)$ agree with the suffix $c_{k_{m-2}} \cdots c_{l-1}$ that follows the last recombination in π. The *extension* πc_l (of length $l+1$) is the path $(c_0 \cdots R c_{k_{m-2}} \cdots c_{l-1} c_l)$, if it exists; since by definition all subsequences that make up a path are subsequences of some x_j, existence of an extension implies that its group is nonempty. The extension πR (of length l) is defined as $(c_0 \cdots R c_{k_{m-2}} \cdots c_{l-1} R)$, and always exists; its group is Ω. Finally, the *path prefix* $\pi[0, t)$ is the path $(c_0 \cdots c_{t-1})$ including any R symbols for recombinations between positions 0 and $t-1$; a path prefix never ends with an R symbol.

For a given sequence x and a path π, the Li and Stephens model assigns a joint likelihood to the event that π occurred and gave rise to sequence x. If π has m parts and has k mismatches to x, this likelihood is

$$\begin{aligned} p(\pi, x) &= p_\rho^m (1 - n p_\rho)^{H-m} p_\mu^k (1 - 3p_\mu)^{H-k} \\ &= \left(\frac{p_\rho}{1 - n p_\rho}\right)^m \left(\frac{p_\mu}{1 - 3p_\mu}\right)^k (1 - n p_\rho)^H (1 - 3p_\mu)^H \end{aligned}$$

where p_ρ is the probability of recombining into a particular other sequence, and p_μ is the probability of a mutation to one of the three other nucleotides. The negative log likelihood takes a particularly simple form,

$$-\log p(\pi, x) = m\rho + k\mu + C,$$

where C is a constant, $\rho = -\log(p_\rho/(1 - n p_\rho))$ and $\mu = -\log(p_\mu/(1 - 3p_\mu))$. This motivates defining the *path score* as $s_x(\pi) = m\rho + k\mu$, where m and k are defined as above. I drop the subscript x from $s_x(\pi)$ when this is possible without creating confusion.

Suppose we want to calculate a path π that minimizes $s(\pi)$. This can be done by iteratively constructing path prefixes π', so that at each step one of them is a prefix of a full path π that minimizes $s(\pi)$. Note that the minimum score achievable by a path π that has π' as its prefix depends on the prefix score $s(\pi')$ and the prefix group $G(\pi')$, but not on the rest of the prefix. This is because $G(\pi')$ is the set of sequences the Li and Stephens model could be copying from at the end of π', and the Markov property of the model implies that the minimum score only depends on the sequence being copied from (and the prefix score). This justifies the definition of *state* of a path (prefix) π' to be the pair $(G(\pi'), s(\pi'))$.

The key observation for the algorithm is that some states (G, s) can be ignored, because any of their extensions give rise to paths and scores that are also achievable via other states. To make this precise I need one more definition. A set S of path prefixes, all of length l, is a *full prefix set* for $x[0, l)$ if for any sequence x' whose prefix $x'[0, l)$ agrees with $x[0, l)$, there exists a path π that achieves the minimum score (i.e. $s_{x'}(\pi) = \min_{\pi'} s_{x'}(\pi')$) and whose prefix $\pi[0, l)$ is in S. If we can somehow find a way to iteratively construct full prefix sets of increasing length, the problem of finding a minimum-score path is solved, because the required path will be an element of the full prefix set for the full-length sequence x. The following theorem shows how to do this:

THEOREM 1. *Suppose S is a full prefix set for $x[0, l)$, S' a set of prefixes of length $l+1$, and let $s_{min} = \min_{\pi \in S} s(\pi)$ and $s'_{min} = \min_{\pi \in S'} s(\pi)$. Then S' is a full prefix set for $x[0, l+1)$ if the following conditions hold:*

a. *a For all $\pi \in S$ and all $a \in \{0, 1\}$ so that πa is an extension and $s(\pi a) < s'_{\min} + \rho$ we have $\pi a \in S'$; and*

b. *b If there is no $\pi \in S$ so that $s(\pi) = s_{\min}$ and $\pi x[l]$ is an extension, then S' contains a path of the form $\pi R x[l]$ with $s(\pi) = s_{\min}$.*

In other words, certain extensions are *not* required to be in S': extensions πa whose score exceed the minimum plus ρ can be left out (since a recombination from the minimum-scoring prefix would give a path that is at least as good), and recombinations can be ignored altogether as long as any current lowest-scoring path has a matching extension (since otherwise postponing the recombination would again be at least as good) – and if not, only a single recombination from a lowest-scoring path needs to be considered.

Algorithm 4 implements these ideas. It does not actually construct prefix sets of paths, but sets of *states* of paths in prefix sets. This is sufficient since the state determines how paths can be extended. By using the PBWT, these states can be represented efficiently, using just the score and a pair of indices into the PBWT that correspond to a set of subsequence matches to sequences in Ω, similar to how the variables s and e in Algorithm 3 represent the interval $[s, e)$ corresponding to a set of subsequence matches. Another difference with the description above is that the algorithm scans the sequence back-to-front, extending partial matches leftward, so that the invariant refers to the full *suffix* set, rather than the full prefix set.

The algorithm computes $gm = s_{\min}$, and keeps a running minimum score gm' that bounds $s'_{\min}$, ignoring states whose new score are not less than $gm' + \rho$. At the end of an iteration, states whose score are not lower than the now updated gm' plus ρ are not immediately removed, but are instead ignored in the next iteration. The algorithm implicitly considers both score bounds implied by gm and gm', but in each situation uses only the tighter bound of the two to decide which states to ignore.

It is possible for different paths to result in overlapping or identical states, resulting in duplicate or otherwise redundant entries in the *st* array. Although redundant entries do not impact the correctness of the algorithm, they can dramatically reduce efficiency. A practical implementation therefore includes a step that occasionally removes redundant states.

The algorithm can be generalized a little by allowing the mutation score $\mu \geq 0$ to depend on the position. The path score is then

defined as $s(\pi) = m\rho + \sum_{i:x[i]\neq\pi[i]} \mu_i$. Theorem 1 continues to hold, and so does Algorithm 4, with the obvious changes. The current approach does not lend itself easily to generalize to a position-dependent recombination probability, as the proof of Theorem 1 relies on delaying the recombination without changing the score, which is only possible if ρ is constant along the sequence.

Note that the algorithm can be simplified when $\mu_i \geq 2\rho$, because a mismatch can always be circumvented by two recombinations (before and after the offending locus), so that only exact matches need to be considered. In human genetics polymorphisms are sparse, and recombinations can only be localized to within hundreds or thousands of positions. Even when a maximum likelihood path is sought it is natural to marginalize over these positions, and this makes the probability of a recombination between two polymorphic sites at least an order of magnitude higher than the probability of a mutation, so that $\mu \gg \rho$. However, in the presence of phasing errors the probability of a mismatch can be much higher than that of a mutation, so that the regime $\mu < 2\rho$ is of practical importance.

Algorithm 4 only computes the optimal score, and to obtain an optimal-scoring path π itself a backtracking step is needed (Algorithm 5). Here it is useful that Algorithm 4 works in the backward direction, so that the result of the backtracking is oriented in the natural direction. To track an optimal path along a sequence, the PBWT index corresponding to that sequence can be tracked using the 'first-to-last' mapping, inverting the steps in lines 6 and 12 in Algorithm 4, and the minimum score of the remaining suffix is updated whenever a difference between this sequence and x is found. Recombinations are followed greedily, as it is always correct to follow a feasible recombination, and it is never clear whether a particular recombination is the last feasible one for a particular sequence. Algorithm 4 collects information about recombinations in the *traceback* list, and when a recombination and score is identified that forms a feasible suffix to the path so far, it is followed.

The naive implementation of Algorithm 5 is somewhat slower than the haploid Li and Stephens algorithm itself, due to the *FL* function which takes $O(\log H)$ time in the implementation shown. In practice the *PBWT* will be stored in compressed form using run-length encoding, which allows a faster implementation of *FL*.

3.4 Diploid Li and Stephens

Where the haploid Li and Stephens algorithm computes a single haplotype path maximizing the probability of a given haploid sequence, the diploid Li and Stephens algorithm aims to find a *pair* of haplotype paths that maximizes the probability of a sequence of diploid *genotypes* under the same model. The approach used to derive the haploid algorithm also works in this case, but the details are more involved.

Let x be a sequence of genotypes, encoded as values 0, 1 or 2 at each position representing homozygous ancestral, heterozygous and homozygous derived genotypes. The aim is to compute a pair of paths α, β that minimizes a score. As before this score contains terms for recombinations and mismatches, but the mismatch term now considers genotypes rather than haplotypes. More precisely, the score associated to the pair $\{\alpha, \beta\}$ is defined as $s(\alpha, \beta) = \rho m(\alpha) + \rho m(\beta) + \mu k(\alpha, \beta)$, where $m(\alpha)$ represents the number of parts of path α, as before, and $k = \sum_i |\alpha[i] + \beta[i] - x[i]|$ counts the number of mismatches of the paths α and β to the genotype sequence x.

The approach of the algorithm is similar to the haploid case, again sequentially building full prefix sets for ever longer sequence prefixes until a minimum path pair is found. To describe the

Algorithm 6 Diploid Burrows-Wheeler Li and Stephens

Input: $x[0,n) \in \{0,1,2\}^n$, PBWT of $x_0, \ldots, x_{H-1}$, scores $\mu \geq 0$, $\rho \geq 0$.
Output: Minimum pair path score under diploid Li and Stephens model

1: Function *consider_recomb*(c, a_1, a_2, j):
2: If $c = 1$: Return $(a_1 + a_2 = 1)$
3: Else: Return $(a_j = c/2)$
4: $i \leftarrow n$; $st \leftarrow [(0, H, 0, H, 0)]$; $gm \leftarrow 0$; $lm[(0, H)] \leftarrow 0$;
5: *traceback* $\leftarrow [(n-1, -1, -1, -1, -1)]$
6: While $i > 0$: ▷ *st* repr. states of path pairs in full suffix set for $x[i, n)$
7: $i \leftarrow i - 1$; $st' \leftarrow []$; $gm' \leftarrow gm + 2\mu$; $lm' \leftarrow \{\}$; *extended* $\leftarrow \{\}$
8: *double_recomb* $\leftarrow$ *False*
9: For $(s_1, e_1, s_2, e_2, score) \times (a_1, a_2)$ in $st \times \{0,1\} \times \{0,1\}$:
10: $score' \leftarrow score + \mu|a_1 + a_2 - x[i]|$
11: $s'_j \leftarrow LF(s_j, a_j, i)$; $e'_j \leftarrow LF(e_j, a_j, i)$ $(j = 1, 2)$
12: If $s'_1 = e'_1$ or $s'_2 = e'_2$ or ($s_1 = s_2$ and $e_1 = e_2$ and $a_1 > a_2$) or $score \geq \min(lm[(s_1, e_1)] + \rho, lm[(s_2, e_2)] + \rho, gm + 2\rho)$ or $score' \geq \min(lm'[(s'_1, e'_1)] + \rho, lm'[(s'_2, e'_2)] + \rho, gm' + 2\rho)$:
13: continue
14: $st'.append((s'_1, e'_1, s'_2, e'_2, score'))$
15: $lm'[(s'_j, e'_j)] \leftarrow \min(score', lm'[(s'_j, e'_j)])$ $(j = 1, 2)$
16: $gm' \leftarrow \min(score', gm')$
17: If *consider_recomb*$(x[i], a_1, a_2, j)$ and $score = lm[(s_{3-j}, e_{3-j})]$:
18: $extended.insert((s_{3-j}, e_{3-j}, a_{3-j}))$ $(j = 1, 2)$
19: For $(s_1, e_1, s_2, e_2, score) \times (a_1, a_2, j)$ in $st \times \{0,1\} \times \{0,1\} \times \{1,2\}$:
20: $a_r \leftarrow a_j$; $a_x \leftarrow a_{3-j}$; $s_x \leftarrow s_{3-j}$; $e_x \leftarrow s_{3-j}$
21: $score' \leftarrow score + \rho + \mu|a_r + a_x - x[i]|$
22: $s'_r \leftarrow LF(0, a_r, i)$; $e'_r \leftarrow LF(H, a_r, i)$
23: $s'_x \leftarrow LF(s_x, a_x, i)$; $e'_x \leftarrow LF(e_x, a_x, i)$
24: If not *consider_recomb*$(x[i], a_1, a_2, j)$ or $s'_r = e'_r$ or $score > lm[(s_x, e_x)]$ or $(s_x, e_x, a_x) \in extended$ or $score' \geq \min(lm'[(s'_x, e'_x)] + \rho, lm'[(s'_r, e'_r)] + \rho, gm' + 2\rho)$:
25: continue
26: If $s'_x < e'_x$:
27: $st'.append((s'_x, e'_x, s'_r, e'_r, score'))$
28: $lm'[(s'_j, e'_j)] \leftarrow \min(score', lm'[(s'_j, e'_j)])$ $(j = x, r)$
29: $gm' \leftarrow \min(score', gm')$
30: $extended.insert((s_x, e_x, a_x))$
31: $traceback.append((i, s_x, e_x, s_r, score + \rho))$ ▷ Not *score*′!
32: If $x[i] \neq 1$ and $x[i] = a_r + a_x$ and not *double_recomb* and $score = gm$ and $(s_r, e_r, a_r) \notin extended$
33: If $score + 2\rho < lm'[(s'_r, e'_r)] + \rho$:
34: $st'.append((s'_r, e'_r, s'_r, e'_r, score + 2\rho))$
35: $traceback.append((i, s_x, -1, s_r, score + 2\rho))$
36: *double_recomb* $\leftarrow$ *True*
37: $gm \leftarrow gm'$; $lm \leftarrow lm'$; $st \leftarrow st'$
38: $gm\_idx1, gm\_idx2 \leftarrow$ any of $\{s_1, s_2 | (s_1, e_1, s_2, e_2, score) \in st$ and $score = gm\}$
39: Return *gm*, *gm_idx*1, *gm_idx*2, *traceback*

approach, the definitions of sequence group, state and full prefix set need to be modified.

The *sequence group* associated to an unordered pair of paths $\{\alpha,\beta\}$ is defined as $G(\alpha,\beta)=\{\{x,y\}|x\in G(\alpha),y\in G(\beta)\}$. Similarly, using the same justification as before, the *state* of an (unordered) path pair $\{\alpha,\beta\}$ is defined to be the pair $(G(\alpha,\beta),s(\alpha,\beta))$. A *full prefix set* S for $x[0,l)$ is defined as a set of (unordered) pairs of path prefixes such that for any sequence x' that extends $x[0,l)$, there exists a path pair $\{\alpha,\beta\}$ that achieves the minimum score $s_{x'}(\alpha,\beta)=\min_{\alpha',\beta'}s_{x'}(\alpha',\beta')$ and whose prefix pair $\{\alpha[0,l),\beta[0,l)\}$ is in S. Finally, to formulate the theorem it is handy to introduce the notation $\bar{S}$ to denote the set of 'haplotype' paths in S, or formally $\bar{S}=\{\alpha|\{\alpha,\beta\}\in S\}$.

THEOREM 2. *Suppose S is a full prefix set for $x[0,l)$ and S' is a set of prefixes of length $l+1$. Let $s_{\min}(\alpha)=\min_{\beta:\{\alpha,\beta\}\in S}s_x(\alpha,\beta)$, $s'_{\min}(\alpha)=\min_{\beta:\{\alpha,\beta\}\in S'}s_x(\alpha,\beta)$ and $s_{\min}=\min_\alpha s_{\min}(\alpha)$, $s'_{\min}=\min_\alpha s'_{\min}(\alpha)$. Then S' is a full prefix set for $x[0,l+1)$ if:*

a. *For all $\{\alpha,\beta\}\in S$ and $a,b\in\{0,1\}$, so that αa and βb are both extensions and $s_x(\alpha a,\beta b)<\min(s'_{\min}+2\rho,s'_{\min}(\alpha a)+\rho,s'_{\min}(\beta b)+\rho)$, we have $\{\alpha a,\beta b\}\in S'$; and*

b. *(If $x[l]=1$:) For all $\alpha\in\bar{S}$ and $a,b\in\{0,1\}$ with $a+b=1$, so that there is no β' satisfying $\{\alpha,\beta'\}\in S$ and $s(\alpha,\beta')=s_{\min}(\alpha)$ and both αa and $\beta' b$ are extensions, S' contains a path pair of the form $\{\alpha a,\beta Rb\}$ with $\{\alpha,\beta\}\in S$ and $s(\alpha,\beta)=s_{\min}(\alpha)$; and*

c. *(If $x[l]=2b$:) For all $\alpha\in\bar{S}$ and $a\in\{0,1\}$, so that there is no β' satisfying $\{\alpha,\beta'\}\in S$ and $s(\alpha,\beta')=s_{\min}(\alpha)$ and both αa and $\beta' b$ are extensions, S' contains a path pair of the form $\{\alpha a,\beta Rb\}$ with $\{\alpha,\beta\}\in S$ and $s(\alpha,\beta)=s_{\min}(\alpha)$; and*

d. *(If $x[l]=2b$:) If there is no pair $\{\alpha',\beta'\}$ for which $s(\alpha',\beta')=s_{\min}$ and either $\alpha' b$ or $\beta' b$ is an extension, then S' contains a path pair of the form $\{\alpha Rb,\beta Rb\}$ with $\{\alpha,\beta\}\in S$.*

Algorithm 6 implements these ideas. The core of the algorithm is formed by lines 11 and 14 that consider regular extensions with a pair of characters (a_1, a_2); lines 22–23 and 27 that consider single recombinations; and line 34 that considers simultaneous recombinations in both haplotypes. The remainder of the algorithm is concerned with implementing the conditions of Theorem 2 to ensure that redundant extensions are ignored. The variables *gm* and *gm'* keep track of the current and next global minimum score $s_{\min}$ and $s'_{\min}$, while the associative arrays *lm*[] and *lm'*[] keep track of $s_{\min}(\alpha)$ and $s'_{\min}(\alpha)$ respectively. The associative array *extended*[] keeps track which paths α have a partner β' that achieves the minimum score $s_{\min}(\alpha)$, and for which both α and β' have extensions required in conditions *b* and *c*; whether the extension is appropriate is computed by the function *consider_recomb*. Finally, the variable *double_recomb* is used to ensure that at most one double recombination is considered at every iteration.

The traceback algorithm for diploid Li and Stephens is similar to the haploid algorithm. Again, the *traceback* list contains records describing the recombinations that have been considered. These records now additionally contain a pair s_x, e_x that represent the range of PBWT indices corresponding to the sequence that does not undergo a recombination. As with the haploid algorithm, the traceback algorithm follows a recombination only if the path scores agree, but now also ensures that the index of the non-recombining path is contained in the range $[s_x,e_x)$. Double recombinations are encoded by setting $e_x=-1$, and for such recombinations only the scores need to agree. A pseudocode implementation is given as Algorithm 7.

Algorithm 7 Diploid traceback

Input: Sequence $x[0,n)$, PBWT of $x_0,\ldots,x_{H-1}$, scores $\mu\geq 0$, $\rho\geq 0$, minimum score *gm*, corresponding indices *gm_idx*1, *gm_idx*2, traceback list *traceback*.
Output: Representation of a minimum-scoring diploid path

1: Function $FL(k,i)$: ▷ "First-to-last" mapping
2: $lo\leftarrow 0$; $hi\leftarrow H$; $a\leftarrow 0$ if $k<f_i^0$ else 1
3: While $lo<hi$: ▷ $LF(j,a,i)\leq k\ \forall j<lo$ and $LF(j,a,i)>k\ \forall j\geq hi$
4: $mid\leftarrow\lfloor(lo+hi)/2\rfloor$
5: If $LF(mid,a,i)\leq k$: $lo\leftarrow mid+1$
6: Else $hi\leftarrow mid$
7: Return $a, lo-1$
8: $i\leftarrow 0$; $path1\leftarrow[(i,a^{i-1}_{gm\_idx1})]$; $path2\leftarrow[(i,a^{i-1}_{gm\_idx2})]$
9: For $(t\_locus,t\_start,t\_end,t\_idx,t\_score)$ in $reverse(tr'back)$:
10: While $i\leq t\_locus$:
11: $a1,gm\_idx1\leftarrow FL(gm\_idx1,i)$
12: $a2,gm\_idx2\leftarrow FL(gm\_idx2,i)$
13: $gm\leftarrow gm-\mu|a1+a2-x[i]|$; $i\leftarrow i+1$
14: If $gm=t\_score$:
15: If $t\_end=-1$: ▷ Double recombination
16: $gm\_idx1\leftarrow t\_start$; $gm\_idx2\leftarrow t\_idx$
17: $path1.append((i,a^{i-1}_{gm\_idx1}))$; $path2.append((i,a^{i-1}_{gm\_idx2}))$
18: $gm\leftarrow gm-2\rho$
19: Else If $t\_start\leq gm\_idx1<t\_end$:▷ Single rec. in path 2
20: $gm\_idx2\leftarrow t\_idx$; $path2.append((i,a^{i-1}_{gm\_idx2}))$
21: $gm\leftarrow gm-\rho$
22: Else if $t\_start\leq gm\_idx2<t\_end$: ▷ Single rec. in path 1
23: $gm\_idx1\leftarrow t\_idx$; $path1.append((i,a^{i-1}_{gm\_idx1}))$
24: $gm\leftarrow gm-\rho$
25: Return $path1, path2$

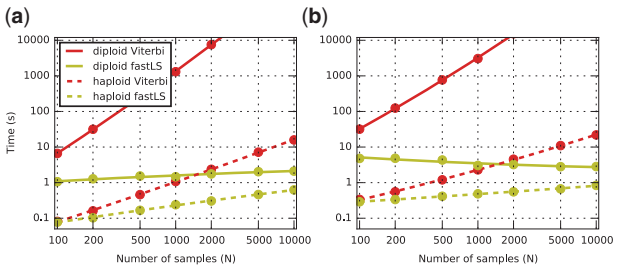

Fig. 2. Running time for inferring inheritance patterns under the haploid (dashes) and diploid Li and Stephens model over a simulated reference set of *n* (horizontal axis) haploid sequences, using the Viterbi (red) and fastLS (green) algorithms, using $\rho/\mu=2$. Dots represent measurements, curves show quadratic fits. (**a**) Results for a simulated reference population of *n* samples. (**b**) Results for a fixed simulated reference population of 100 000, subsampled to *n* samples

4 Performance

For testing the fastLS algorithms were implemented in C++, with all tables stored in uncompressed form in memory. To validate the implementations and to compare runtimes, standard Viterbi

algorithms for the haploid and diploid LS model were also implemented. Traceback was included in the fastLS algorithms, but was excluded from the Viterbi implementations because of memory constraints. Two sets of simulations were performed. For the first, 30 Mb of sequence in populations of size 100 to 10 000 were simulated by scrm (Staab *et al.*, 2015) using the 'standard simulation' model of Li and Durbin (2011) which roughly resembles the demography of the European population. For each population I simulated an additional 50 samples to serve as input sequences. This resulted in a number of segregating sites ranging from 129 945 for the 150-sample case, to 436 361 for 10 050 samples. For the second set, I simulated a single population of 100 000 samples under the same model (resulting in 621 156 segregating sites) and sub-sampled reference populations of 100 to 10 000 samples from these (Fig. 2).

The run-times of the Viterbi algorithms show the expected linear and quadratic dependence on H. The fastLS algorithms show a weak dependence on H. In the case of the sub-sampled population, which have a fixed number of loci (not all of which segregate in the sample), the dependence on H is weakest, and in fact the diploid algorithm becomes faster for larger populations, probably because longer haplotype matches can be found in larger populations, resulting in more efficient pruning of the prefix sets.

A1. Appendix

A1.1 Derivation of Algorithm 3

To derive the PBWT algorithm for sequence matching we first need to describe the structure of M. From Figure 1 we see that

$$C(\mathbf{p}_i) = Hi; \quad C(0) = Hn; \quad C(1) = Hn + F^0$$

where $F^a = \sum_{j=0}^{n-1} f_j^a$ is the number of symbols $a \in \{0,1\}$ in X, and f_i^a is the (haplotype) frequency of a at position i in $x_0, \ldots, x_{H-1}$. Let $F_i^a := \sum_{j=0}^{i-1} f_j^a$ be the cumulative haplotype frequency across positions up to $i-1$, and set $F^a = F_n^a$. Then $R^a(i)$ satisfies

$$R^0(Hi) = F_i^0, i \le n \tag{4}$$

$$R^{\mathbf{p}_i}(r) = r - C(0) - F_{i+1}^0, r \in [C(0) + F_{i+1}^0, C(0) + F_{i+2}^0] \tag{5}$$

$$R^{\mathbf{p}_i}(r) = f_{i+1}^0 + r - C(1) - F_{i+1}^1, r \in [C(1) + F_{i+1}^1, C(1) + F_{i+2}^1] \tag{6}$$

I define $r_i^a(k)$ so that

$$R^a(Hi + k) = F_i^a + r_i^a(k) \quad \text{for } k \in [0, H] \text{ and } a \in \{0,1\}, \tag{7}$$

or equivalently, $r_i^a(k)$ counts how often a appears in $BWT(X)[Hi, Hi + k)$. To derive the PBWT sequence matching algorithm, it suffices to track one of the bounding variables, say s, for two steps through Algorithm 2. Assume that the subsequence matched so far starts at position i, so that $s = Hi + k$, $k \in [0, H]$, and that the next character to be matched is $a \in \{0,1\}$. The first step replaces s with

$$s' = C(a) + R^a(s) = C(a) + F_i^a + r_i^a(k)$$

where the second equality follows from (7). The function $r_i^a(k)$ returns the number of occurrences of a before the kth row within the block starting at row iH in M. This block includes all sequences that start with $\mathbf{p}_i$, so that $0 \le r_i^a(k) \le f_i^a$ for $k \in [0, H]$, and the conditions for (5) and (6) apply, allowing the result of the second step to be computed. The sequence now ends with the symbol $\mathbf{p}_{i-1}$, so that if $a = 0$, s' is replaced by

$$\begin{aligned} s'' &= C(\mathbf{p}_{i-1}) + R^{\mathbf{p}_{i-1}}(s') \\ &= H(i-1) + [C(0) + F_i^0 + r_i^0(k)] - C(0) - F_i^0 \\ &= H(i-1) + r_i^0(k) \end{aligned}$$

whereas if $a = 1$,

$$\begin{aligned} s'' &= C(\mathbf{p}_{i-1}) + R^{\mathbf{p}_{i-1}}(s') \\ &= H(i-1) + [C(1) + F_i^1 + r_i^1(k)] + f_i^0 - C(1) - F_i^1 \\ &= H(i-1) + f_i^0 + r_i^1(k) \end{aligned}$$

Since $R^0(r) + R^1(r) = r$ for $r \le Hn$, it follows that $r_i^0(k) + r_i^1(k) = k$ for $0 \le k \le n$, so that the last-to-first function mapping k to the new value k' satisfying $s'' = H(i-1) + k'$ is $LF(k, a, i)$ as defined in (3).

A1.2 Proof of Theorem 1

The key observation is that if S' contains a path π' with state (G', s'), then S' does not need to contain any path π (of the same length) with state (G, s) if $G' \supseteq G$ and $s' \le s$. In this case I say that π' *undercuts* π, or symbolically $\pi' \le \pi$. In addition, if $\pi' R \le \pi$ I also say that $\pi' \le \pi$, again because all scores that are achievable with π as prefix are also achievable with prefix π'.

Since S is a full prefix set for $x[0, l)$, a trivial full prefix set for $x[0, l+1)$ is formed by the union of simple extensions $S'_x = \{\pi a | \pi \in S, a \in \{0,1\}\}$, and recombination extensions $S'_r = \{\pi R a | \pi \in S, a \in \{0,1\}\}$. To prove that $S' \subseteq S'_r \cup S'_x$ is also a full prefix set, we need to show that any path $\pi \in S'_x \cup S'_r \backslash S'$ is undercut by some path $\pi' \in S'$. In the proof below I will identify for any such π a π' that *strictly* undercuts π (written as $\pi' < \pi$)—that is, either the score is strictly lower or the group is strictly larger—but which is not necessarily an element of S'. If an element is found that is not in S', the process can be repeated, finding a $\pi'' < \pi' < \pi$, and so forth. This process has to stop eventually, with an element in S', because s cannot decrease indefinitely and G cannot increase indefinitely.

Proof: First consider an arbitrary element $\pi \in S'_x \backslash S'$. Because $\pi \notin S'$ we have $s(\pi) \ge s'_{\min} + \rho$. Consider $\pi' R$ with $\pi' \in S'$ such that $s(\pi') = s'_{\min}$, then $s(\pi' R) = s'_{\min} + \rho$ and $G(\pi' R) = \Omega \supset G(\pi)$, so that $\pi' R < \pi$, and therefore $\pi' < \pi$.

Next, consider an arbitrary element of S'_r, say $\pi R a$. We may assume that $s(\pi) = s_{\min}$, as otherwise $\pi' R a$ with $s(\pi') = s_{\min}$ strictly undercuts it. We may also assume that $a = x[l]$, since otherwise let πc be some extension of π (which must exist), then $s(\pi c R) \le s(\pi) + \mu + \rho = s(\pi R a)$ and $G(\pi c R) = \Omega \supset G(\pi R a)$ so that $\pi c R < \pi R a$ and therefore $\pi c < \pi R a$. Finally, if πa exists, then $s(\pi a R) = s(\pi R a)$ and $G(\pi a R) \supset G(\pi R a)$ so that $\pi a < \pi R a$. This completes the proof.

A1.3 Proof of Theorem 2

The structure of this proof is identical to the previous one. The equivalent observation is that a full prefix set S' does not need to contain a path pair $\{\alpha, \beta\}$ if S' already contains a path pair $\{\alpha', \beta'\}$ with $s(\alpha', \beta') \le s(\alpha, \beta)$ and $G(\alpha', \beta') \supseteq G(\alpha, \beta)$; in this case I say that the path pair $\{\alpha', \beta'\}$ undercuts $\{\alpha, \beta\}$, or symbolically $\{\alpha', \beta'\} \le \{\alpha, \beta\}$. I also write $\{\alpha', \beta'\} \le \{\alpha, \beta\}$ if any one of $\{\alpha' R, \beta'\} \le \{\alpha, \beta\}$, $\{\alpha', \beta' R\} \le \{\alpha, \beta\}$ or $\{\alpha' R, \beta' R\} \le \{\alpha, \beta\}$ is true.

A trivial full prefix set for $x[0, l+1)$ is formed by the union $S'_x \cup S'_r \cup S'_{rr}$, where $S_x = \{\{\alpha a, \beta b\} | \{\alpha, \beta\} \in S; a, b \in \{0,1\}\}$, $S_r =$

$\{\{\alpha a, \beta Rb\}|\{\alpha,\beta\} \in S; a,b \in \{0,1\}\}$ and $S_{rr} = \{\{\alpha Ra, \beta Rb\}|\{\alpha,\beta\} \in S; a,b \in \{0,1\}\}$. The task is to prove that any path pair in S'_x, S' r or S'_{rr} but not in S' is undercut by some element of S', and again I do this by identifying for any $\pi \in S'_x \cup S'_r \cup S'_{rr} \backslash S'$ a π' that strictly undercuts π.

Proof: Consider an arbitrary $\{\alpha a, \beta b\} \in S'_x$ not in S', so that $s(\alpha a, \beta b) \geq \min(s'_{\min} + 2\rho, s'_{\min}(\alpha a) + \rho, s'_{\min}(\beta b) + \rho)$. Suppose first that $s(\alpha a, \beta b) \geq s'_{\min} + 2\rho$, and let $\alpha' a'$ and $\beta' b'$ be such that $s(\alpha' a', \beta' b') = s'_{\min}$, then $G(\alpha' a' R, \beta' b' R) \supset G(\alpha a, \beta b)$ and $s(\alpha' a' R, \beta' b' R) = s'_{\min} + 2\rho \leq s(\alpha a, \beta b)$, so $\{\alpha' a' R, \beta' b' R\} < \{\alpha a, \beta b\}$, and so $\{\alpha' a', \beta' b'\} < \{\alpha a, \beta b\}$. Alternatively, suppose that $s(\alpha a, \beta b) \geq s'_{\min}(\alpha a) + \rho$, and let $\beta' b'$ be a path so that $\{\alpha, \beta'\} \in S$ and $s(\alpha a, \beta' b') = s'_{\min}(\alpha a)$, then $G(\alpha a, \beta' b' R) \supset G(\alpha a, \beta b)$ and $s(\alpha a, \beta' b' R) = s'_{\min}(\alpha a) + \rho \leq s(\alpha a, \beta b)$, so that $\{\alpha a, \beta' b' R\} < \{\alpha a, \beta b\}$, and so $\{\alpha a, \beta' b'\} < \{\alpha a, \beta b\}$. The case $s(\alpha a, \beta b) \geq s'_{\min}(\beta b) + \rho$ is similar.

Next, consider an arbitrary element $\{\alpha a, \beta Rb\} \in S'_r$. We may assume that $s(\alpha,\beta) = s_{\min}(\alpha)$ as otherwise it is possible to undercut this pair by choosing β appropriately. We may also assume that no $\{\alpha, \beta'\}$ exists in S so that $s(\alpha,\beta') = s_{\min}(\alpha)$ and αa and $\beta' b$ are extensions, for if such a pair exists, the pair $\{\alpha a, \beta' bR\}$ undercuts $\{\alpha a, \beta Rb\}$ as it achieves the same score and has a strictly larger group. Now suppose $x[l] = 1$. If $a + b \neq 1$, for any extension $\beta b'$ of β we have $s(\alpha a, \beta b' R) \leq s(\alpha,\beta) + \mu + \rho = s(\alpha a, \beta Rb)$ and $G(\alpha a, \beta b' R) \supset G(\alpha a, \beta Rb)$ so that $\{\alpha a, \beta b' R\} < \{\alpha a, \beta Rb\}$, as required. To deal with the case $x[l] \neq 1$, say $x[l] = 0$, suppose $b = 1$ and let $\beta b'$ be any extension, then $s(\alpha a, \beta b' R) \leq s(\alpha,\beta) + (a + 1)\mu + \rho = s(\alpha a, \beta R1)$ so that $\{\alpha a, \beta b' R\} < \{\alpha a, \beta R1\}$, as required. The case $x[l] = 2$ is similar.

Finally, consider an arbitrary element $\{\alpha Ra, \beta Rb\} \in S'_{rr}$. As before we may assume that $s(\alpha,\beta) = s_{\min}$. Let's first deal with the case $x[l] = 1$. If $a = b$ then let $\alpha a'$ be an arbitrary extension, then $s(\alpha a' R, \beta Rb) \leq s(\alpha,\beta) + \mu + 2\rho = s(\alpha Ra, \beta Rb)$ and $G(\alpha a' R, \beta Rb) \supset G(\alpha Ra, \beta Rb)$ so $\{\alpha a' R, \beta Rb\} < \{\alpha Ra, \beta Rb\}$. If instead $a \neq b$, then let $\alpha a'$ and $\beta b'$ be arbitrary extensions. If $a' = a$ then $\{\alpha a' R, \beta Rb\} < \{\alpha Ra, \beta Rb\}$ by a now familiar argument. If $b' = b$ then $\{\alpha Ra, \beta b' R\}$ is the required strictly undercutting path pair. If both $a' \neq a$ and $b' \neq b$ then $a' \neq b'$ and $\{\alpha a' R, \beta b' R\}$ achieves the same score and a larger group, and therefore strictly undercuts $\{\alpha Ra, \beta Rb\}$. It remains to deal with the case $x[l] \neq 1$, say $x[l] = 0$. If either $a = 1$ or $b = 1$ (or both), say $b = 1$, then let $\beta b'$ be an arbitrary extension, then $s(\alpha Ra, \beta b' R) \leq (a + 1)\mu + 2\rho = s(\alpha Ra, \beta Rb)$ so that $\{\alpha Ra, \beta b' R\} < \{\alpha Ra, \beta Rb\}$. So we can assume that $a = b = 0$. The argument in the case $x[l] = 2$ is similar. Finally, suppose there is a pair $\{\alpha', \beta'\}$ with $s(\alpha', \beta') = s_{\min}$ and either $\alpha' b$ or $\beta' b$ is an extension, say $\beta' b$ is, then $\{\alpha' Rb, \beta' bR\} \leq \{\alpha Rb, \beta Rb\}$ as required. This completes the proof.

Acknowledgements

Thanks to Sorina Maciuca and Zam Iqbal who introduced me to the idea of position symbols which led directly to this work; and to Gil McVean and Jerome Kelleher for helpful comments on the manuscript.

Funding

This work was supported by Wellcome Trust grant 090532/Z/09/Z.

Conflict of Interest: none declared.

References

Beaumont,M.A. (2010) Approximate Bayesian Computation in evolution and ecology. *Ann. Rev. Ecol. Evol. Syst.*, **41**, 379–406.

Burrows,M. and Wheeler,D.J. (1994) *A Block Sorting Lossless Data Compression Algorithm. Technical Report 12*, Digital Equipment Corporation.

Durbin,R. (2014) Efficient haplotype matching and storage using the positional Burrows-Wheeler transform (PBWT). *Bioinformatics*, **30**, 1266–1272.

Griffiths,R. and Marjoram,P. (1997) An ancestral recombination graph. In: Donnelly,P. and Tavaré,S. (eds) *Progress in Population Genetics and Human Evolution, Volume 87 of IMA Volumes in Mathematics and Its Applications*. Springer Verlag, Berlin.

Howie,B.N. *et al.* (2009) A flexible and accurate genotype imputation method for the next generation of genome-wide association studies. *PLoS Genet.*, **5**, e1000529.

Hudson,R.R. (1983) Properties of a neutral allele model with intragenic recombination. *Theor. Pop. Biol.*, **23**, 183–201.

Kingman,J.F.C. (1982) On the genealogy of large populations. *J. Appl. Probab.*, **19**, 27–43.

Langmead,B. *et al.* (2009) Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biol.*, **10**, R25.

Li,H. and Durbin,R. (2009) Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, **25**, 1754–1760.

Li,H. and Durbin,R. (2011) Inference of human population history from individual whole-genome sequences. *Nature*, **475**, 493–496.

Li,N. and Stephens,M. (2003) Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data. *Genetics*, **165**, 2213–2233.

Staab,P.R. *et al.* (2015) scrm: efficiently simulating long sequences using the approximated coalescent with recombination. *Bioinformatics*, **31**, 1680–1682.

The International HapMap Consortium (2005) A haplotype map of the human genome. *Nature*, **437**, 1299–1320.

The Wellcome Trust Case Control Consortium (2007) Genome-wide association study of 14 000 cases of seven common diseases and 3000 shared controls. *Nature*, **447**, 661–678.