Abstract

Hidden Markov models of haplotype inheritance such as the Li and Stephens model allow for computationally tractable probability calculations using the forward algorithms as long as the representative reference panel used in the model is sufficiently small. Specifically, the monoploid Li and Stephens model and its variants are linear in reference panel size unless heuristic approximations are used. However, sequencing projects numbering in the thousands to hundreds of thousands of individuals are underway, and others numbering in the millions are anticipated.

To make the Li and Stephens forward algorithm for these datasets computationally tractable, we have created a numerically exact version of the algorithm with observed average case $\mathcal{O}(nk^{0.35})$ runtime, avoiding any tradeoff between runtime and model complexity. We demonstrate that our approach also provides a succinct data structure for general purpose haplotype data storage. We discuss generalizations of our algorithmic techniques to other hidden Markov models.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Applied computing → Bioinformatics

Keywords and phrases Haplotype, Hidden Markov Model, Forward Algorithm, Lazy Evaluation

Supplement Material https://github.com/yoheirosen/sublinear-Li-Stephens/

Funding This work was supported by the National Human Genome Research Institute of the National Institutes of Health under Award Number 5U54HG007990, the National Heart, Lung, and Blood Institute of the National Institutes of Health under Award Number 1U01HL137183-01, and grants from the W.M. Keck foundation and the Simons Foundation.

Acknowledgements We would like to thank Jordan Eizenga for his helpful discussions throughout the development of this work.

An average-case sublinear exact Li and Stephens forward algorithm

Yohei M. Rosen[1]
University of California, Santa Cruz, California
New York University School of Medicine, New York, New York
yohei@ucsc.edu
https://orcid.org/0000-0002-3870-2169

Benedict J. Paten
University of California, Santa Cruz, California
bpaten@ucsc.edu

1 Introduction

Probabilistic models of haplotypes describe how variation is shared in a population. One application of these models is to calculate the probability $P(o|H)$ of a haplotype o given the assumption that it is a member of a population represented by a *reference panel* of haplotypes H. This computation has been used in estimating recombination rates [8], a problem of interest in genetics and in medicine. It may also be used to detect errors in genotype calls.

Early approaches to haplotype modeling used coalescent [7] models which were accurate but computationally complex, especially when including recombination. Li and Stephens wrote the foundational computationally tractable haplotype model [8] with recombination. Under their model, the probability $P(o|H)$ can be calculated using the forward algorithm for hidden Markov models. Generalizations of their model have been used for haplotype phasing and genotype imputation. Most of these algorithms [10, 1, 14, 3, 12] use the *forward probabilities* calculated as intermediate values in the forward algorithm.

1.1 The Li and Stephens model

Consider a *reference panel* H of k haplotypes sampled from some population. Each haplotype $h_j \in H$ is a sequence $(h_{j,1}, \ldots, h_{j,n})$ of alleles at a contiguous sequence $1, \ldots, n$ of genetic sites. Classically [8], the sites are biallelic, but the model extends to multiallelic sites. [11]

Consider an observed sequence of alleles $o = (o_1, \ldots, o_n)$ representing another haplotype. The monoploid Li and Stephens model (LS) [8] specifies a probability that o is descended from the population represented by H. LS can be written as a hidden Markov model wherein the haplotype o is assembled by copying (with possible error) consecutive contiguous subsequences of haplotypes $h_j \in H$.

▶ **Definition 1** (Li and Stephens HMM). Define $x_{j,i}$ as the event that the allele o_i at site i of the haplotype o was copied from the allele $h_{j,i}$ of haplotype $h_j \in H$. Take parameters

$$\rho^*_{i-1\to i} \quad \text{the probability of recombination between sites } i-1 \text{ and } i \tag{1}$$

$$\mu_i \quad \text{the probability of a mutation from one allele to another at site } i \tag{2}$$

[1] Yohei Rosen was supported in part by a Howard Hughes Medical Institute Medical Research Fellowship.

and from them define the transition and recombination probabilities

$$p(x_{j,i}|x_{j',i-1}) = \begin{cases} 1 - k\rho_i & \text{if } j = j' \\ \rho_i & \text{if } j \neq j' \end{cases} \qquad \text{where } \rho(i) = \frac{\rho^*_{i-1\to i}}{k} \tag{3}$$

$$p(o_i|x_{j,i}) = \begin{cases} 1 - (A-1)\mu_i & \text{if } o_i = h_{j,i} \\ \mu_i & \text{if } o_i \neq h_{j,i} \end{cases} \qquad \text{where } A = \text{number of alleles} \tag{4}$$

The forward algorithm for hidden Markov models allows calculation of $P(o|H)$ in $\mathcal{O}(nk^2)$ time using an $n \times k$ dynamic programming matrix of *forward states*

$$p_i[j] = P(x_{j,i}|o_1, \ldots, o_i, H) \tag{5}$$

In practice, the Li and Stephens forward algorithm is $\mathcal{O}(nk)$. (See §3)

1.1.1 Li and Stephens like algorithms for large populations

The $\mathcal{O}(nk)$ time complexity of the forward algorithm is intractable for reference panels with large size k. The UK Biobank has amassed $k = 500,000$ array samples. Whole genome sequencing projects, with a denser distribution of sites, are catching up. Major sequencing projects with $k = 100,000$ or more samples are nearing completion. Others numbering k in the millions have been announced. These large population datasets have significant potential benefits: They are statistically likely to more accurately represent population frequencies and those employing genome sequencing can provide phasing information for rare variants.

In order to handle datasets with size k even fractions of these sizes, modern haplotype inference algorithms depend on models which are simpler than the Li and Stephens model or which sample subsets of the data. For example, the common tools Eagle-2, Beagle, HAPI-UR and Shapeit-2 and -3 [10, 1, 14, 3, 12] either restrict where recombination can occur, fail to model mutation, model long-range phasing approximately or sample subsets of the reference panel.

Lunter's "fastLS" algorithm [11] demonstrated that haplotypes models which include all k reference panel haplotype could find the Viterbi maximum likelihood path in time sublinear in k, using preprocessing to reduce redundant information in the algorithm's input. However, his techniques do not extend to the forward and forward-backward algorithms.

1.2 Our contributions

We have developed an arithmetically exact forward algorithm whose expected time complexity is a function of the expected allele distribution of the reference panel. This expected time complexity proves to be $\mathcal{O}(k^{0.35})$ in reference panel size. We have also developed a technique for succinctly representing large panels of haplotypes whose size also scales as a sublinear function of the expected allele distribution.

Our forward algorithm contains three optimizations, all of which might be generalized to other bioinformatics algorithms. In (§2), we rewrite the reference panel as a sparse matrix containing the minimum information necessary to directly infer all allele values. In (§3), we define recurrence relations which are numerically equivalent to the forward algorithm but use minimal arithmetic operations. In (§4), we delay computation of forward states using a lazy evaluation algorithm which benefits from blocks of common sequence. Our methods apply to other models which share certain properties with the monoploid Li and Stephens model.

2 Sparse representation of haplotypes

The forward algorithm to calculate the probability $P(o|H)$ takes as input a length n vector o and a $k \times n$ matrix of haplotypes H. Therefore time complexity better than $\mathcal{O}(nk)$ is impossible unless there is preprocessing of its input. However, such preprocessing can be amortized over many queries o.

2.1 Information content of a reference panel

Recall that $(o_i)_{i=1}^n$ is the allele sequence of the emitted haplotype o. (§3) will show that $\phi_i(o_i, H), 1 \leq i \leq n$ defined below are sufficient data to calculate $P(o|H)$.

▶ **Definition 2.** The **information content** ϕ of H for allele a at site i is defined as

$$\phi_i(a) = \begin{cases} Match_i(a) & := \{h_{j,i} \mid h_{j,i} = a\} \quad \text{if } |Match_i(a)| \leq |NonMatch_i(a)| \\ NonMatch_i(a) & := \{h_{j,i} \mid h_{j,i} \neq a\} \quad \text{if } |NonMatch_i(a)| < |Match_i(a)| \end{cases} \tag{6}$$

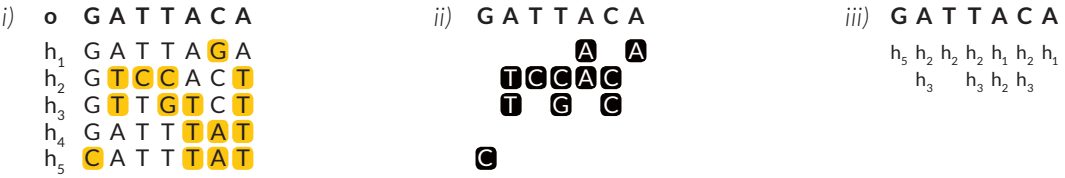

Figure 1 **i)** Reference panel $\{h_1, \ldots, h_5\}$ with mismatches to haplotype o shown in yellow. **ii)** Elements of $\phi_i(o_i)$ in black. **iii)** Vectors to encode $\phi_i(o_i)$ at each site.

2.2 Relation of information content to allele frequency spectrum

Our sparse representation of the haplotype reference panel benefits from the recent finding [6] that the distribution over sites of minor allele frequencies is biased towards low frequencies[2].

We will compute the expected time sum of the information content over all sites assuming first that all sites are biallelic[3]. In the biallelic case $\phi_i(\cdot)$ is always the set of haplotypes displaying the minor allele at site i and the distribution of $\phi_i(a)$ is the allele frequency spectrum.

▶ **Lemma 3.** *Let $\mathbb{E}[\overline{f}](k)$ be the expected mean minor allele frequency for k genotypes. Then*

$$\mathbb{E}\left[\sum_i \left|\phi_i(a)\right|\right] = \mathbb{E}[\overline{f}](k) \tag{7}$$

▶ **Corollary 4.** *If $\mathcal{O}(\mathbb{E}[\overline{f}]) < \mathcal{O}(k)$, then $\mathcal{O}(\sum_i \left|\phi_i(a)\right|) < \mathcal{O}(k)$ in expected value.*

[2] We confirm these results in section 5.2
[3] The generalization is trivial

2.3 Implementation

For biallelic sites, we store our ϕ_i's using a length-n vector of length $|\phi_i|$ vectors containing the indices j of the haplotypes $h_j \in \phi_i$ and a length-n vector listing the major allele at each site. (See Figure 1 panel iii) Random access by key i to iterators to the first elements of sets $\phi_i(a)$ is $\mathcal{O}(1)$ and iteration across these $\phi_i(a)$ is linear in the size of $\phi_i(a)$. For multiallelic sites, the data structure uses slightly more space but has the same speed guarantees.

Generating these data structures takes $\mathcal{O}(nk)$ time but is embarrassingly parallel in n. Our "*.slls" data structure doubles as a succinct haplotype index which could be distributed instead of a large vcf record. A vcf $\rightarrow$ slls conversion tool is found in our github repository.

Adding or rewriting a haplotype is average case constant time per site per haplotype.

3 Efficient dynamic programming

We begin with the recurrence relation of the $\mathcal{O}(nk)$ Li and Stephens forward algorithm [8]:

$$p_i[j] = \mu_i(j)\big((1-k\rho_i)p_{i-1}[j] + \rho_i S_{i-1}\big) \tag{8}$$

$$S_i = \sum_{j=1}^{k} p_i[j] \tag{9}$$

We will reduce the number of summands in (9) and reduce the number indices j for which (8) is evaluated, using the **information content** defined in (§2.1).

▶ **Lemma 5.** *The summation (9) is calculable using strictly fewer than k summands.*

Proof. Suppose first that $\mu_i(j) = \mu$ for all $j \leq k$. Then

$$S_i = \sum_{j=1}^{k} p_i[j] = \mu \sum_{j=1}^{k} \big((1-k\rho_i)p_{i-1}[j] + \rho_i S_{i-1}\big) \tag{10}$$

$$= \mu\big((1-k\rho_i)S_{i-1} + k\rho_i S_{i-1}\big) = \mu S_{i-1} \tag{11}$$

and therefore, relaxing the requirement on $\mu_i(j)$,

$$S_i = \alpha S_{i-1} + \beta \sum_{j \in \phi_i(o_i)} p_i[j] \tag{12}$$

where

$$\alpha = \mu_i \qquad \beta = \frac{1-2\mu_i}{1-\mu_i} \qquad \text{if } \phi_i(a) = Match_i(a) \tag{13}$$

$$\alpha = 1-\mu_i \qquad \beta = \frac{2\mu_i - 1}{\mu_i} \qquad \text{if } \phi_i(a) = NonMatch_i(a) \tag{14}$$

◀

▶ **Lemma 6.** *If $j \notin \phi_i(o_i)$ and $j \notin \phi_{i-1}(o_{i-1})$, then S_i can be calculated without knowing $p_{i-1}[j]$ and $p_i[j]$, as can $p_i[j']$ for $j' \neq j$.*

Proof. By inspection of equation (12). ◀

▶ **Corollary 7.** *The recurrences (9) and the minimum set of recurrences (8) needed to compute (9) can be evaluated in $\mathcal{O}(|\phi|)$ time, assuming that $p_{i-1}[j]$ have been computed $\forall j \in \phi_i(o_i)$.*

We address the assumption on prior calculation of the necessary $p_{i-1}[j]$'s in section 4.

Figure 2 Illustration, with one line per operation, of arithmetic for **i)** conventional $\mathcal{O}(nk)$ Li and Stephens recurrence relations **ii)** Our procedure specified in equation (12). Sums in yellow.

3.1 Time complexity

Recall that we defined $\mathbb{E}[\overline{f}](k)$ as the expected mean minor allele frequency in a sample of size k. By Corollary 7 the procedure in eq. (12) has expected time complexity $\mathcal{O}\left(n\mathbb{E}[\overline{f}](k)\right)$.

4 Lazy evaluation of dynamic programming rows

Corollary 7 was conditioned on the assumption that specific forward probabilities had already been evaluated. We will describe a second algorithm which performs this task efficiently by avoiding arithmetic which will prove unnecessary at future steps.[4]

4.1 Eliminating redundant recurrence evaluations

The recurrence relations (8) are linear maps $r_i[j] : \mathbb{R} \to \mathbb{R}$ of the form

$$r_i[j] : x_j \longmapsto \alpha_i x_j + \beta_i \tag{15}$$

From these, define the linear maps

▶ **Definition 8.** For any $i_1 < i_2$, define the *update map* $r_{i_1 \to i_2}[j] = r_{i_2}[j] \circ r_{i_2-1}[j] \circ \cdots \circ r_{i_1+1}[j]$

This update map is defined such that $r_{i_1 \to i_2}[j](p_{i_1}[j]) = p_{i_2}[j]$.

▶ **Lemma 9.** *At each i there exist only two unique maps among the $r_i[j]$.*

Proof. Define $\mu_i^\circ = \mu_i$ and $\mu_i^\bullet = 1 - \mu_i$ if $\phi_i(a) = Match_i(a)$ and vice versa otherwise. Then

$$r_i^\circ(x) = \mu_i^\circ\big((1 - k\rho)x + \rho S_i\big) \tag{16}$$

$$r_i^\bullet(x) = \mu_i^\bullet\big((1 - k\rho)x + \rho S_i\big) = \frac{\mu_i^\bullet}{\mu_i^\circ} r_i^\circ \tag{17}$$

◀

This lemma allows us to rewrite each $r_{i_1 \to i_2}[j]$ as a binary vector of the form $(\circ, \circ, \bullet, \ldots, \circ, \bullet, \circ)$.

Our basic delayed evaluation rule is:

When Algorithm 1 is applied independently to all h_j, the aggregate algorithm has $\mathcal{O}(nk)$ time complexity, so we will share work between haplotypes j using equivalence classes segregated by runs of homology.

[4] This approach is known as *lazy evaluation*.

Algorithm 1: General approach to evaluating $p_{(\cdot)}[j]$ for a given haplotype j

Calculate $p_1[j]$ **and** set $\ell \leftarrow 1$
for *all sites* $i > 1$ **do**
  if $r_{\ell\to i} = (\circ, \circ, \ldots, \circ)$ **then** Do nothing
  if $r_{\ell\to i} = (\circ, \circ, \ldots, \bullet)$ **then** Evaluate $p_i[j] = r_{\ell\to i}(p_\ell[j])$ **and** set $\ell \leftarrow i$

4.2 Equivalence classes of update map prefixes

Under the conditions of Algorithm 1, if, at step i, $p_{(\cdot)}[j_1]$ and $p_{(\cdot)}[j_1]$ were both last calculated at the same index $\ell < i$, then the sequences of h_{j_1} and h_{j_2} are identical between ℓ and i, so

$$r_{\ell\to i}[j_1] = r_{\ell\to i}[j_2] = \underbrace{(\circ, \circ, \ldots, \circ)}_{i-\ell} \tag{18}$$

We put j_1 and j_2 into an equivalence class $J[\ell]$ to avoid recalcuating $r_{\ell\to i} := \underbrace{(\circ, \circ, \ldots, \circ)}_{i-\ell}$.

It is also inefficient to construct $r_{\ell\to i}$ independently for each $1 < \ell < i$. Observe:

▶ Remark. If $J[\ell]$ is empty we need not calculate $r_{\ell\to i}$.

▶ **Lemma 10.** *If $i_1 < i_2 < i$, then $r_{i_1\to i} = r_{i_2\to i} \circ r_{i_1\to i_2}$*

Lemma 10 allows us to calculate intermediate prefixes of the maps $r_{\ell\to i}$ and extend them at a later time. To make this concrete, suppose that we have an index $\blacktriangleleft_\ell$ where $\ell < \blacktriangleleft_\ell < i$. Then we can evaluate the prefix $r_\ell^{\blacktriangleleft} = r_{i\to\blacktriangleleft_\ell}$ of $r_{\ell\to i}$ knowing that $r_{\ell\to i} = r_{\blacktriangleleft_\ell\to i} \circ r_\ell^{\blacktriangleleft}$ can be evaluated at a later time.

4.3 The lazy evaluation algorithm

The data below specifies the state at each step i of our lazy evaluation algorithm. The algorithm initialization is described in Algorithm 2 and the recurrence in Algorithm 3.

- The maps $j \mapsto J[\ell]$; ℓ defined as the index at which $p_{(\cdot)[j]}$ was most recently calculated
- The maps $J[\ell] \mapsto r_\ell^{\blacktriangleleft}$; $r_\ell^{\blacktriangleleft}$ defined as the prefix of $r_{\ell\to i}$ which was most recently calculated
- The maps $J[\ell] \mapsto \blacktriangleleft_\ell$; $\blacktriangleleft_\ell$ defined as the index up to which a prefix of $r_{\ell\to i}$ which was most recently calculated
- The maps $r_1^\circ, r_2^\circ, \ldots, r_i^\circ$

Algorithm 2: Lazy evaluation initialization

Input: reference panel size k **and** active rows $\phi_1(o_1)$
All $j \notin \phi_1$ are assigned to equivalence class $J[0]$
$r_0^{\blacktriangleleft} \leftarrow r_1^\circ$ **and** $\blacktriangleleft_0 \leftarrow 1$
All $j \in \phi_1$ are assigned to equivalence class $J[1]$
$r_1^{\blacktriangleleft} \leftarrow$ *identity* **and** $\blacktriangleleft_1 \leftarrow 1$

Calculating a closed form expression for the time complexity of the lazy evaluation algorithm 3 is not straightforward. It is easy to show that it bounded by $\mathcal{O}(nk)$, since the first loop is worst-case $\mathcal{O}(k)$. However, we find experimentally that asymptotically, this lazy evaluation component does not contribute to overall computational complexity. (See Fig. 6)

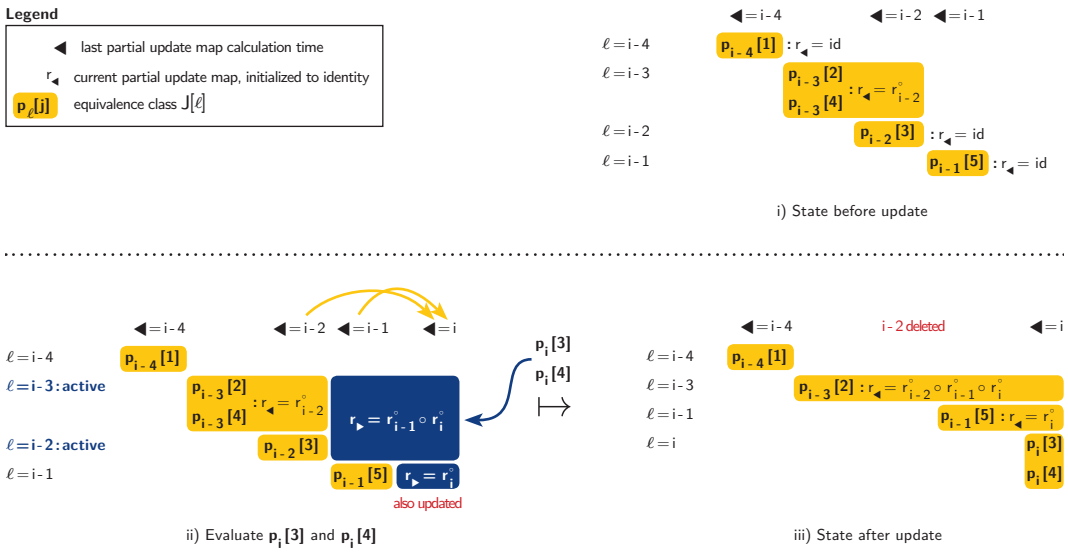

Figure 3 Top: Example of lazy evaluation state. **Bottom:** Example of update step as described in Algorithm 3

5 Results

5.1 Implementation

Our algorithm was implemented as a C++ library located at `https://github.com/yoheirosen/sublinear-Li-Stephens` Details of 3 will be found there.

We also implemented the linear time monoploid Li and Stephens forward algorithm in C++ as to evaluate it on identical footing. Profiling was performed using a single Intel Xeon X7560 core running at 2.3 GHz on a shared memory machine. Our reference panels H were the phased haplotypes from the 1000 Genomes [2] phase 3 vcf records for chromosome 22 and subsamples thereof. Haplotypes o were randomly generated simulated descendants.

5.2 Minor allele frequency distribution for the 1000 Genomes dataset

We simulated haplotypes o of $1,000,000$ bp length on chromosome 22 and recorded the sizes of the sets $\phi_i(o_i)$ for $k = 5008$. These data produced a mean $|\phi_i(o_i)|$ of 59.9 ($\frac{|\phi_i(o_i)|}{k} = 0.012$ for k = 5008.) The distribution (Fig. 4) is skewed toward low frequencies; the minor allele is unique at 71% of sites, and it is below 1% frequency at 92% of sites.

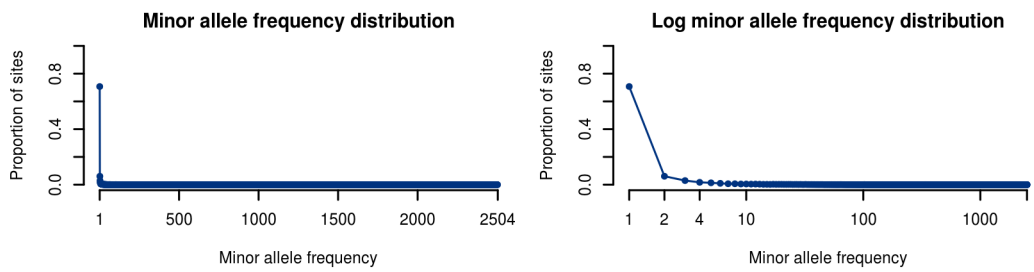

Figure 4 Biallelic site minor allele frequency distribution from 1000 Genomes chromosome 22

Algorithm 3: Lazy evaluation recurrence

Input: active rows $\phi_i(o_i)$ **and** previous lazy evaluation state
$J_{active} \leftarrow$ image of $\phi_i(o_i)$ under the maps $j \mapsto J[\cdot]$
$\ell_{min} \leftarrow$ earliest ℓ such that $J[\ell] \in J_{active}$
$r_i^{\blacktriangleright} \leftarrow r_i^{\circ}$
for $\lambda = i - 1$ ***to*** ℓ_{min} **do**
  $r_\lambda^{\blacktriangleright} \leftarrow r_{\lambda+1}^{\blacktriangleright} \circ r_\lambda^{\circ}$
  $r_i^{\blacktriangleleft} \leftarrow$ *identity* **and** $\blacktriangleleft_i \leftarrow i$ for one ℓ with $\blacktriangleleft_\ell = \lambda$
for $J[\ell] \in J_{active}$ *with* $\blacktriangleleft_\ell \neq i$ **do**
  $r_\ell^{\blacktriangleleft} \leftarrow r_\lambda^{\blacktriangleright} \circ r_\ell^{\blacktriangleleft}$ **and** $\blacktriangleleft_\ell \leftarrow i$
for $j \in \phi_i$ **do**
  Evaluate $p_i[j] = r_\ell^{\blacktriangleleft}(p_{\ell(j)}[j])$ **and** reassign j to $J[i]$
$r_i^{\blacktriangleleft} \leftarrow$ *identity* **and** $\blacktriangleleft_i \leftarrow i$

5.3 Comparison of our algorithm with the linear time forward algorithm

For $k = 5008$, on average, time per site is 37 μs for our algorithm and 1308 μs for the linear LS algorithm. For the forthcoming 100,000 Genomes Project, these numbers can be extrapolated to 251 μs for our algorithm and 260,760 μs for the linear LS algorithm.

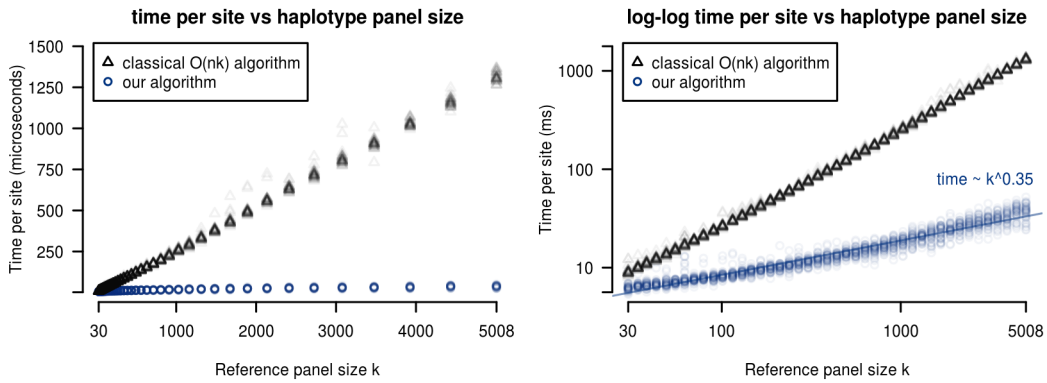

Figure 5 Runtime per site as a function of haplotype reference panel size k for our algorithm (blue) as compared to the classical linear time algorithm (black)

5.3.1 Lazy evaluation of dynamic programming rows

In the average case, the time complexity of our lazy evaluation algorithm does not contribute to the overall time complexity of the algorithm. (Fig. 6, right) The lazy evaluation runtime also contributes minimally to the total runtime of our algorithm. (Fig. 6, left)

5.4 Sparse haplotype encoding

5.4.1 Generating our sparse vectors

We generated the haplotype panel data structures from (§2) using the vcf-encoding tool `vcf2slls` which we provide. We built indices with multiallelic sites, which increases their

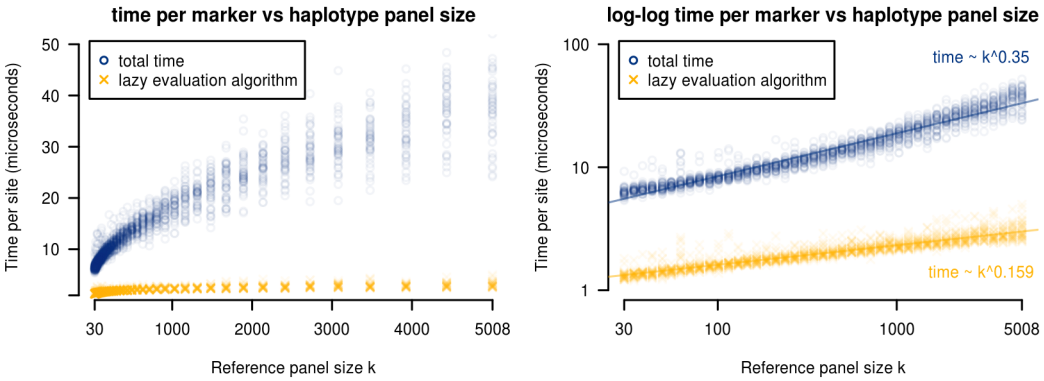

Figure 6 Time per site for the lazy evaluation subalgorithm (yellow) vs. the full algorithm (blue)

time and memory profile relative to the results in (§5.2) but allows direct comparison to vcf records. Encoding of chromosome 22 was completed in 38 minutes on a single CPU core. Use of M CPU cores will reduce runtime proportional to M.

5.4.2 Size of sparse haplotype index

In uncompressed form, our whole genome `*.slls` index for chromosome 22 of the 1000 genomes dataset was 285 MB in size versus 11 GB for the vcf record using `uint16_t`'s to encode haplotype ranks. When compressed with gzip, the same index was 67 MB in size versus 205 MB for the vcf record.

In the interest of speed (both for our algorithm and the $\mathcal{O}(nk)$ algorithm) our experiments loaded entire chromosome sparse matrices into memory and stored haplotype indices as `uint64_t`'s. This requires on the order of 1 GB memory for chromosome 22. For long chromosomes or larger reference panels on low memory machines, algorithm can operate on sequential chunks of the reference panel.

6 Discussion and significance

To the best of our knowledge, ours is the first forward algorithm for any haplotype model to attain sublinear time complexity with respect to reference panel size. Our algorithms could be incorporated into haplotype inference strategies by interfacing with our C++ library. This opens the potential for tools which are tractable on haplotype reference panels at the scale of current 100,000 to 1,000,000+ sample sequencing projects.

6.1 Applications which use individual forward probabilities

Our algorithm attains its runtime specifically for the problem of calculating the single overall probability $P(o|H, \rho, \mu)$ and does not compute all nk forward probabilities. We can prove that if m many specific forward probabilities are also required as output, and if the time complexity of our algorithm is $\mathcal{O}(\sum_i |\phi_i|)$, then the time complexity of the algorithm which also returns the m forward probabilities is $\mathcal{O}(\sum_i |\phi_i| + m)$.

In general, haplotype phasing or genotype imputation tools use stochastic traceback or other similar sampling algorithms. The standard algorithm for stochastic traceback samples states from the full posterior distribution and therefore requires all forward probabilities.

The algorithm output and lower bound of its speed is therefore $\mathcal{O}(nk)$. The same is true for many applications of the forward-backward algorithm.

There are two possible approaches which might allow runtime sublinear in k for these applications. Using stochastic traceback as an example, first is to devise an $\mathcal{O}(f(m))$ sampling algorithm which uses $m = g(k)$ forward probabilities such that $\mathcal{O}(f \circ g(k)) < \mathcal{O}(k)$. The second is to succinctly represent forward probabilities such that nested sums of the nk forward probabilities can be queried from $\mathcal{O}(\phi) < \mathcal{O}(nk)$ data. This should be possible, perhaps using the positional Burrows-Wheeler transform [5] as in [11], since we have already devised a forward algorithm with this property for a different model in [13].

6.2 Generalizability of algorithm

The optimizations which we have made are not strictly specific to the monoploid Li and Stephens algorithm. Necessary conditions for our reduction in the time complexity of the recurrence relations are

▶ Condition 1. The number of distinct transition probabilities is bounded.

▶ Condition 2. The number of distinct emission probabilities is bounded.

Favourable conditions for efficient time complexity of the lazy evaluation algorithm are

▶ Condition 1. The number of unique update maps added per step is bounded.

▶ Condition 2. The update map extension operation is composition of matrices of bounded size. This can be generalized to a broad algebraic class[5] of update operations provided that they have bounded runtime.

The reduction in time complexity of the recurrence relations depends on the Markov property, however we hypothesize that the delayed evaluation needs only the semi-Markov property.

6.2.1 Other haplotype forward algorithms

Our optimizations are of immediate interest for other haplotype copying models. The following related algorithms have been explored without implementation.

▶ Example (Diploid Li and Stephens). We have yet to implement this model but expect average runtime at least subquadratic in reference panel size k. We build on the statement of the model and its optimizations in [9]. We have found the following recurrences which may be combined with a system of lazy evaluation algorithms:

▶ **Lemma 11.** *The diploid Li and Stephens HMM may be expressed using the recurrences*

$$S_{i-1} = \alpha_c S_{i-1} + \beta_c \sum_{j \in \phi_i} S_{i-1}(j) + \gamma_c \sum_{(j_1, j_2) \in \phi_i^2} p_{i-1}[j_1, j_2] \qquad \mathcal{O}(|\phi_i|^2) \tag{19}$$

$$S_i(j) = \alpha_c S_{i-1} + \beta_c S_{i-1}(j) + \gamma_c \sum_{j_2 \in \phi_i} p_{i-1}[j, j_2] \qquad \mathcal{O}(|\phi_i|^2) \tag{20}$$

$$p_i[j_1, j_2] = \alpha_p p_{i-1}[j_1, j_2] + \beta_p (S_{i-1}(j_1) + S_{i-1}(j_2)) + \gamma_p S\, \mathcal{O}(|\phi_i|) \qquad \mathcal{O}(1) \tag{21}$$

where $\alpha_{(\cdot)}, \alpha_{(\cdot)}, \alpha_{(\cdot)}$ depend only on the diploid genotype o_i.

▶ Example (Multipopulation Li and Stephens). [4] We maintain separate sparse haplotype panel representations $\phi_i^A(o_i)$ and $\phi_i^B(o_i)$ and separate lazy evaluation mechanisms for the two populations A and B. Expected runtime guarantees are similar.

[5] Specifically, any collection of operations forming a *category* in the sense of category theory

This model, and versions for > 2 populations, will be important in large sequencing cohorts (such as NHLBI TOPMed) where assuming a single related population is unrealistic.

▶ Example (More detailed mutation model). It may also be desirable to model distinct mutation probabilities for different pairs of alleles at multiallelic sites. Runtime is worse than the biallelic model but remains average case sublinear.

▶ Example (Sequence graph Li and Stephens analogue). In [13] we described a hidden Markov model for a haplotype-copying with recombination but not mutation in the context of sequence graphs. Assuming we can decompose our graph into nested sites then we can achieve a fast forward algorithm with mutation.

▶ Example (Semi-Markovian recombination model). The lazy evaluation algorithm 3 may efficiently allow time-since-recombination dependent transition probabilities.

References

1. Brian L Browning and Sharon R Browning. A unified approach to genotype imputation and haplotype-phase inference for large data sets of trios and unrelated individuals. *The American Journal of Human Genetics*, 84(2):210–223, 2009.
2. 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68, 2015.
3. Olivier Delaneau, Jean-Francois Zagury, and Jonathan Marchini. Improved whole-chromosome phasing for disease and population genetic studies. *Nature methods*, 10(1):5, 2013.
4. Peter Donnelly and Stephen Leslie. The coalescent and its descendants. *arXiv preprint arXiv:1006.1514*, 2010.
5. Richard Durbin. Efficient haplotype matching and storage using the positional burrows–wheeler transform (pbwt). *Bioinformatics*, 30(9):1266–1272, 2014.
6. Alon Keinan and Andrew G Clark. Recent explosive human population growth has resulted in an excess of rare genetic variants. *science*, 336(6082):740–743, 2012.
7. John Frank Charles Kingman. The coalescent. *Stochastic processes and their applications*, 13(3):235–248, 1982.
8. Na Li and Matthew Stephens. Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data. *Genetics*, 165(4):2213–2233, 2003.
9. Yun Li, Cristen J Willer, Jun Ding, Paul Scheet, and Gonçalo R Abecasis. Mach: using sequence and genotype data to estimate haplotypes and unobserved genotypes. *Genetic epidemiology*, 34(8):816–834, 2010.
10. Po-Ru Loh, Petr Danecek, Pier Francesco Palamara, Christian Fuchsberger, Yakir A Reshef, Hilary K Finucane, Sebastian Schoenherr, Lukas Forer, Shane McCarthy, Goncalo R Abecasis, et al. Reference-based phasing using the haplotype reference consortium panel. *Nature genetics*, 48(11):1443, 2016.
11. G Lunter. Fast haplotype matching in very large cohorts using the Li and Stephens model. biorxiv (2016).
12. Jared O'Connell, Kevin Sharp, Nick Shrine, Louise Wain, Ian Hall, Martin Tobin, Jean-Francois Zagury, Olivier Delaneau, and Jonathan Marchini. Haplotype estimation for biobank-scale data sets. *Nature genetics*, 48(7):817, 2016.
13. Yohei Rosen, Jordan Eizenga, and Benedict Paten. Modelling haplotypes with respect to reference cohort variation graphs. *Bioinformatics*, 33(14):i118–i123, 2017.
14. Amy L Williams, Nick Patterson, Joseph Glessner, Hakon Hakonarson, and David Reich. Phasing of many thousands of genotyped samples. *The American Journal of Human Genetics*, 91(2):238–251, 2012.