ABSTRACT
Motivation: A promising class of methods for large-scale population genomic inference use the conditional sampling distribution (CSD), which approximates the probability of sampling an individual with a particular DNA sequence, given that a collection of sequences from the population has already been observed. The CSD has a wide range of applications, including imputing missing sequence data, estimating recombination rates, inferring human colonization history and identifying tracks of distinct ancestry in admixed populations. Most well-used CSDs are based on hidden Markov models (HMMs). Although computationally efficient in principle, methods resulting from the common implementation of the relevant HMM techniques remain intractable for large genomic datasets.

Results: To address this issue, a set of algorithmic improvements for performing the exact HMM computation is introduced here, by exploiting the particular structure of the CSD and typical characteristics of genomic data. It is empirically demonstrated that these improvements result in a speedup of several orders of magnitude for large datasets and that the speedup continues to increase with the number of sequences. The optimized algorithms can be adopted in methods for various applications, including the ones mentioned above and make previously impracticable analyses possible.

Availability: Software available upon request.

Supplementary Information: Supplementary data are available at Bioinformatics online.

Methods incorporating the CSD generally fall into one of several categories. Likelihoods can be approximated using CSD-based importance sampling, or directly as a product of CSDs (Li and Stephens, 2003; McVean et al., 2004; Fearnhead and Donnelly, 2009) coupled with composite methods (Fearnhead and Donnelly, 2008a) or as a product of CSDs (Li and Stephens, 2003; McVean et al., 2004). In conjunction with expectation-maximization or Markov chain Monte Carlo, these methods have been used for estimation of fine-scale recombination rates (Crawford et al., 2004; Fearnhead and Smith, 2002; Li and Stephens, 2004; McVean et al., 2004), gene conversion parameters (Crawford et al., 2004; Yin et al., 2004), population demography (Davison et al., 2004; Hellenthal et al., 2008; and population structure (Lawson et al., 2013). It is also possible to infer and use the hidden states in the HMM CSD to infer and use the hidden states in the HMM CSD. This has been used for admixture inference (Price et al., 2006; Sundquist et al., 2008; Wegmann et al., 2010), in which genomic segments corresponding to ancestral populations are identified and also within a pseudo-Gibbs sampling framework to phase genotype sequence data into haplotype sequence data and to impute missing data (Howie et al., 2009; Li et al., 2010; Marchini et al., 2005; Stephens and Scheet, 2005).
Nearly, all of these methods rely on iterative Monte Carlo or expectation–maximization techniques. As a result, they are computationally intensive, often requiring several hours, or, in some cases, days, to produce a result, even for modest non-genomic datasets \cite{Howie et al. 2009}, directly extending the methods to large genomic datasets is thus often impractical. Moreover, nearly all of the running time is expended on CSD computation, and so the choice of CSD is often made on the basis of efficiency and (arguably) at the expense of accuracy \cite{Browning and Browning 2007, Howie et al. 2009, Li and Stephens 2008, Schecter and Stephens 2009, Stephens and Schecter 2009, Stephens and Schecter 2009}.

In this article, we help to overcome these limitations by proposing two related optimizations to the relevant HMM-based CSD computations. Consider sampling a large number of sequences from a population. If the sampled sequences are very long, it is likely that nearly all of them will be unique. However, for most relatively short regions, the number of unique subsequences will be reduced. This intuition forms the basis of the first optimization, which locally increases to about 1850 assumptions on mutation and recombination rates, this speedup computing to show the applicability to other HMM-based CSDs, including provide two sufficient conditions for our optimizations and use them extending this line of work to many sequences. On simulated data, \pi indicates a recent common ancestor, which in turn indicates a this optimum.

A second common feature of the sampled sequences is an abundance of non-polymorphic sites. These sites are informative— for example, a local over-abundance of non-polymorphic sites indicates a recent common ancestor, which in turn indicates a low propensity for recombination, and should be included in the analysis. Indeed, \cite{Li and Durbin 2001} used the physical distribution of polymorphic and non-polymorphic sites between a pair of sequences to infer past population sizes of humans. Using the fact that non-polymorphic sites do not differentiate the sequences, we show that it is possible to reduce the complexity of the CSD computation at non-polymorphic sites. We stress that our solution is different from simply ignoring non-polymorphic sites; we are proposing algorithmic improvements to incorporate non-polymorphic sites into the analysis in an efficient way.

In formally describing and evaluating our optimizations, we restrict attention to the most accurate HMM-based CSD, \picking, proposed by \cite{Paul et al. 2011} and consider the problem of computing the conditional sampling probability (CSP), denoted \picking(a|m), of a particular individual \(a\) given a collection \(m\) of previously sampled individuals. Incidentally, in the case the size \(n\) of the previously observed sample \(m\) is 1, the HMM underlying \picking is equivalent to the HMM used in the aforementioned work of \cite{Li and Durbin 2001}. We anticipate that \picking provides one way of extending this line of work to many sequences. On simulated data, our algorithmic improvement leads to a speedup of about 550\% for \(n = 5000\) previously sampled individuals; by making regularity assumptions on mutation and recombination rates, this speedup increases to about 1850\%. Importantly, we show that the empirically observed speed-up increases with the number \(n\) of previously sampled individuals.

Although we describe our optimizations in the context of computing \picking, they are more generally applicable. We provide two sufficient conditions for our optimizations and use them to show the applicability to other HMM-based CSDs, including those of \cite{Birdhead and Donnelly 2001} and \cite{Li and Stephens 2003}, as well as CSDs for more complex demographic models and population genetic HMMs. Also, in the Supplementary Material, we describe extending our algorithms to allow for efficient inference of hidden states, often termed posterior decoding.

We stress that the work presented here is fundamentally different from previous works on "approximating" CSD-based population genetic inference \cite{Howie et al. 2009} and consider a fixed-size subset of the haplotype configuration \(n\), chosen using a measure of 'closeness' to the sampled haplotype \(u\). In order to reduce the state space of the HMM-based CSD and speed up computations \cite{Schecter and Stephens 2009} and \cite{Browning and Browning 2007}. Consider an HMM with reduced state space by compacting the configuration \(n\) into a substantially smaller haplotype model. More recently, \cite{Balakrishnan et al. 2013} have proposed an approximate HMM formulation relying on a partition of the sampled sequences similar to that proposed herein (Section 2.3). As described earlier, using such heuristics improves computational efficiency, but ultimately at the expense of accuracy. The purpose of this article is to provide highly optimized 'and exact' computation for a large class of approximate CSDs, rather than to introduce additional approximations to the underlying models.

## 2 METHODS

Herein, we describe the HMM formulation of \(\hat{\pi}_{\text{SMC}}\), the algorithms that are currently being used to compute \(\hat{\pi}_{\text{SMC}}(a|m)\) and the optimizations we are proposing to improve the running time.

We remark that the theoretical analysis of our algorithms is limited to asymptotic time (and space) complexity. As a measure of real-world performance, asymptotic analyses often leave much to be desired. Consider, for example, a sample in which 1 out of every 1000 sites is polymorphic. If we denote by \(k\) the total number of sites and \(k_p\) the number of polymorphic sites, then formally \(O(k) = O(k_p)\). Nevertheless, we would like to distinguish between an algorithm that operates on each of the \(k\) sites and an algorithm that operates only on the \(k_p\) polymorphic sites, as the latter will be some 1000\% faster; thus we write the complexities for the two algorithms as \(O(k)\) and \(O(k_p)\), respectively.

### 2.1 Notation

We consider haplotypes in the finite-locus finite-alleles setting. Throughout, we denote by \(H:\{0, 1\}^n\) the haplotype configuration, \(\hat{H}\) a sample configuration of haplotypes, \(\hat{\pi}_{\text{SMC}}\), \(\pi_{\text{SMC}}\) the sub-haplotype for a range of loci \(\ell \leq C\), \(\pi_{\text{SMC}}\) the sub-haplotype for a range of loci \(\ell \leq C\) is denoted by \(\pi_{\text{SMC}}(\ell, C)\). A sample configuration of haplotypes is specified by a vector \(n = (n_0, n_1, \ldots, n_{C-1})\), with \(n_i\) being the number of haplotypes of type \(i\) in the sample. The set of 'unique' haplotypes associated with configuration \(n\) is denoted by \(\hat{H}_{\text{uni}}(n)\), \(\pi_{\text{SMC}}(\ell, C)\), \(\pi_{\text{SMC}}(\ell, C)\). Finally, the total number of haplotypes is denoted by \(|n| = n\), the number of unique haplotypes by \(\hat{H}_{\text{uni}}(n)\) and the number of polymorphic loci, which generally depends on the sample \(n\) by \(k_p\).

### 2.2 A brief description of \(\hat{\pi}_{\text{SMC}}\)

Suppose that, conditioned on having already observed a haplotype configuration \(n\), we wish to sample a new haplotype \(u\). By generalizing the technique of \cite{Birdhead and Donnelly 2001} based on the diffusion process, \cite{Paul et al. 2011} introduced the CSD \(\pi_{\text{SMC}}\), intended to approximate key properties of the coalescent with recombination, the model under which inference is to be performed. The central idea of \(\pi_{\text{SMC}}\) is to fix the unknown ancestry of \(n\) to be the 'trunk genealogy' \(A(n)\), in which lineages associated with the haplotypes do not mutate, recombine, or coalesce with one another, but rather
order to use standard dynamic programming methodologies for inference, the idea is to consider the conditional genealogy \(\text{CSP}\) for the conditional genealogy limitation, Paul et al. (2015) found that the Markov approximation underlying \(\hat{\text{C}}\) has been absorbed, the process terminates; the type of every lineage of \(C\) evolves backwards in time subject to mutation, recombination, coalescence and 'absorption' into one of the lineages of \(A(\alpha)\). When every lineage of \(C\) has been absorbed, the process terminates; the type of every lineage of \(C\) is now determined and a sample for \(\alpha\) is generated.

Although a recursion for computing the CSP \(\hat{\text{CSP}}(\alpha|\omega)\) is known, it is computationally intractable for all but the smallest datasets. To remedy this limitation, Paul et al. (2015) adopted a sequentially Markov framework (McVean andCardin, 2003) for the conditional genealogy \(\text{C}\). The central idea is to consider the 'marginal' conditional genealogy \(\tilde{c}_i\) associated with haplotype \(\omega\) is sampled; within \(\text{C}\), lineages evolve backwards in time subject to mutation, recombination, coalescence and 'absorption' into one of the lineages of \(A(\alpha)\). When every lineage of \(\text{C}\) has been absorbed, the process terminates; the type of every lineage of \(\text{C}\) is now determined and a sample for \(\alpha\) is generated.

In general, the random sequence of marginal conditional genealogies is not Markov, due to the potential for coalescence events within the conditional genealogy. Nonetheless, it is possible to 'approximate' this sequence as Markov by using a two-locus transition distribution. Mutation can then be realized at each locus independently as a Poisson process on sequence as Markov by using a two-locus transition distribution. Mutation and recombination can be generalized to accommodate an arbitrary distribution over haplotypes \(\hat{h}\) in the case of a recombination; in fact, the algorithms stated, the transition distribution imposes a uniform distribution on the absorbing haplotype in the case of a recombination, in fact, the algorithms can be generalized to accommodate an arbitrary distribution over haplotypes that does not depend on \(d^*\) or \(\hat{\text{C}}\). In Section 2.4, we discuss the applicability of these optimizations to more general (and more specialized) classes of approximate CSDs:

\section*{2.4 Improving efficiency through the transition distribution}

Consider the description of \(\hat{\text{CSP}}\) given above and more rigorously defined in (Fearnhead and Donnelly, 2001). If a recombination does not occur between loci \(\ell=1\) and \(\ell\), then \((d_{\ell-1},\ldots,d_1) = (d_\ell,\ldots,d_1)\); moreover, if recombination does occur, the absorbing haplotype \(h_\ell\) is independent of \(h_{\ell-1}\) and uniformly distributed. As a result, we have the following property (We remark that Property 1 is a sufficient, though not necessary, condition for the algorithmic optimizations described in this and subsequent sections. For example, as stated, the transition distribution imposes a uniform distribution on the absorbing haplotype in the case of a recombination, in fact, the algorithms can be generalized to accommodate an arbitrary distribution over haplotypes that does not depend on \(d^*\) or \(\hat{\text{C}}\).

\section*{Property 1. The initial and transition probabilities \(\xi\) and \(\phi\) take the following functional form:}

\[\xi(d,h) = \sum_{\omega} \frac{\hat{n}_\omega}{\sum_{\omega} \hat{n}_\omega}, \quad \phi(d,h) = \sum_{\omega} \frac{\hat{n}_\omega}{\sum_{\omega} \hat{n}_\omega},\]

where \(\xi(d,h)\) is known analytic expressions. Using Property 1 in conjunction with definitions

\[Q(d) = \sum_{\omega} Q_{d\omega}(d), \quad U_\ell(d) = \sum_{\omega} U_{\ell\omega}(d),\]

we can express equations (1) and (2) as

\[\hat{\text{CSP}}(\alpha|\omega) = \sum_{d\omega} Q_{d\omega}(d),\]

and with base case

\[F_0(d,h) = \xi(d,h) = \sum_{\omega} \frac{\hat{n}_\omega}{\sum_{\omega} \hat{n}_\omega},\]

Using these recursions, the dynamic program in Algorithm 1 can be used to compute \(\hat{\text{CSP}}(\alpha|\omega)\). Within the pseudocode description, the time complexity of Lines 6, 7 and 8 are \(O(\hat{n}_\omega)\) and \(O(n_\omega)\), respectively. As a result, the time complexity of Lines 5-9, and for the algorithm as a whole, is \(O(n_\omega + \hat{n}_\omega)\). This represents a substantial improvement over the quadratic dependence on \(n_\omega\) in the naive forward algorithm for HMMs. This simple optimization has already been generally adopted in Hidden Markov Models (HMMs) on our website and serves as a baseline for improvement.
Algorithm 1 Compute \( \mathcal{N}(s,n) \) using a forward-type recursion improved by considering Property 3.

1. for all \( s \in S \) do
2. Compute \( F_0(d,h) \) by \( \mathcal{D} \), \( \forall h \in \mathcal{H}_0 \)
3. Compute \( \mathcal{Q}_0(d) \) using \( \mathcal{D} \)
4. end for
5. for \( s = 1 \rightarrow k \) and \( d \in \mathcal{D} \) do
6. Compute \( U_{s-1}(d) \) using \( \mathcal{D} \)
7. Compute \( F_s(d,h) \) using \( \mathcal{D} \), \( \forall h \in \mathcal{H}_s \)
8. Compute \( \mathcal{Q}_s(d) \) using \( \mathcal{D} \)
9. end for
10. Compute \( \mathcal{N}(s,n) \) using \( \mathcal{D} \)

### 2.5 Improving efficiency through the emission distribution

The state of the HMM at locus \( \ell \) is a tuple \((d_i,h_i)\). However, the emission probability of \( u(\ell) \) is governed only by the time interval \( d_i \) and the allele \( h_i \). As a result, the following property holds:

**Property 2.** Consider a subset \( B \subseteq \mathcal{H}_s \) such that there exists an allele \( a \) with \( h(\ell) = a \) for all \( h \in B \). Then, for each absorption interval \( d \in \mathcal{D} \), the emission distribution \( \pi(\ell, h) \) is identical for all \( h \in B \). We indicate this fact by writing \( \pi(\ell, h) = \pi(\ell, B) \) for all \( h \in B \).

With this in mind, define a ‘partition’ \( C \) of the haplotype configuration \( n \) to be a collection of blocks of the form \((B_i, e_i, e_i')\), such that

- For every \((B_i, e_i, e_i') \in C\), there exists a sub-haplotype \( x \) such that \( h(\ell, e_i, e_i') = x \) for all \( h \in B \).
- For every haplotype \( h \in \mathcal{H}_s \) and \( 1 \leq i \leq k \), there exists ‘exactly’ one \((B_i, e_i, e_i') \in C\) with \( h(\ell, e_i, e_i') = h \) and \( e_i \leq e_i' \).

For a given locus \( \ell \), a configuration partition \( C \) induces a partition of the haplotypes \( \mathcal{H}_s \), denoted by \( \mathcal{C}_\ell \), and Property 3 applies to each \( B \in \mathcal{C}_\ell \). In the next sections, we present new sets of recursions and dynamic programming algorithms valid for an arbitrary partition \( C \). The computational complexity of these algorithms will depend on \( C \) through two functions, namely \( \Psi(C) \) and \( \Omega(C) \), defined as follows. For locus \( \ell \), define \( \psi(C) = \lvert \mathcal{C}_\ell \rvert \), the number of blocks in \( \mathcal{C}_\ell \) and define \( \omega(C) \) to be the total number of haplotypes in blocks of the configuration partition ‘ending’ at locus \( \ell \):

\[
\Psi(C) = \sum_{i=1}^{k} \psi(C_i) = \sum_{i=1}^{k} \lvert \mathcal{C}_i \rvert, \\
\Omega(C) = \sum_{i=1}^{k} \omega(C_i) = \sum_{i=1}^{k} \lvert \mathcal{H}_s \rvert.
\]

In some cases, we are primarily concerned with polymorphic loci, and so we define \( \Psi(C) \) to be the analog of \( \Psi(C) \) summed over ‘only’ polymorphic loci.

Finally, we define the trivial partition \( C_T \) for haplotype configuration \( n \) as the partition containing a single block \((\{1\}, \{1\}, 1)\) for each \( h \in \mathcal{H}_s \). Note that \( \Psi(C_T) = k \cdot n_h \) and \( \Omega(C_T) = n_h \). See Figure 2 for an illustration of both \( C_T \) and two non-trivial configuration partitions.

#### 2.5.1 A general configuration partition of \( n \)

Let \( C \) be a configuration partition of \( n \). Begin by defining

\[
Q_s(d,B) = \sum_{h \in \mathcal{H}_s} F_s(d,h) \quad \text{so that} \quad Q_s(d) = \sum_{B \in \mathcal{C}_s} Q_s(d,B),
\]

and for \( s = 0 \) we have \( Q_0(d) = F_0(d,h) \).

Now suppose \((B_e, e_e, e_e') \in \mathcal{C}_\ell \). Applying Definition 3 and Property 3 to equation (8), we get

\[
Q_s(d,B) = \pi(\ell, B) \left[ \sum_{B_h \in \mathcal{C}_{s+1}} Q_{s+1}(d,h) + \frac{\omega(B_h)}{\omega(C)} \right].
\]

**Fig. 2.** Illustration of three alternative configuration partitions. Each row represents a haplotype, with white and black circles representing the allele at each of eight polymorphic loci. The line style of each sub-haplotype indicates the block to which it belongs. (a) The trivial configuration partition \( C_T \); \( \Psi(C_T) = 40 \) and \( \Omega(C_T) = 5 \). (b) A non-trivial configuration partition, \( C \); \( \Psi(C) = 24 \) and \( \Omega(C) = 12 \). (c) The configuration partition \( C \) found by the algorithm described in Section 3. For \( s = 3 \), \( \Psi(C) = 24 \) and \( \Omega(C) = 15 \), where \( n_h = \sum n_h \).

Similarly, by induction and making use of equations (8) and (9), it is possible to show that, for \( e_e \leq e_0 \leq e_1 \), and \( h \in \mathcal{H}_s \),

\[
F_s(d,h) = \pi(\ell, B) \sum_{B_h \in \mathcal{C}_{s+1}} Q_{s+1}(d,h) + \frac{\omega(B_h)}{\omega(C)}.
\]

Finally, we define the trivial partition \( C_T \) for haplotype configuration \( n \) as the partition containing a single block \((\{1\}, \{1\}, 1)\) for each \( h \in \mathcal{H}_s \). Note that \( \Psi(C_T) = k \cdot n_h \) and \( \Omega(C_T) = n_h \). See Figure 2 for an illustration of both \( C_T \) and two non-trivial configuration partitions.

#### 2.5.2 The absence of polymorphism

In many reasonable evolutionary scenarios, a great many loci will not be polymorphic. Accommodating such loci in the analysis is important and can be done efficiently making use of Property 3. In particular, for a non-polymorphic locus \( \ell \), Property 3 applies to the trivial set \( B_0 = \mathcal{H}_s \) and therefore the emission distribution can be written \( \pi(\ell, (d,B_0)) = \pi(\ell, d) \) and moreover, \( Q_s(d,B_0) = Q_s(d) \).

Suppose consecutive loci \( \ell_1, \ldots, \ell_s \) are not polymorphic. Rewriting equations (8) and (9) for block \((B_h, \ell_s, \ell_s')\) yields, for \( e_s \leq e_s' \),

\[
Q_s(d,B) = \pi(\ell, B) \left[ \sum_{B_h \in \mathcal{C}_{s+1}} Q_{s+1}(d,h) + \frac{\omega(B_h)}{\omega(C)} \right].
\]

and for \( e_s \leq e_s' \) and \( h \in B_h = \mathcal{H}_s \),

\[
F_s(d,h) = \pi(\ell, B) \sum_{B_h \in \mathcal{C}_{s+1}} Q_{s+1}(d,h) + \frac{\omega(B_h)}{\omega(C)}.
\]

where \( T_s(d) = \pi(\ell, B) \sum_{B_h \in \mathcal{C}_{s+1}} Q_{s+1}(d,h) \) and solves the recursion

\[
T_s(d) = \pi(\ell, B) \sum_{B_h \in \mathcal{C}_{s+1}} Q_{s+1}(d,h)
\]

for \( e_s \leq e_s' \), with base case \( T_1(d) = 1 \).
Algorithm 2 Compute $\hat{p}_{\text{base}}(\omega n)$ using a forward-type recursion improved by considering Properties 1 and 2 for a configuration partition $C$.

1. for all $d \in D$ do
2. Compute $T_{0}(d, h)$ using \[5\] if $h \in H_0$
3. Compute $Q_t(d, B)$ using \[6\] and $T_{0}(d, B) = 1$, $\forall B, (l, e) \in C$
4. Compute $Q_t(d)$ using \[7\]
5. end for
6. for $l = 1 \rightarrow k$ and $d \in D$ do
7. Compute $U_{l-1}(d)$ using \[9\]
8. Compute $Q_{t}(d, B)$ and $T_{l}(d, B)$ using \[4\] and \[8\], $\forall B, (l, e) \in C$ such that $l \leq \ell = \ell_t$; compute $Q_{t}(d)$ using \[7\]
9. Compute $T_{l}(d, B)$ using \[10\], $\forall h \in B$ and $\forall B, (l, e) \in C$
10. Compute $Q_{t}(d, B)$ and $T_{l}(d, B) = 1$, $\forall B, (l, e) \in C$
11. end for
12. Compute $\hat{p}_{\text{base}}(\omega n)$ using \[11\].

Algorithm 3 Computation of $\hat{p}_{\text{base}}(\omega n)$ improved by considering Properties 1 and 2 and a special case for non-polymorphic loci, for a configuration partition $C$ such that $\forall B, (l, e) \in C$, $\ell_t$ is polymorphic.

1. Algorithm 2 lines 1–5; and set $T_{0}(d) = 1 \forall d \in D$ and $C = 1$
2. for $l = 1 \rightarrow k$ and $d \in D$ do
3. if locus $l$ is polymorphic then
4. if locus $l$ is not polymorphic then
5. Compute $Q_{t-1}(d, B)$ and $T_{l-1}(d, B)$ using \[8\] and \[9\]
6. end if
7. Algorithm 2 lines 7–10
8. Set $T_{l}(d) = 1$ and $C = l + 1$
9. else
10. Compute $U_{l-1}(d)$, $Q_{t}(d)$, and $T_{l}(d)$ using \[4\], \[5\], and \[6\]
11. end if
12. end for
13. Compute $\hat{p}_{\text{base}}(\omega n)$ using \[11\].

Now, let $C$ be a configuration partition with $(B, l, e) \in C$. Suppose that there is a stretch of non-polymorphic loci $l_1, \ldots, l_h$ such that $l \leq l_1 \leq \ldots \leq l_h$. Applying Definition 2 to equation \[11\], yields, for $C = k_c$, \[12\]

\[Q_{t}(d) = T_{l}(d)Q_{t_{l_{h}}}Q_{t_{l_{h-1}}} \cdots Q_{t_{l_{1}}}(d) \text{ if } l \leq l_1 \leq \ldots \leq l_h \]

Similarly, considering the definition of $T_{l}(d, B)$ along with equation \[11\],

\[T_{l}(d, B) = T_{l}(d)T_{l_{h}} \cdots T_{l_{i+1}}(d, B) \text{ if } l \leq l_1 \leq \ldots \leq l_h \]

Algorithm 2 can be modified to accommodate such stretches of non-polymorphic loci as a special case, making use of equations \[12\] and \[13\] to directly compute the values of $Q_{t}(d)$ and $T_{l}(d)$ at each non-polymorphic locus $l$. If we then assume (without loss of generality) that each $(B, l, e) \in C$ has $\ell_t$ at a polymorphic locus, then at the final non-polymorphic locus, for which $l = l_{t}$, equation \[12\] and \[13\] may be used to compute $Q_{t}(d, B)$ and $T_{l}(d, B)$, for each $B \in C$. This modification is detailed in Algorithm 3.

Within Algorithm 3 the time complexity of Lines 2 and 3 is $O(1)$, of Line 7 is $O(m + \psi(C))$, of Line 8 is $O(m + \psi(C))$, and of Line 10 is $O(m)$. As a result, the time complexity of Lines 2–12, and of the dynamic program, is $O(k_{t}m + m\psi(C) + \Omega(C))$. Relative to Algorithm 2 less computation needs to be done for non-polymorphic loci; thus, in the typical case of many non-polymorphic loci, this dynamic program will have a decreased running time. For $C = C_{Y}$, the time complexity is $O(k_{t}m^{2} + \psi(C))$.

Algorithm 4 Computation of $\hat{p}_{\text{base}}(\omega n)$ improved by considering Properties 1 and 2 and a special ‘optimized’ case for non-polymorphic loci, for a configuration partition $C$ such that $\forall B, (l, e) \in C$, $\ell_t$ is polymorphic.

1. Algorithm 2 line 1
2. for polymorphic $l \in [2 \rightarrow k]$ and $d \in D$ do
3. if locus $l$ is not polymorphic then
4. Compute $Q_{t-1}(d)$ and $T_{l-1}(d)$ using \[8\]
5. end if
6. Algorithm 2 lines 4–8
7. end for
8. Compute $\hat{p}_{\text{base}}(\omega n)$ using \[11\].

2.5.3 An optimization for non-polymorphic loci

The key recursions \[12\] and \[13\] for non-polymorphic loci can be written in matrix form as $Q_{t} = E_{m}Q_{t} + 1$, $Q_{t} = E_{m}Q_{t} + 1$, where $Q_{t}$ and $E_{m}$ are $m$-dimensional column vectors having $d$th entry $Q_{t}(d)$ and $T_{l}(d),$ respectively, and $E_{m}$ and $Q_{t}$ are $(m \times m)$-dimensional matrices having $d$th entry $e_{m}(d)$ and $m$th entry $1$, respectively; and $Z_{t}$ is an $(m \times m)$-dimensional matrix having $d$th entry $Z_{t}(d)$.

Now, suppose that the mutational model is symmetric and the mutation rate constant for all loci. Then, $E_{m} = 1$ and for all non-polymorphic loci $l$. Similarly, if the recombination rate between each pair of loci is constant, then $Z_{t} = 1$ and $Z_{t} = 0$ do not depend on $l$. With these assumptions, for $C = C_{Y}$, the key recursions \[12\] and \[13\] for $l \neq \ell$ can be ‘analytically’ skipped.

The modified dynamic program associated with this optimization is given in Algorithm 3. The time complexity of Line 4 is $O(mn)$, and of Line 6 is $O(m + \psi(C))$. Thus, the time complexity for the dynamic program is $O(k_{t}m^{2} + m\psi(C) + \Omega(C))$. This refinement once again reduces the computation required for non-polymorphic loci, and so we might expect substantial improvements in performance over Algorithms 2 and 3.

For the choice $C = C_{Y}$, the time complexity is $O(k_{t}m^{2} + m\psi(C) + \Omega(C))$. Note that the assumptions necessary for Algorithm 3 namely a symmetric mutation model and uniform mutation and recombination rates, can be relaxed, but at the expense of additional pre-computation. For example, given non-uniform, but locally similar, recombination rates; each stretch of non-polymorphic loci could then use the pre-computed values associated with the closest recombination rate.

2.6 A fast algorithm for configuration partitions

In Section 2.5 we assumed a configuration partition $C$ to be specified and showed that, for Algorithms 2 and 3 the time complexity depends on $C$ through the functions $\psi(C)$ or $\Omega(C)$ and $\Omega(C)$ and more particularly their sum. It is intuitively clear that a configuration partition minimizing $\psi$ as well as $\Omega$, and vice versa, and in general these quantities are inversely related; minimizing a convex combination of these quantities is therefore difficult. In this section, we propose a fast and simple parameterized algorithm for constructing reasonably good configuration partitions.

Given a configuration $C$, the algorithm proceeds sequentially over the loci: initially, let $\ell_{1} = 1$. Given $\ell_{s}$, find the largest polymorphic locus $\ell_{t_{s}}$ such that $\ell_{s} \leq \ell_{t_{s}} \leq k$, and the number of unique sub-haplotype between loci $\ell_{s}$ and $\ell_{t_{s}}$ is at most some threshold parameter $s$. Then, for each unique sub-haplotype $x$ between $\ell_{s}$ and $\ell_{t_{s}}$, group all $h \in H_0$ such that $\delta(\ell_{s}, x) = h$ into the same group.
Although these constants will depend on the implementation and hardware, on both asymptotic time complexity results, running time appears to depend linearly on the configuration partition \( s = \Psi_1/\Psi_1 \) of \( \Omega_1/\Psi_1 \). The configuration \( \Omega_1/\Omega_1 \) was generated using coalescent simulation for 500 individuals, each having \( 10^5 \) bi-allelic loci, using population-scaled mutation rate \( \theta = 0.005 \) per locus and population-scaled recombination rate \( \rho = 0.001 \) between each pair of adjacent loci, and resulting in \( \Psi_1 = 1724 \) polymorphic loci and \( n_0 = 324 \) unique haplotypes.

Recall that Algorithm H with \( C = \Psi_1 \) is equivalent to Algorithm E.

**3 RESULTS**

We have presented three optimized algorithms for computing the conditional sampling probability (CSP) \( \hat{\pi}_{\text{SMC}}(a|n) \). Briefly, Algorithm E-H rely on a partition \( C \) of the configuration \( n \). We have characterized optimal such partitions and proposed a simple and fast method for constructing good partitions \( C = \Psi_1 \) (cf. Section 2.4).

For the sake of comparison, we also consider the trivial partition \( C = \Psi_1 \). Relative to Algorithm E, Algorithms F and H represent successive improvements in efficiency for non-polymorphic loci. In this section, we provide an empirical analysis of these algorithms and demonstrate that our optimizations yield a substantial speedup.

The optimized algorithms, along with their asymptotic time complexities, are summarized in Table 1. Intuitively, for a fixed number of haplotypes \( n \), and assuming coarse homogeneity across the genome, the runtimes of each of these algorithms should be linear in the number of loci. We are interested in determining the constants associated with this linear behaviour for each algorithm. Note, however, that for the cases when \( C = \Psi_1 \), the time complexities do not depend on \( n \) directly, but rather the number of unique haplotypes \( n_0 \). For a particular value of \( n \), the quantity \( n_0 \) will increase with the number of loci under consideration until \( n_0 \approx n \); only at this point do the runtimes become linear in the number of loci. A similar argument can be made for a more general configuration partition \( C \).

In order to attain and analyse the linear behaviour for the modestly sized configurations that are considered, we formally interpret even non-unique haplotypes to be unique, thereby forcing \( n_0 = n \).

We produce data using coalescent simulation: we assume a symmetric 2-allele model and with population-scaled mutation rate \( \theta = 0.005 \) per locus and population-scaled recombination rate \( \rho = 0.001 \) between each pair of adjacent loci. For each of several values of \( n \), we thus simulate an \( n \)-haplotype \( 2 \times 10^5 \) locus configuration \( n \). We compute the partitions \( C = \Psi_1 \) and \( \Psi_1 \) and subsequently record the running time of each algorithm in computing \( \hat{\pi}_{\text{SMC}}(a|n) \), for an arbitrary haplotype \( a \in \Psi_1 \). Throughout, we use a time discretization consisting of \( m = 16 \) intervals. The running times are plotted, on a logarithmic scale, as a function of \( n \) in Figure 4.

### Table 1. A summary of the proposed algorithms along with their asymptotic time complexities

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>( \text{Algorithm}_E )</th>
<th>( \text{Algorithm}_F )</th>
<th>( \text{Algorithm}_G )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C = \Psi_1 )</td>
<td>( O(km(n + n_0)) )</td>
<td>( O(km^2 + m(\Psi_1(C) + \Omega_1(C))) )</td>
<td>( O(km^2 + m(\Psi_1(C) + \Omega_1(C))) )</td>
</tr>
<tr>
<td>( C = \Psi_2 )</td>
<td>( O(km(n + n_0)) )</td>
<td>( O(km^2 + m(\Psi_1(C) + \Omega_1(C))) )</td>
<td>( O(km^2 + m(\Psi_1(C) + \Omega_1(C))) )</td>
</tr>
</tbody>
</table>

### Fig. 3. Empirically observed running time of Algorithm E used to compute \( \hat{\pi}_{\text{SMC}}(a|n) \) for a particular configuration \( n \) and an arbitrary \( a \in \Psi_1 \). Several values of \( s \in \{2, \ldots, 500 \} \) were used, and each circle corresponds to a particular value of \( s \). The curve of circles demonstrates the trade-off between small \( \Psi_1/\Psi_1 \) values and large \( \Omega_1/\Omega_1 \) values, and a linear model indicates the constant associated with \( \Psi_1/\Psi_1 \) is \( \approx 1.5 \times \) that associated with \( \Omega_1/\Omega_1 \). As predicted by the asymptotic time complexity results, running time appears to depend linearly on both \( \Psi_1/\Psi_1 \) and \( \Omega_1/\Omega_1 \), and fitting a linear model indicates the constant associated with \( \Psi_1/\Psi_1 \) is \( \approx 1.5 \times \) that associated with \( \Omega_1/\Omega_1 \). The configuration \( \Omega_1/\Omega_1 \) was generated using coalescent simulation for 500 individuals, each having \( 10^5 \) bi-allelic loci, using population-scaled mutation rate \( \theta = 0.005 \) per locus and population-scaled recombination rate \( \rho = 0.001 \) between each pair of adjacent loci, and resulting in \( \Psi_1 = 1724 \) polymorphic loci and \( n_0 = 324 \) unique haplotypes.

Table 1. A summary of the proposed algorithms along with their asymptotic time complexities

Recall that Algorithm H with \( C = \Psi_1 \) is equivalent to Algorithm E.
Algorithm 1. Observe that Algorithm 3, even for larger values of n.

Algorithm 4.

Fig. 4. Log-scaled plots of the running time (in milliseconds) required to compute $\hat{\pi}_{\text{SMC}}(n|\alpha)$ for n with $2 \times 10^5$ loci and $|\alpha| = n$, as a function n, for each of Algorithms A. Configurations were generated using coalescent simulation as described in the text and results obtained on a single core of a MacPro with dual quad-core 3.0GHz Xeon CPUs. (a) $C = C_T$, the trivial configuration partition. (b) $C = C^*$, the configuration partition described in Section 4.

Table 2. A summary of several key statistics from Figure 4.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$n = 100$</th>
<th>$n = 2000$</th>
<th>$n = 5000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm A</td>
<td>45 (1.0x)</td>
<td>870 (1.0x)</td>
<td>2153 (1.0x)</td>
</tr>
<tr>
<td>Algorithm B</td>
<td>3.5 (13x)</td>
<td>21 (44x)</td>
<td>54 (40x)</td>
</tr>
<tr>
<td>Algorithm C</td>
<td>0.63 (71x)</td>
<td>18 (48x)</td>
<td>49 (44x)</td>
</tr>
<tr>
<td>Algorithm D</td>
<td>0.14 (320x)</td>
<td>0.68 (1300x)</td>
<td>1.17 (1845x)</td>
</tr>
</tbody>
</table>

The table indicates the time (in seconds) required to compute the CSP $\hat{\pi}_{\text{SMC}}(n|\alpha)$ for $|\alpha| = n$, per $1 \times 10^5$ loci. The speed-up versus Algorithm A with $C = C_T$, equivalent to the commonly used Algorithm B, is given in parentheses.

4 DISCUSSION

We have presented a number of optimized algorithms for computing the CSP $\hat{\pi}_{\text{SMC}}(n|\alpha)$. Our optimizations are based on two intuitive observations: first, the number of unique haplotypes in a genomic sample is dramatically reduced within relatively short regions and second, the large number of non-polymorphic loci in a genomic sample, though informative, do not distinguish between haplotypes. These observations are formalized and leveraged to refine the recursive equations for computing $\hat{\pi}_{\text{SMC}}(n|\alpha)$, yielding optimized, yet exact, algorithms.

We have described our optimization algorithms in the context of the HMM associated with the CSD $\hat{\pi}_{\text{SMC}}$ proposed by Paul et al. (2011). It is natural to question whether similar optimizations are applicable to related CSDs, such as those proposed by Fearnhead and Donnelly (2001) and Li and Stephens (2003). In Section 4, we described two sufficient conditions: Property A, which stipulates that, upon recombination, a new hidden haplotype is chosen independently and uniformly at random and Property B, which stipulates that the emission distribution depends only on the allele at the current locus of the hidden haplotype. The aforementioned CSDs do satisfy both of these properties; in particular, stronger forms of Property B hold for both CSDs, enabling additional optimizations. We have not empirically analysed the resulting optimized algorithms, but by considering the resulting asymptotic time complexities, analogous to those in Table 2, we anticipate that the speedups obtained will be qualitatively comparable to those observed for $\hat{\pi}_{\text{SMC}}$, though the corresponding magnitudes are difficult to estimate.

It is also interesting to consider CSDs for more complex demographic scenarios. A theoretically straightforward extension of $\hat{\pi}_{\text{SMC}}$ to variable population size, for example, will continue to satisfy both Properties A and B and will therefore be amenable to very similar optimizations. On the other hand, extension to structured populations, populations that are divided into several demes between which there is limited migration, will not satisfy Property B as the new hidden haplotype chosen upon recombination depends on
Blockwise HMM computation

the deme in which recombination occurs. Nonetheless, a relaxed version of Property \( P \) will be satisfied along with Property \( H \) and we anticipate analogous optimizations will be possible. The outcome is similar if \( P_{	ext{aux}} \) is extended to conditionally sampling diploid, rather than haploid, individuals.

Related optimizations may be possible in other contexts as well. For example, Li and Durbin (2011) make use of a population genetic HMM which satisfies conditions that correspond to Properties \( H \) and \( H \) and is used to analyse genomic data. It is interesting to note that, in order to make their method practicable, Li and Durbin consider non-overlapping 100 bp windows as their set of loci; using the optimization detailed in this article may render such compromises unnecessary. It is less clear whether our optimizations are applicable to other population genetic HMMs, such as those considered by Hobolth et al. (2008), and Dutheil et al. (2009), nonetheless, we hope that our work will foster progress in this area.

We conclude by recalling that a broad range of population genetic methods have been developed and will continue to be developed, based on the CSD. These methods are generally computationally intensive, and approximations are often made on the basis of efficiency and at the expense of accuracy; with the advent of inexpensive genomic sequencing, such computational problems will be compounded. We have introduced several optimizations for CSD computation that can potentially speed up this computation by several orders of magnitude without introducing additional approximations. We believe that these optimizations will enable analyses that were previously impracticable, particularly for large genomic datasets. We also hope that the optimizations will encourage more accurate methods, and in particular more accurate CSDs, to be developed and used for population genomic inference.

Acknowledgements

We thank Anand Bhaskar and Matthias Steinrücken for fruitful discussion on both the theoretical and practical aspects of this work.

Funding: NIH National Research Service Award Trainee appointment (T32-HG00047) to JSP and an NSF CAREER Grant (DBI 0846015) and a Packard Fellowship for Science and Engineering to YSS.

Conflict of Interest: none declared.

REFERENCES


