Counting All Possible Ancestral Configurations of Sample Sequences in Population Genetics

Yun S. Song, Rune Lyngsø, and Jotun Hein

Abstract— Given a set D of input sequences, a genealogy for D can be constructed backwards in time, using such evolutionary events as mutation, coalescent and recombination. An ancestral configuration (AC) can be regarded as the multiset of all sequences present at a particular point in time in a possible genealogy for D. The complexity of computing the likelihood of observing D depends heavily on the total number of distinct ACs of D, and therefore it is of interest to estimate that number. For D consisting of binary sequences of finite length, we consider the problem of enumerating exactly all distinct ACs. We assume that the root sequence type is known and that mutation process is governed by the infinite-sites model. When there is no recombination, we construct a general method of obtaining closed-form formulas for the total number of ACs. The enumeration problem becomes much more complicated when recombination is involved. In that case, we devise a method of enumeration based on counting contingency tables and construct a dynamic programming algorithm for the approach. Lastly, we describe a method of counting the number of ACs that can appear in genealogies with less than or equal to a given number R of recombinations. Of particular interest is the case in which R is close to the minimum number of recombinations for D.

Index Terms—Ancestral configurations, coalescent, recombination, contingency table, enumeration

1 Introduction

ONE of the standard problems in mathematical population genetics is to compute the likelihood $P(D)$ of observing a given data set D under the assumed model of evolution. The likelihood $P(D)$ can be defined as a formal sum over all possible genealogies consistent with D. A genealogy, in turn, can be viewed as a sequence of Markov states, and the likelihood $P(D)$ can be determined, in principle, by summing products of transition probabilities over all possible sequences of states in a Markov chain with given initial and final states. There are recursion relations for computing $P(D)$ exactly[1], [2], [15], but that approach quickly becomes infeasible as data size grows, and one must then resort to Monte Carlo methods [3], [5], [6], [7], [11], [12], [16]. Given a set D of input sequences, a genealogy for D can be constructed backwards in time, using such evolutionary events as mutation, coalescent and recombination. An ancestral configuration (AC) can be regarded as the multiset of all sequences present at a particular point in time in a possible genealogy for D. For both deterministic and stochastic approaches, the efficiency of a method of computing $P(D)$ depends heavily on the total number of distinct ACs of D.

In this paper, we address the problem of enumerating all *distinct* ACs for a given data set D, consisting of binary sequences of finite length. That is, we are interested in computing the total number of distinct ACs in all possible genealogies that could have generated D. If an AC appears in two or more different genealogies, it is counted only once. We consider both the classic coalescent [9], [10], in which recombination is absent, and its extension where recombination is allowed. When a mutation event occurs in a genealogy, then it always leads to a new AC in the genealogy, but a coalescent or a recombination event may lead to an AC that has already been encountered in the genealogy.

In the absence of recombination, genealogies can be represented by time-ordered binary trees, whereas a case with recombination requires a more complicated graphical representation called the ancestral recombination graph (ARG) [5]. It is well known that the coalescent with recombination is considerably more difficult to study than the classic coalescent. One obvious reason for that contrast is that there are many more inequivalent ARGs than trees. As many computations of interest—computing $P(D)$, for example—involve studying a set of genealogies consistent with the observed data, perhaps it is not so surprising that including recombination in the model of evolution poses many challenges. One of our goals here is to illustrate more precisely why recombination is difficult to study, by comparing the total number of ACs when there is recombination with that when it is absent. It has been known to many, if not most people in population genetics that there can be many more ACs when there is recombination than when it is absent, but we are not aware of any work that tried to examine exactly by how much.

As mentioned above, when there are many ACs, solving recursion relations to compute $P(D)$ exactly is often infeasible. In such a case, we are interested in asking whether it would be possible to approximate the true recursion by a "collapsed" recursion, in which ACs are lumped together, such that we just need to consider transitions, with appropriately defined probability, between lumped ACs to estimate $P(D)$. We show in this paper that this idea of lumping ACs works at least in the context of our enumeration problem. What we achieve here can be viewed as a small step towards meeting our desired goal.

In our work, we assume the infinite sites model of mutation, which means that there can be at most one mutation per site. Hence, the input data can be regarded as defining a

Y.S. is with the Department of Computer Science, University of California at Davis, Davis, CA 95616, USA. E-mail: yssong@cs.ucdavis.edu

R.L. and J.H. are with the Department of Statistics, University of Oxford, 1 South Parks Road, Oxford, OX1 3TG, UK. E-mail: lyngsoe, hein@stats.ox.ac.uk

set of binary sequences. For ease of discussion, we assume that the root sequence is known, and use 0 to denote the ancestral type at the root and 1 to denote a mutant type. We remark that the techniques described in this paper can be generalized to the root-unknown case. When there is no recombination, we construct a general method of obtaining closed-form formulas for the total number of ACs; these formulas are polynomials in the multiplicity of distinct sequences in D. We have implemented this method in *Mathematica* and the program is called `aceTrees` (short for "ancestral configuration enumeration for trees"). As expected, the enumeration problem becomes much more complicated when recombination is involved. In that case, we discuss a method based on counting non-negative integer valued tables with fixed row and column sums, commonly known as contingency tables. For two sites, we show how a closed-form formula can be obtained. For an arbitrary number of sites, we construct a dynamic programming algorithm for counting ACs. We have implemented this algorithm in *C++*; the software is called `aceARGs` (short for "ancestral configuration enumeration for ARGs"). Lastly, we discuss a method of counting the number of ACs that can appear in ARGs with less than or equal to a given number R of recombinations. Of particular interest is the case in which R is (or is close to) the minimum number of recombinations for D. We have implemented this algorithm in *C*; the software is called `greven`. This method interpolates between the case where there is no recombination and the case where an arbitrary number of recombinations are allowed. As it should, `greven` agrees with `aceTrees` and `aceARGs` in those two respective limits. As a further check, we have made alternative, independent implementations of all three programs in *Python*. All our programs implementing the methods discussed in this paper are available upon request.

The organization of this paper is as follows. In Section 2, we consider the problem of enumerating ACs when there is no recombination, in which case genealogies can be represented by coalescent trees. Mitochondria data from [17] are considered there as an example. By counting the total number of ACs, we illustrate why an exact computation of the likelihood is difficult for that data. The aforementioned method of counting ACs in unconstrained recombination graphs is discussed in Section 3. We compare the number of ACs in coalescent trees with that in ARGs and highlight their differences. In Section 4, we consider enumerating ACs that can appear in ARGs with at most R recombinations. We conclude in Section 5 with some general remarks on our work.

2 Ancestral Configurations in Coalescent Trees

In this section, we focus on the case in which the evolution under the infinite-sites model can be represented by a tree. A given data set D is compatible with a tree if it passes the three gamete test: for every pair of sites, not all of the allele types 01, 10 and 11 appear in the data. If the three gamete test is passed, then there exists a unique perfect phylogeny τ for D [4], [8], with at most one mutation per site, but it is important to note that in general there are many coalescent trees consistent with τ. (Recall that, unlike coalescent trees, a perfect phylogeny may be non-binary and that relative time ordering of two of its interior vertices is not defined if one vertex is not a descendant of the other.)

2.1 Definition of an ancestral configuration

In our algorithm for counting ACs, we partition the information contained in a configuration into two parts, one encoding sequence types and the other their multiplicity. We use $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d$ to denote d *distinct* finite binary sequences (allele types) $\boldsymbol{x}_i$ of some fixed length—for example, $\boldsymbol{x}_1 = 0100, \boldsymbol{x}_2 = 0010$, and $\boldsymbol{x}_3 = 1100$, with d being 3. We define $T = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d)$ and $\boldsymbol{n} = (n_1, \ldots n_d)$, where $n_i > 0$, for all $i \in \{1, \ldots, d\}$, denote their multiplicity. We say that $(T, \boldsymbol{n})$ and $(T', \boldsymbol{n}')$ are equivalent, denoted $(T, \boldsymbol{n}) \sim (T', \boldsymbol{n}')$, if there exists a permutation $\sigma \in S_d$ such that $(T_\sigma, \boldsymbol{n}_\sigma) = (T', \boldsymbol{n}')$, where $T_\sigma = (\boldsymbol{x}_{\sigma(1)}, \ldots, \boldsymbol{x}_{\sigma(d)})$ and $\boldsymbol{n}_\sigma = (n_{\sigma(1)}, \ldots, n_{\sigma(d)})$. By a *configuration*, we mean an equivalence class $[T, \boldsymbol{n}] \in \{(T, \boldsymbol{n})\}/\sim$.

As we describe presently, every coalescent tree defines a sequence of configurations. Consider a data set $D = [T, \boldsymbol{n}]$, where $T = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d)$ and $\boldsymbol{n} = (n_1, \ldots, n_d)$. A configuration is said to be *ancestral* to D if it "appears" in at least one possible coalescent tree that derives D. (Note that D is ancestral to itself according to this definition.) Let us illustrate this definition through a specific example. Consider the simple example $D = [(1, 0), (2, 2)]$, consisting of four binary sequences of length one. There are several coalescent trees that can give rise to D under the infinite-sites model of mutation. Three of them are shown in Figure 1. A cross-section, corresponding to a particular time slice, of a coalescent tree defines an AC. Going backwards in time, configuration changes whenever either a mutation or a coalescent event occurs.

For $D = [(1, 0), (2, 2)]$, the seven configurations shown in Figure 1 form a complete set of configurations ancestral to D; i.e. any configuration appearing in a coalescent tree for D must be one of the seven configurations. The corresponding Markov chain transition graph C_D, where vertices correspond to the ACs for D and edges correspond to allowed transitions, is as shown in Figure 2. This graph shows that the sequence of ACs arising in any coalescent tree for D must be one of the following:

$$\begin{aligned}
&\psi_1 \to \psi_2 \to \psi_4 \to \psi_6 \to \psi_7,\\
&\psi_1 \to \psi_2 \to \psi_5 \to \psi_6 \to \psi_7,\\
&\psi_1 \to \psi_3 \to \psi_5 \to \psi_6 \to \psi_7.
\end{aligned}$$

In general, every coalescent tree compatible with any given data set D starts in the configuration corresponding to D and ends in the configuration consisting of a single all-zero sequence (corresponding to the most recent common ancestor).

Let $P_{i,j}$ denote some appropriately-defined transition probability corresponding to the transition $\psi_i \to \psi_j$. Then the probability of the sequence $\psi_{i_1} \to \psi_{i_2} \to \ldots \to \psi_{i_k}$ can be defined as $P_{i_1,i_2} P_{i_2,i_3} \cdots P_{i_{k-1},i_k}$. In [1], Ethier and Griffiths showed that the probability of observing D can be obtained

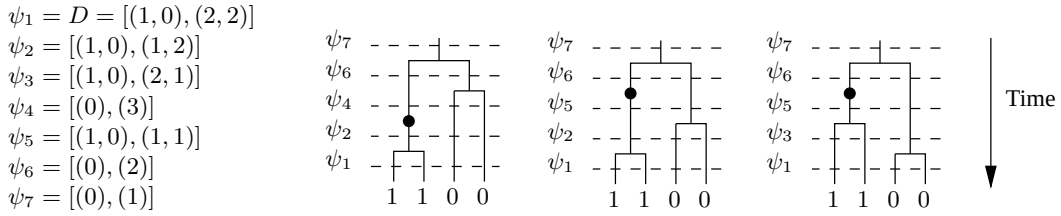

Fig. 1. Coalescent trees and ancestral configurations $\psi_1, \ldots, \psi_7$. A cross-section, corresponding to a particular time slice, of a tree defines an AC.

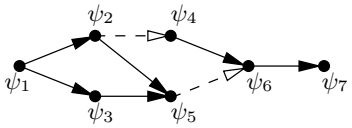

Fig. 2. Markov chain transition graph C_D for $D = [(1,0),(2,2)]$. Dashed lines denote mutation events, while solid lines denote coalescent events. This directed graph is acyclic and contains 3 distinct paths from the initial configuration $\psi_1 = D$ to the absorbing configuration ψ_7.

by summing such probabilities over all possible sequences of ACs. In the example $D = [(1,0),(2,2)]$ considered above,

$$P(D) = P_{1,2}\,P_{2,4}\,P_{4,6}\,P_{6,7} + P_{1,2}\,P_{2,5}\,P_{5,6}\,P_{6,7} + P_{1,3}\,P_{3,5}\,P_{5,6}\,P_{6,7}.$$

Ethier and Griffiths[1] formulated a recursion for evaluating $P(D)$. In general, computing $P(D)$ *exactly* using the recursion becomes infeasible when sample size is large, and one must then resort to Monte Carlo methods (see [6], [7]). In what follows, we develop a method of counting exactly the total number of inequivalent ACs of an arbitrary data set D. This kind of enumeration should prove useful for studying when exact computation of $P(D)$ becomes infeasible.

2.2 Enumeration of ACs: A general solution

We now describe a general, efficient method of obtaining closed-form formulas for the total number of inequivalent ACs. The actual ACs themselves can easily be extracted from our method.

2.2.1. Notation

For ease of discussion, we first introduce some useful notation. We use $\mathbf{1}_k$ to denote a k-tuple of 1s, and $\boldsymbol{e}_i$ to denote a vector with a 1 at the ith entry and $0s$ elsewhere; the length of $\boldsymbol{e}_i$ will be clear from the context of its usage. For $\boldsymbol{n} = (n_1, \ldots, n_k)$, define $\pi(\boldsymbol{n}) := \prod_{i=1}^{k} n_i$. Let $\mathscr{D}_i$ denote a deletion operator which, when acting on a vector of length $k \geq i$, deletes the ith entry of the vector, thus changing its length to $k-1$. Let $\mathscr{R}_{i;z}$ denote a replacement operator which replaces the ith entry of a vector with z.

Let $\mathcal{T} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k)$ and $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_k)$. As usual, $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k$ are k distinct binary sequences. In $\mathcal{T}$, a site is called a *singlet* if there is exactly one allele type $\boldsymbol{y}_i \in \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k\}$ with value 1 at that site. Here, note that μ_i need not be 1. In $\mathcal{T} = (100, 101, 010)$, for example, sites two and three are singlets. Let a be a singlet site and $\boldsymbol{y}_i$ the unique allele type with a 1 at that site. With $\boldsymbol{y}_i^a$ denoting the allele type obtained from $\boldsymbol{y}_i$ after the 1 at site a mutates to a 0, define $\mathscr{M}_a(\mathcal{T}, \boldsymbol{\mu})$ as

$$\begin{cases} (\mathscr{R}_{i;\boldsymbol{y}_i^a}(\mathcal{T}), \mathscr{R}_{i;1}(\boldsymbol{\mu})), & \text{if } \boldsymbol{y}_i^a \notin \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k\}, \\ (\mathscr{D}_i(\mathcal{T}), \mathscr{D}_i(\boldsymbol{\mu} + \boldsymbol{e}_j)), & \text{if } \boldsymbol{y}_i^a = \boldsymbol{y}_j \in \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k\}. \end{cases}$$

Since a singlet site a uniquely determines i, we omit i in writing $\mathscr{M}_a(\mathcal{T}, \boldsymbol{\mu})$. In the example $\mathcal{T} = (100, 101, 010)$ considered above, the unique allele type associated with $a = 2$ is $\boldsymbol{y}_3$, and $\boldsymbol{y}_3^2 = (000)$ is not in $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_3\}$. For $a = 3$, the associated unique allele type is $\boldsymbol{y}_2$, and $\boldsymbol{y}_2^3 = (100)$, which is equal to $\boldsymbol{y}_1$. Hence, $\mathscr{M}_2(\mathcal{T}, \boldsymbol{\mu}) = ((100, 101, 000), (\mu_1, \mu_2, 1))$ and $\mathscr{M}_3(\mathcal{T}, \boldsymbol{\mu}) = ((100, 010), (\mu_1 + 1, \mu_3))$.

2.2.2. Coalescent and mutation events

Consider a data set $D = [T, \boldsymbol{n}]$. The multiplicity n_i of an allele type $\boldsymbol{x}_i$ decreases by exactly one when two sequences of that allele type coalesce. The number of ACs with allele types T is therefore equal to $\pi(\boldsymbol{n})$, which is equal to 1 (for $[T, \boldsymbol{n}]$ itself) plus the number of configurations that can be reached from $[T, \boldsymbol{n}]$ via coalescent events only. When every allele type has coalesced completely, we end up with the configuration $[T, \mathbf{1}_d]$.

In $T = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d)$, suppose that $\boldsymbol{x}_i$ is the only allele type with a 1 at site a. Consider a configuration $[T, \boldsymbol{m}]$ with $m_i = 1$, reached by some coalescent events starting from $[T, \boldsymbol{n}]$. Now, a mutation event can occur to change the character of $\boldsymbol{x}_i$ at site a from 1 to 0, making $[T, \boldsymbol{m}]$ jump to a different configuration $[T', \boldsymbol{m}']$, where $T' \neq T$. Let $\boldsymbol{x}_i^a$ denote the allele type obtained after such a mutation event. There are two possible cases. First, if $\boldsymbol{x}_i^a$ is equal to one of the alleles in T, say $\boldsymbol{x}_j$, then the configuration after the mutation event is $[\mathscr{D}_i(T), \mathscr{D}_i(\boldsymbol{m} + \boldsymbol{e}_j)]$. Note that the multiplicity of $\boldsymbol{x}_j$ has increased by one. How many configurations are there with allele types $\mathscr{D}_i(T)$? For $k \neq i$ and $k \neq j$, the multiplicity of $\boldsymbol{x}_k$ can be any integer between n_k and 1; independently of the mutation event, coalescent events within allele type $\boldsymbol{x}_k$ can reduce the multiplicity from n_k to 1. Further, since the multiplicity of $\boldsymbol{x}_j$ increases by 1 after the mutation event, it can range from $n_j + 1$ to 1. Hence, the total number of possible ACs with allele types $\mathscr{D}_i(T)$ is $\pi(\mathscr{D}_i(\boldsymbol{n} + \boldsymbol{e}_j))$. Second, if $\boldsymbol{x}_i^a$ is not equal to any allele in T, then the configuration after the mutation event is $[\mathscr{R}_{i;\boldsymbol{x}_i^a}(T), \boldsymbol{m}]$. Note that we still have $m_i = 1$. How many configurations are there with allele types $\mathscr{R}_{i;\boldsymbol{x}_i^a}(T)$? Similar to the first case, since $\boldsymbol{m}$ is bounded by $\mathscr{R}_{i;1}(\boldsymbol{n})$ and $\mathbf{1}_d$, the number of configurations with allele types $\mathscr{R}_{i;\boldsymbol{x}_i^a}(T)$ is $\pi(\mathscr{R}_{i;1}(\boldsymbol{n}))$.

2.2.3. A graph construction algorithm

Our method of counting ACs is to deal with coalescent events for each allele type configuration $\mathcal{T}$ combinatorially and find all possible allele type configurations using a simple graph construction algorithm. More precisely, the general idea goes as follows. For a given input data set D, suppose that there are ACs with $\mathcal{T}$ as allele type configuration. If the maximum multiplicity of $\mathcal{T}$ over all such ACs is $\boldsymbol{\mu}$, then, as discussed above, a simple coalescent argument shows that the total number of ACs with allele type configuration $\mathcal{T}$ is $\pi(\boldsymbol{\mu})$. Hence, if we know all possible allele type configurations $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_p$ and their maximum multiplicity $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_p$, respectively, then we can easily determine the total number of ACs of D by summing over $\pi(\boldsymbol{\mu}_j)$. Our graph construction algorithm below finds such $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_p$ and $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_p$.

We wish to construct a directed graph G_D where a vertex v_j is labeled by $(T_j, \boldsymbol{n}_j)$, to be determined by the following iterative procedure:

1. Given a data set $D = [T, \boldsymbol{n}]$, let G_D initially be a graph with no edges and with $v_1 = (T, \boldsymbol{n})$ as its only vertex. Here, $(T, \boldsymbol{n})$ is an arbitrary representative of the equivalence class $[T, \boldsymbol{n}]$.
2. Let V_0 denote the set of all vertices in G_D with out-degree zero. For all $v_j = (T_j, \boldsymbol{n}_j) \in V_0$, determine the set S_{v_j} of singlet sites (defined above) in T_j.
3. If $S_{v_j} = \varnothing$ for all $v_j = (T_j, \boldsymbol{n}_j) \in V_0$, terminate the procedure. Otherwise, arbitrarily order V_0 and sequentially carry out the following steps for v_j satisfying $S_{v_j} \neq \varnothing$: Determine $\mathscr{M}_a(T_j, \boldsymbol{n}_j)$ for all $a \in S_{v_j}$. If $\mathscr{M}_a(T_j, \boldsymbol{n}_j) = v_k \in G_D$, draw a directed edge from v_j to v_k. If not, then add $v_k = \mathscr{M}_a(T_j, \boldsymbol{n}_j)$ to G_D and draw a directed edge from v_j to v_k.
4. Go back to step 2.

The graph G_D compactly encodes all possible ACs. In a vertex $v_j = (T_j, \boldsymbol{n}_j) \in G_D$, T_j captures the set of distinct binary strings, whereas $\boldsymbol{n}_j$ determines the range of possible multiplicity of the strings; i.e. if there are r distinct binary sequences in T_j, then the multiplicity can be anything between $\boldsymbol{n}_j$ and $\mathbf{1}_r$. Further, it is straightforward to show that if v_j and v_k are two distinct vertices in G_D, then $T_j \neq T_k$. The total number $\alpha(D)$ of configurations ancestral to D is therefore given by

$$\alpha(D) = \sum_{i \in V(G_D)} \pi(\boldsymbol{n}_i), \tag{2.1}$$

where $V(G_D)$ is the index set of the vertices in G_D. We have implemented the above algorithm in *Mathematica*. For a given data set $D = [(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d), (n_1, \ldots, n_d)]$, our program generates closed-form formulas for $\alpha(D)$ in terms of $n_1, \ldots, n_d$.

(REMARK: We have an independent program, to be discussed later, that can exhaustively search through evolutionary histories to compute the number of ACs. That program can be used to analyze small sample sizes (usually less than 30 sequences). We have checked that it produces the same answers as does our combinatorial method described above.)

$$\begin{aligned}
v_1 &= ((100, 110, 001), (n_1, n_2, n_3)) \\
v_2 &= ((100, 001), (n_1 + 1, n_3)) \\
v_3 &= ((100, 110, 000), (n_1, n_2, 1)) \\
v_4 &= ((100, 000), (n_1 + 1, 1)) \\
v_5 &= ((000, 001), (1, n_3)) \\
v_6 &= ((000), (2))
\end{aligned}$$

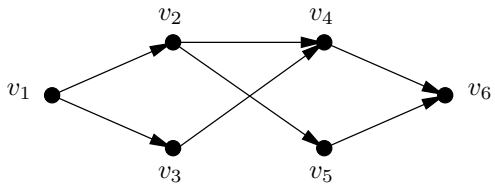

Fig. 3. Application of our enumeration algorithm to $D = [(100, 110, 001), (n_1, n_2, n_3)]$. Shown on the right hand side is the final graph G_D constructed by our algorithm.

TABLE 1

Mitochondria data from [17], consisting of 63 sequences and 18 segregating sites. There are $413,243,616$ ACs for this data set.

Haplotype	Multiplicity
100101000000010000	2
100101000100010000	2
010000000000000001	1
001000001000000000	3
000101000000010000	19
000111000000010000	1
000000000011000001	1
000000000011100001	1
000000000000000111	4
000000000000000101	8
000000000000000000	5
000000000000000001	4
000000010000000000	3
000000100000001000	1

2.3 A toy example

Consider the data set $D = [(100, 110, 001), (n_1, n_2, n_3)]$, where n_1, n_2, and n_3 are some arbitrary positive integers. Applying the algorithm from Section 2.2 leads to the graph G_D illustrated in Figure 3. From (2.1), the total number of configurations ancestral to D is thus given by

$$\begin{aligned}
\alpha(D) &= \sum_{i=1}^{6} \pi(\boldsymbol{n}_i) \\
&= n_1 n_2 n_3 + n_3(n_1 + 1) + n_1 n_2 + (n_1 + 1) + n_3 + 2 \\
&= n_1(n_2 + 1)(n_3 + 1) + 2n_3 + 3.
\end{aligned}$$

2.4 Mitochondria DNA data

We now consider the data set from Ward *et al.*[17], which consists of sequences from the control region of mitochondria DNA (mtDNA), sampled from 63 individuals in a single Amerindian tribe. There are 18 segregating sites in the data, shown in Table 1.

Computing the likelihood of observing a given data set is important—for example, for parameter estimation—but, as mentioned in Section 2.1, doing it exactly is infeasible when sample size is large. The same mtDNA data set was considered

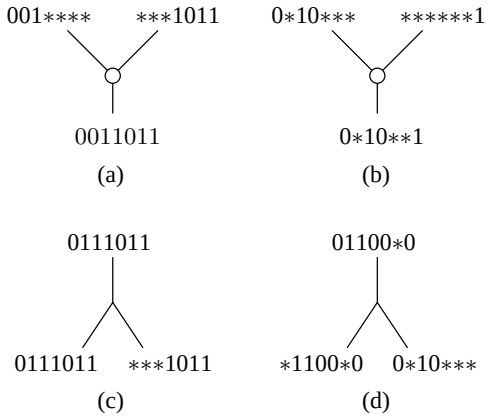

Fig. 4. Illustration of possible events. Time flows from top to bottom. (a) and (b) are examples of recombination events. Let $s[i:j]$ denote the substring of s in-between i and j, inclusive. Going backwards in time, when a recombination event is encountered, the lineage of a sequence $s[1:L]$ breaks up into two parts, one corresponding to the lineage of a sequence s_1 whose prefix $s_1[1:k]$ is identical to $s[1:k]$ and the other corresponding to the lineage of a second sequence s_2 whose suffix $s_2[k+1:L]$ is identical to $s[k+1:L]$. The suffix $s_1[k+1:L]$ and the prefix $s_2[1:k]$ carry non-ancestral material, denoted by "$*$"s. (c) and (d) are examples of coalescent events. Two sequences s_1 and s_2 can coalesce if their characters are identical at common ancestral positions. The final sequence contains the union of the ancestral material in s_1 and s_2.

by Griffiths and Tavaré [6], who developed a Markov chain Monte Carlo method to analyze the data. Our goal here is to compute the total number of inequivalent ACs of the data, thus illustrating why computing the likelihood exactly using recursions is difficult.

Let $T = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{14})$ denote the 14 distinct haplotypes shown in Table 1, with $\boldsymbol{x}_i$ being the ith row. Applying our algorithm from Section 2.2 to $D = [T, (n_1, \ldots, n_{14})]$ generates a directed graph G_D with $12,896$ vertices, and one can easily obtain a closed-form formula, which is too long to write down here, for the total number $\alpha(D)$ of ACs. For $\boldsymbol{n} = (1, 1, \ldots, 1)$, $\alpha(D) = 128,640$. For $\boldsymbol{n} = (2, 2, 1, 3, 19, 1, 1, 1, 4, 8, 5, 4, 3, 1)$, which being the multiplicity of the actual data, $\alpha(D) = 413,243,616$.

3 Ancestral Configurations in Unconstrained ARGs

In this section, we turn to enumerating ancestral configurations in unconstrained ancestral recombination graphs (ARGs)[5]. This case is much more complicated than the classic coalescent case, and, in general, it is difficult to obtain closed-form formulas for the number of ACs. Below we translate the problem of counting ACs into counting contingency tables and provide a dynamic programming algorithm.

3.1 Definition of an AC in ARGs

In the absence of recombination, if the input data D contains n sequences, then any configuration ancestral to D contains at most n sequences. This is no longer true when recombinations are allowed. Going backwards in time, when a recombination event is encountered, the lineage of a sequence breaks up into two parts, distributing its ancestral material to two different sequences which carry additional non-ancestral material (denoted by "$*$"s). This concept is illustrated in Figure 4, where possible coalescent events are also described. To simplify things, we assume that recombination breakpoints occur at the midpoints of consecutive sites in D. Hence, if D consists of some segregating sites in a region X, the distribution of ancestral material between two consecutive segregating sites i and $i+1$ in X is completely determined by the configuration at sites i and $i+1$.

Of particular interest is a class of ARGs in which, for any recombination event, both the prefix and the suffix involved in a recombination event contain some ancestral material. Such a class of ARGs was the main focus of [5], and we also limit our attention to such ARGs in this paper. Hence, for our purpose, a sequence appearing in an ARG is generally a string over $\{0, 1, *\}$, but the all-$*$ string is not allowed. As before, 0 denotes the ancestral type at the root and 1 denotes a mutant type. A configuration is defined as in Section 2.1, except that now strings are defined over $\{0, 1, *\}$. To every ARG, there corresponds a sequence of configurations, which can be viewed as generating the ARG backwards in time. Given a data set D, a configuration is said to be *ancestral* to D if it appears in at least one possible sequence of configurations that corresponds to some ARG consistent with D. Shown in Figure 5 is a sequence of ACs and its corresponding ARG. It is important to note that there are many—in fact, infinitely many—distinct ARGs that can derive the same initial data, and that the total number of ACs can be immensely large. This is the main point that we wish to illustrate in this paper. For the simple example $D = [(10, 11, 01), (1, 1, 2)]$ considered in Figure 5, there are only 220 ACs in total. We shall soon see that, as the number of sites and the number of sequences increase, the total number of ACs grows extremely fast.

3.2 A warm-up example

Before we plunge into the core of our enumeration work, let us consider a simple example for which it is not too difficult to obtain a complete set of ACs by hand. The reader not too familiar with the ARG is recommended to go through this example. Consider the initial data $D = [(00, 11), (1, 1)]$. If recombinations are not allowed, then it is clear that there are only 5 ACs, namely $[(00, 11), (1, 1)]$, $[(00, 01), (1, 1)]$, $[(00, 10), (1, 1)]$, $[(00), (2)]$, and $[(00), (1)]$. In the presence of recombination, there are 30 ACs, shown in Figure 6. These ACs can be generated iteratively by asking what happens to a given AC under a mutation, a coalescent or a recombination event; note that some of these events may not be possible, depending on the AC. On the right hand side of Figure 6 is a Markov chain transition graph C_D, depicting the possible transitions between ACs. Note that the graph C_D contains directed cycles. AC 1 is the initial configuration, and AC 21 is called the grand common ancestor. To any ARG with the grand common ancestor as the root, there corresponds a unique path from AC 1 to AC 21.

Ancestral Configurations

$1:[(10,11,01),(1,1,2)]$	$4:[(10,01),(1,2)]$	$7:[(00),(2)]$
$2:[(10,1*,*1,01),(1,1,1,2)]$	$5:[(00,01),(1,2)]$	$8:[(00),(1)]$
$3:[(10,1*,01),(1,1,2)]$	$6:[(00,01),(1,1)]$	

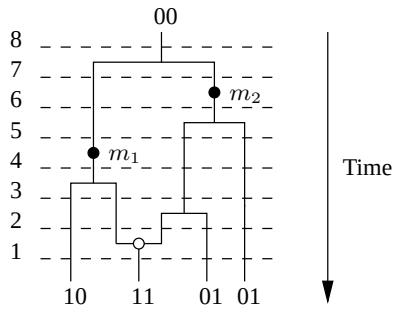

Fig. 5. A sequence of ACs for $D = [(10, 11, 01), (1, 1, 2)]$ and its corresponding ARG. Filled circles denote mutation events. Open circle denotes a recombination event with breakpoint between the first and the second sites. Mutation events at the first and the second sites are denoted by m_1 and m_2, respectively. Note that there are other ARGs that could have generated D. In total, there are 220 ACs of D.

Ancestral Configurations

$1:[(00,11),(1,1)]$	$11:[(00),(2)]$	$21:[(00),(1)]$
$2:[(0*,*0,11),(1,1,1)]$	$12:[(0*,10,*1),(1,1,1)]$	$22:[(01,10),(1,1)]$
$3:[(00,1*,*1),(1,1,1)]$	$13:[(*0,01,1*),(1,1,1)]$	$23:[(0*,*0),(2,2)]$
$4:[(00,10),(1,1)]$	$14:[(0*,*0,*1),(2,1,1)]$	$24:[(0*,*0,*1),(1,1,1)]$
$5:[(00,01),(1,1)]$	$15:[(0*,*0,1*),(1,2,1)]$	$25:[(0*,*0,1*),(1,1,1)]$
$6:[(0*,*0,1*,*1),(1,1,1,1)]$	$16:[(01,*0),(1,1)]$	$26:[(00,*0),(1,1)]$
$7:[(0*,*0,01),(1,1,1)]$	$17:[(0*,*0,00),(1,1,1)]$	$27:[(00,0*),(1,1)]$
$8:[(0*,*0,10),(1,1,1)]$	$18:[(0*,10),(1,1)]$	$28:[(0*,*0),(1,1)]$
$9:[(00,0*,*1),(1,1,1)]$	$19:[(00,*1),(1,1)]$	$29:[(0*,*0),(2,1)]$
$10:[(00,1*,*0),(1,1,1)]$	$20:[(00,1*),(1,1)]$	$30:[(0*,*0),(1,2)]$

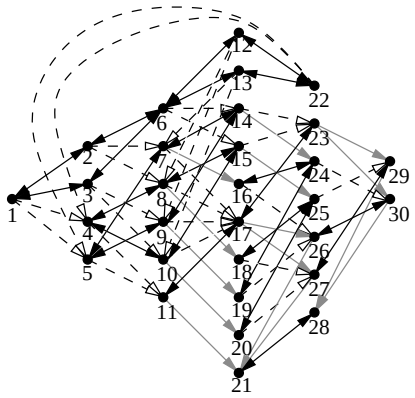

Fig. 6. Configurations ancestral to $D = [(00, 11), (1, 1)]$ and the Markov chain transition graph C_D. Mutation events are denoted by dashed arrows. Coalescent and recombination events are denoted by solid arrows. Note that some solid arrows are bidirectional, one direction for recombination and the other for coalescent. Unidirectional arrows denote coalescent events.

3.3 Possible number of 1s and 0s in an AC

Let n denote the total number of binary sequences in D, with each sequence being of length L. Let n_0^i (resp. n_1^i) denote the total number of 0s (resp. 1s) at site i. Note that $n_0^i + n_1^i = n$ for all $i \in \{1, \ldots, L\}$. (For $D = [(10, 01, 11), (2, 3, 1)]$, $n_0^1 = 3, n_1^1 = 3$ for the first site and $n_0^2 = 2, n_1^2 = 4$ for the second site.) Let r_0^i and r_1^i denote the total number of 0s and of 1s, respectively, at site i in an arbitrary AC ψ. The infinite-sites model of mutation imposes constraints on possible values of (r_0^i, r_1^i). For $n_1^i \neq 0$, a mutation event at site i can occur in AC ψ only if $r_1^i = 1$. If $r_1^i = 1$ and the mutation event occurs, then AC ψ jumps to a new AC ψ' with $(r_0^i + 1)$ 0s and no 1s.

To determine the possible range of r_0^i and r_1^i, let us first consider the $L = 1$ case. Since every sequence is of length one, it follows from the above discussion that there exists *exactly* one AC for any pair $(r_0^1, r_1^1) \in I(n_0^1, n_1^1)$, where $I(n_0, n_1)$ is defined as the set of all non-negative integer pairs (p, q) such that if $n_0 \neq 0$ and $n_1 \neq 0$, then

$$\begin{array}{ll} \text{either} & \text{(i) } 1 \leq p \leq n_0 \text{ and } 1 \leq q \leq n_1, \\ \text{or} & \text{(ii) } 1 \leq p \leq n_0 + 1 \text{ and } q = 0; \end{array} \tag{3.2}$$

if $n_0 = 0$ and $n_1 = n$, then

$$\begin{array}{ll} \text{either} & \text{(i) } p = 0 \text{ and } 1 \leq q \leq n, \\ \text{or} & \text{(ii) } p = 1 \text{ and } q = 0; \end{array} \tag{3.3}$$

and, if $n_0 = n$ and $n_1 = 0$, then

$$1 \leq p \leq n \text{ and } q = 0. \tag{3.4}$$

There are $n_0(n_1 + 1) + 1$ possible values of (p, q) in the first case, $n + 1$ in the second, and n in the last. There exists no AC with r_0^1 0s and r_1^1 1s if $(r_0^1, r_1^1) \notin I(n_0^1, n_1^1)$. Now comes a crucial point. Even for $L > 1$, if an arbitrary number of recombination events are allowed, there exists *at least one* AC for any combination of $(r_0^1, r_1^1), \ldots, (r_0^L, r_1^L)$ satisfying $(r_0^i, r_1^i) \in I(n_0^i, n_1^i)$, for all $i \in \{1, \ldots, L\}$. There exists no AC with r_0^i 0s and r_1^i 1s at site i if $(r_0^i, r_1^i) \notin I(n_0^i, n_1^i)$. One way to see these points is as follows. Starting from the input data D, use recombination events to distribute the ancestral material of the first sequence to L new sequences, each carrying exactly one ancestral site. Do the same for all other sequences in D, so that we end up with $n \cdot L$ sequences, each carrying exactly one ancestral site. Then, determining the range of r_0^i and r_1^i for each site i reduces to the $L = 1$ case. We stress that the above statement would not hold for $L > 1$ in the absence of recombination (or if too few recombination events are allowed). For example, consider $D = [(01), (2)]$. In the absence of recombination, there is no AC with $r_0^1 = 1$ and $r_1^2 = 2$.

3.4 Important observations

We now highlight a couple of important facts. First, the set of ACs does not depend on the order of 0s and 1s within a site when arbitrary number of recombinations are allowed. For instance, $[(01, 10), (1, 1)]$ has the same set of ACs as the example $[(00, 11), (1, 1)]$ considered in Section 3.2 (see

Figure 6). This fact can easily be explained as follows. Let D and D' be two data sets of the same size, such that the ith site of D is equal to the ith site of D' up to some rearrangement of elements within the site. Then, D can be transformed into D', or vice versa, using a series of appropriate recombination and coalescent events. It therefore follows that D and D' have the same set of ACs.

Second, if two input data sets D and D' differ by some permutation σ of their sites, then they have the same *number* of ACs. Any AC of D can be transformed into an AC of D' via σ, and vice versa.

For D a data set containing n sequences of length L, let $X(D) = \{(n_0^1, n_1^1), \ldots, (n_0^L, n_1^L)\}$ denote the multiset of ordered pairs (n_0^i, n_1^i), where n_0^i and n_1^i are the total number of 0s and 1s, respectively, at site i of D. Then, the above two facts together imply that two data sets D and D' have the same number of ACs if $X(D) = X(D')$ as multisets.

3.5 Main idea: Counting restricted contingency tables

We here describe our method of counting ACs. Intuitively, our approach is to construct ACs moving along the sites, say from left to right. Given that we have constructed ACs for the first $k-1$ sites, we construct ACs for the first k sites by appending additional material (i.e., $0, 1$ or $*$) to the right. We iterate this procedure until the last site is processed. As we describe below, this procedure of extending ACs can be translated into constructing contingency tables, where column sums are partially determined by the multiplicity of the strings in the AC that is being extended (i.e., an AC for the first $k-1$ sites) and row sums are partially determined by the allowed numbers of 0s and 1s at site k (c.f., Section 3.3).

3.5.1. Extending an AC

Suppose that ψ is an AC for the first $k-1$ sites. Further suppose that ψ contains d distinct strings, with multiplicity $c_1, c_2, \ldots, c_d$ satisfying $c_1 + c_2 + \cdots + c_d = c$. Since ψ is an AC for the first $k-1$ sites, it does not contain any all-$*$ string of length $k-1$. However, a string s in an AC for the first k sites may contain the all-$*$ string of length $k-1$ as its prefix, if the character at the kth site of s is either 0 or 1, but not $*$. For illustration, consider $D = [(0001, 0000, 1101, 1110), (1, 1, 1, 1)]$. An AC for the first two sites is shown in Figure 7a, where we simply list strings to represent the AC. It has $c = 4$ and the multiplicity of "00" is 2, while every other sequence type has multiplicity 1. In any AC for the first two sites, the string "$**$" is not allowed, but "$**0$" or "$**1$" may appear in an AC for the first three sites. Shown in Figures 7b-d are three examples of ACs for the first three sites; they each contain "$**0$". The all-$*$ string "$***$" is not allowed in any AC for the first three sites.

In an AC for the first k sites, how many 0s and 1s can be present at site k? We have already answered that question in Section 3.3; i.e. the number r_0 of 0s and the number r_1 of 1s satisfy $(r_0, r_1) \in I(n_0^k, n_1^k)$, where n_0^k and n_1^k respectively denote the number of 0s and 1s at site k in the input data D. How about the number r_* of $*$s at site k? Since an all-$*$ string of length-k is not allowed, r_* is bounded from above, while the minimum value of r_* is determined by the total number of strings in ψ and the number of non-$*$ characters at site k. A moment's thought leads to the conclusion that $\max(c - r_0 - r_1, 0) \le r_* \le c$. Similarly, the number c_* of all-$*$ prefixes of length $k-1$ satisfies $\max(r_0 + r_1 - c, 0) \le c_* \le r_0 + r_1$. Note that c_* and r_* must satisfy $c + c_* = r_0 + r_1 + r_*$ for consistency. In Figures 7b-d, the ACs shown have $(r_0, r_1) = (4, 0) \in I(3, 1)$ and $r_* = 1$ at site three, and each AC has exactly one sequence containing "$**$" as a prefix of length 2 (i.e., $c_* = 1$). As required, $c + c_* = r_0 + r_1 + r_*$.

3.5.2. Restricted contingency tables

As above, let ψ be an AC for the first $k-1$ sites containing d distinct strings. The pairing of a length-$(k-1)$ string in ψ or a length-$(k-1)$ all-$*$ string with a character at site k can be concisely summarized by a table of the following form:

				row sums
$A_{0,1}$	$\cdots$	$A_{0,d}$	$A_{0,*}$	r_0
$A_{1,1}$	$\cdots$	$A_{1,d}$	$A_{1,*}$	r_1
$A_{*,1}$	$\cdots$	$A_{*,d}$	0	r_*
c_1	$\cdots$	c_d	c_*	

column sums:

For $j = 1, \ldots, d$, a particular entry $A_{i,j}$ corresponds to the multiplicity of a length-k string obtained from appending character i to the right of a length-$(k-1)$ string of type j in ψ, whereas $A_{i,*}$ corresponds to the multiplicity of a length-k string obtained from appending character i to the right of an all-$*$ string of length $k-1$. To avoid generating an all-$*$ string of length-k, we impose the condition $A_{*,*} = 0$. A column sum c_j, for $j = 1, \ldots, d$, is given by the multiplicity of string type j in ψ, and row sums r_0 and r_1, respectively denoting the number of 0s and 1s at site k in the corresponding new AC, satisfy $(r_0, r_1) \in I(n_0^k, n_1^k)$ (see Section 3.3). Further, as described above, the number c_* of all-$*$ prefixes of length $k-1$ and the number r_* of $*$s at site k satisfy $\max(c - r_0 - r_1, 0) \le r_* \le c$, $\max(r_0 + r_1 - c, 0) \le c_* \le r_0 + r_1$, and $c_1 + \cdots + c_d + c_* = r_0 + r_1 + r_*$. The number of non-zero entries in the above contingency table gives the number of *distinct* length-k strings in the corresponding new AC. In the next iteration, non-zero $A_{i,j}$ will appear as possible column sums when constructing contingency tables for site $k+1$. Returning to our example, contingency tables corresponding to the ACs discussed before are shown in Figures 7b-d. In Figure 7b, all strings in the AC are distinct, and the corresponding contingency table contains only 1s as non-zero entries. In Figures 7c and 7d, the string 000 has multiplicity 2, and the entry $A_{0,1}$, which corresponds to the multiplicity of the length-3 string obtained from appending 0 to the right of 00, is 2.

In summary, there is a one-to-one correspondence between non-negative integer valued contingency tables described above and ACs for the first k sites that can be created by appending 0s, 1s and $*$s to the right of the strings in an AC for the first $k-1$ sites. Hence, we can enumerate ACs by counting contingency tables.

Since the table described above has $A_{*,*} = 0$, we call it a *restricted* contingency table with row sums $\boldsymbol{r} = (r_0, r_1, r_*)$ and column sums $\boldsymbol{c} = (c_1, \ldots, c_d, c_*)$. If the entry $A_{*,*}$ were not restricted to be zero, we would have a standard (or an *unrestricted*) contingency table with row sums $\boldsymbol{r}$ and

(a) $\begin{bmatrix} 0\,0 \\ 0\,0 \\ 1\,* \\ *\,1 \end{bmatrix}$ $c_1 = 2$, $c_2 = 1$, $c_3 = 1$

(b)
$$\begin{bmatrix} 0\,0\,* \\ 0\,0\,0 \\ 1\,*\,0 \\ *\,1\,0 \\ *\,*\,0 \end{bmatrix} \quad \begin{array}{cccc|c} 1 & 1 & 1 & 1 & 4 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ \hline 2 & 1 & 1 & 1 & \end{array}$$

(c)
$$\begin{bmatrix} 0\,0\,0 \\ 0\,0\,0 \\ 1\,*\,* \\ *\,1\,0 \\ *\,*\,0 \end{bmatrix} \quad \begin{array}{cccc|c} 2 & 0 & 1 & 1 & 4 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ \hline 2 & 1 & 1 & 1 & \end{array}$$

(d)
$$\begin{bmatrix} 0\,0\,0 \\ 0\,0\,0 \\ 1\,*\,0 \\ *\,1\,* \\ *\,*\,0 \end{bmatrix} \quad \begin{array}{cccc|c} 2 & 1 & 0 & 1 & 4 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ \hline 2 & 1 & 1 & 1 & \end{array}$$

Fig. 7. Constructing ACs for $D = [(0001, 0000, 1101, 1110), (1, 1, 1, 1)]$ by extending strings. For clarity of illustration, ACs are represented as lists of strings inside [ ]; a string is read horizontally. (a) An AC for the first two sites. (b)-(d) ACs and their corresponding restricted contingency tables obtained from extending the AC in (a) by an additional column with $r_0 = 4, r_1 = 0, r_* = 1$. Note that $c_* = 1$ in (b)-(d), thus satisfying $c + c_* = r_0 + r_1 + r_*$.

column sums $\boldsymbol{c}$. Restricted and unrestricted contingency tables are related as follows. Let $N_u(\boldsymbol{r}, \boldsymbol{c})$ denote the number of *unrestricted* contingency tables with column sums $\boldsymbol{c}$ and row sums $\boldsymbol{r}$. Similarly, let $N_r(\boldsymbol{r}, \boldsymbol{c})$ denote the number of *restricted* contingency tables with row sums $\boldsymbol{r}$ and column sums $\boldsymbol{c}$. For $\boldsymbol{r} = (r_0, r_1, r_*)$ and $\boldsymbol{c} = (c_1, \ldots, c_d, c_*)$, define $\boldsymbol{r}' = (r_0, r_1, r_* - 1)$ and $\boldsymbol{c}' = (c_1, \ldots, c_d, c_* - 1)$. Then, it is straightforward to show that

$$N_r(\boldsymbol{r}, \boldsymbol{c}) = \begin{cases} N_u(\boldsymbol{r}, \boldsymbol{c}) - N_u(\boldsymbol{r}', \boldsymbol{c}') & \text{if } r_* \geq 1 \text{ and } c_* \geq 1, \\ N_u(\boldsymbol{r}, \boldsymbol{c}) & \text{otherwise.} \end{cases} \tag{3.5}$$

We have not yet discussed how to obtain the total number of ACs. The case with two sites is amenable to analytic techniques. For more than two sites, we propose a solution via dynamic programming. These topics are subsequently discussed in the next two subsections.

3.6 Counting ACs for two sites

If there are only two sites in D, we only need to consider contingency tables of size 3×3. This simplifies the problem considerably, and it is possible to sum over all necessary 3×3 contingency tables explicitly to obtain closed-form formulas. We describe this analytical result below.

It follows from the discussion in the previous section that the number of ACs with c_0 0s and c_1 1s at the first site and r_0 0s and r_1 1s at the second site is given by the following sum of restricted contingency table numbers:

$$\sum_{j=\max(r_0+r_1, c_0+c_1)}^{r_0+r_1+c_0+c_1} N_r\big((r_0, r_1, j - r_0 - r_1), (c_0, c_1, j - c_0 - c_1)\big),$$

where the sum over j accounts for the allowed values of c_* and r_*. By using (3.5), this sum can be shown to be equal to $N_u((r_0, r_1, c_0 + c_1), (c_0, c_1, r_0 + r_1))$. Hence, the total number $\beta(D)$ of ACs for the case with two sites is

$$\sum_{\substack{(c_0, c_1) \in I(n_0^1, n_1^1), \\ (r_0, r_1) \in I(n_0^2, n_1^2)}} N_u((r_0, r_1, c_0 + c_1), (c_0, c_1, r_0 + r_1)), \tag{3.6}$$

where, as before, $I(n_0^i, n_1^i)$ determines the allowed number of 0s and 1s at site i.

The key observation is that the expression shown in (3.6) can be summed explicitly. A 3×3 *unrestricted* contingency table of the form

			row sums
a	b	v	r_0
c	d	w	r_1
x	y	z	$c_0 + c_1$
c_0	c_1	$r_0 + r_1$	
column sums:			

has only 4 degrees of freedom; once the entries a, b, c, d are chosen, the other entries get fixed by given row and column sums. Moreover, a, b, c, d satisfy the following set of constraints:

$$\begin{aligned} 0 \leq a + b \leq r_0, \\ 0 \leq c + d \leq r_1, \\ 0 \leq a + c \leq c_0, \\ 0 \leq b + d \leq c_1. \end{aligned}$$

These constraints imply

$$\begin{aligned} a \in \mathcal{A} &= \{0, \ldots, \max(c_0, r_0)\}, \\ b \in \mathcal{B} &= \{0, \ldots, \min(c_1, r_0 - a)\}, \\ c \in \mathcal{C} &= \{0, \ldots, \min(r_1, c_0 - a)\}, \\ d \in \mathcal{D} &= \{0, \ldots, \min(r_1 - c, c_1 - b)\}, \end{aligned}$$

and therefore (3.6) can be written as

$$\beta(D) = \sum_{\substack{(c_0, c_1) \in I(n_0^1, n_1^1), \\ (r_0, r_1) \in I(n_0^2, n_1^2)}} \sum_{a \in \mathcal{A}} \sum_{b \in \mathcal{B}} \sum_{c \in \mathcal{C}} \sum_{d \in \mathcal{D}} 1.$$

This sum can be performed explicitly. For example, for $n_0^1 = n_1^1 = n_0^2 = n_1^2 = n/2$, for n even, the total number $\beta(D)$ of ACs is given by the following degree-8 polynomial:

$$\begin{aligned} \beta(D) \;=\; & 2 + \frac{179}{84} n + \frac{2339}{1260} n^2 + \frac{151}{160} n^3 + \frac{3851}{11520} n^4 \\ & + \frac{31}{384} n^5 + \frac{259}{23040} n^6 + \frac{11}{13440} n^7 + \frac{11}{430080} n^8. \end{aligned} \tag{3.7}$$

Note that the degree 8 is equal to the number $3^L - 1$ of distinct strings over $\{0, 1, *\}$, except for the all-$*$ string, of length $L = 2$

3.7 Counting ACs for arbitrary number of sites

For more than two sites, we adopt a dynamic programming approach for counting *restricted* contingency tables. As mentioned in Section 3.5, our algorithm progresses sequentially along the sites, from left to right. There also exists a more efficient, albeit somewhat more complicated, dynamic programming formulation in terms of counting *unrestricted* contingency tables, but we will not discuss that here. Further, we remark that we considered an alternative approach to counting ACs that is based on counting lattice points bounded by polytopes. Computing the number of such lattice points is a widely-studied problem in mathematics (for example, see [13] and references therein). Moreover, there is a public software called *LattE* (available at http://www.math.ucdavis.edu/∼latte/) that enumerates lattice points, but we found that it is significantly

slower than our own program for enumerating ACs via counting restricted contingency tables; it seems that $L = 2$ is the only case that *LattE* can handle.

Recall that both contingency tables shown in Figures 7c and 7d have $2, 1, 1, 1$ as non-zero entries, although the two tables have different corresponding ACs. In each AC, as the corresponding contingency table captures, there are 4 distinct types of length-3 strings, with exactly one type being of multiplicity 2, while every other type is of multiplicity 1. Now, consider extending the ACs in Figures 7c and 7d to create ACs for the first four sites. For both cases, we need to find contingency tables of the following form:

					row sums
$A_{0,1}$	$A_{0,2}$	$A_{0,3}$	$A_{0,4}$	$A_{0,*}$	r_0
$A_{1,1}$	$A_{1,2}$	$A_{1,3}$	$A_{1,4}$	$A_{1,*}$	r_1
$A_{*,1}$	$A_{*,2}$	$A_{*,3}$	$A_{*,4}$	0	r_*
2	1	1	1	c_*	column sums:

where $(r_0, r_1) \in I(n_0^4, n_1^4)$. As a consequence, the same set of contingency tables will be generated, although the corresponding set of ACs will be different for the two cases. (In the ACs obtained from extending the AC in Figure 7c, possible prefixes of length-3 are "000", "1∗∗", "∗10", "∗∗0" and "∗∗∗", whereas in the ACs obtained from extending the AC in Figure 7d, possible prefixes of length-3 are "000", "1 ∗ 0", "∗1∗", "∗ ∗ 0" and "∗ ∗ ∗".) Hence, since we are only interested in *counting* ACs, and not in constructing the actual ACs themselves, we do not need to consider extensions of the ACs in Figures 7c and 7d separately. At each iteration of our algorithm, we only keep track of distinct multiplicity configurations and how often each configuration appears. Then, we construct contingency tables for each distinct multiplicity configuration. This method considerably reduces the number of contingency tables to construct.

We now describe our enumeration algorithm. Given a matrix M, let $\Lambda(M)$ denote the descending array of positive integers in M. For example, if

$$M = \begin{pmatrix} 3 & 2 & 0 & 4 \\ 1 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix},$$

then $\Lambda(M) = (4, 3, 2, 2, 1, 1)$. Suppose that M and M' are contingency tables that arise while considering site k. When constructing contingency tables for site $k+1$, we do not need to distinguish M from M' if $\Lambda(M) = \Lambda(M')$; we just need to keep track of how many tables for site k have $\Lambda(M)$ as descending array of positive integers. (Note that $\Lambda(M)$ are used as column sums when constructing contingency tables for site $k + 1$.)

Given a k-tuple $\boldsymbol{v} = (v_1, v_2, \ldots, v_k)$ of non-negative integers, we define $|\boldsymbol{v}| = \sum_{i=1}^{k} v_i$. Let $\mathcal{R}(\boldsymbol{c}, \boldsymbol{r})$ be the set of all restricted contingency tables with column sums $(\boldsymbol{c}, c_*)$ and row sums $(\boldsymbol{r}, r_*)$, where $\max(|\boldsymbol{r}| - |\boldsymbol{c}|, 0) \le c_* \le |\boldsymbol{r}|$ and $\max(|\boldsymbol{c}| - |\boldsymbol{r}|, 0) \le r_* \le |\boldsymbol{c}|$. As usual, (n_0^i, n_1^i) denote the number of 0s and 1s at site i in the input data D.

For each site i, we associate a set $\mathcal{C}_i$ of column sums. To initialize the algorithm, we define

$$\mathcal{C}_1 := \left\{ \Lambda(r_0, r_1) \;\middle|\; (r_0, r_1) \in I(n_0^1, n_1^1) \right\}.$$

For each $\boldsymbol{c}^1 \in \mathcal{C}_1$, the *multiplicity* of $\boldsymbol{c}^1$ is defined as

$$\mu(\boldsymbol{c}^1) := \left| \left\{ (r_0, r_1) \in I(n_0^1, n_1^1) \mid \Lambda(r_0, r_1) = \boldsymbol{c}^1 \right\} \right|.$$

For the kth site, where $k > 1$, $\mathcal{C}_k$ is recursively defined as

$$\mathcal{C}_k := \left\{ \Lambda(M) \;\middle|\; \begin{array}{l} M \in \mathcal{R}(\boldsymbol{c}^{k-1}, \boldsymbol{r}^k), \text{ where} \\ \boldsymbol{c}^{k-1} \in \mathcal{C}_{k-1} \text{ and } \boldsymbol{r}^k \in I(n_0^k, n_1^k) \end{array} \right\}.$$

Note that only distinct $\Lambda(\boldsymbol{r}^k)$ need to be considered when constructing $\mathcal{C}_k$. For every $\boldsymbol{c}^{k-1} \in \mathcal{C}_{k-1}$ and every $\boldsymbol{c}^k \in \mathcal{C}_k$, we count the following number of restricted contingency tables M with $\Lambda(M) = \boldsymbol{c}^k$:

$$w(\boldsymbol{c}^{k-1}, \boldsymbol{c}^k) := \sum_{\boldsymbol{r}^k \in I(n_0^k, n_1^k)} \left| \left\{ M \in \mathcal{R}(\boldsymbol{c}^{k-1}, \boldsymbol{r}^k) \middle| \Lambda(M) = \boldsymbol{c}^k \right\} \right|.$$

This can then be used to define the multiplicity of $\boldsymbol{c}^k \in \mathcal{C}_k$ as

$$\mu_k(\boldsymbol{c}^k) = \sum_{\boldsymbol{c}^{k-1} \in \mathcal{C}_{k-1}} \mu_{k-1}(\boldsymbol{c}^{k-1})\, w(\boldsymbol{c}^{k-1}, \boldsymbol{c}^k).$$

In practice, $\mathcal{C}_k$ and $\mu_k(\cdot)$ can be determined concurrently.

The total number $\beta(D)$ of ACs for D is obtained by summing the multiplicity of column sums for the last site:

$$\beta(D) = \sum_{\boldsymbol{c}^L \in \mathcal{C}_L} \mu_L(\boldsymbol{c}^L). \tag{3.8}$$

More generally, the total number of ACs for $D[1:k]$, restriction of D to the first k sites, is

$$\beta(D[1:k]) = \sum_{\boldsymbol{c}^k \in \mathcal{C}_k} \mu_k(\boldsymbol{c}^k).$$

We have written a C++ program that implements the dynamic programming algorithm described above. For two sites, we have checked numerically that (3.8) gives the same answers as does (3.6).

3.8 Some explicit enumeration of ACs

For n even, $n_0^i = n_1^i = n/2$ for every site i leads to the largest number of ACs. For n odd, the largest number of ACs is achieved if $n_0^i = \lfloor n/2 \rfloor + 1$ and $n_1^i = \lfloor n/2 \rfloor$ for every site i, where $\lfloor \cdot \rfloor$ denotes the floor function. The left hand side of Table 2 shows such largest possible number of ACs for small number n of sequences and small number L of sites. It is quite striking that the number of ACs grows so fast as the data size increases. These numbers were obtained using our software that implements the dynamic programming algorithm described in Section 3.7. For $L = 2$ and n even, our closed-form formula (3.7) agrees with the numerical values shown in Table 2.

For fixed L, the total number of ACs is bounded from above by a polynomial in n of degree $3^L - 1$. There are $3^L - 1$ non-negative integer valued variables $m_1, \ldots, m_{3^L-1}$, and the maximum value that any variable can take is n. Hence, the total number of ACs is $O(n^{3^L-1})$. As this simple analysis shows, the total number of ACs depends more crucially on the number of sites than on the number of sequences. This qualitative behavior of growth is also apparent in Table 2.

In the absence of recombination, there are 2^L distinct binary sequences of length L, but that the genealogy is given by

TABLE 2

Total number of ACs for data D with, for every site i, $n_0^i = a_0$ and $n_1^i = a_1$, where $a_0 = a_1 = \lfloor n/2 \rfloor$ for n even, and $a_0 = \lfloor n/2 \rfloor + 1, a_1 = \lfloor n/2 \rfloor$ for n odd. Under these conditions, the maximum value of $\alpha(D)$ is $1 + a_0(a_1 + 2^L - 1)$. These values are shown on the right hand side. Shown on the left hand side is the total number of ACs when an arbitrary number of recombinations are allowed.

$\beta(D)$ (With Recombination)

	L			
n	2	3	4	5
2	30	573	16 875	689 175
3	108	6 286	743 387	149 861 079
4	330	62 589	32 482 009	35 523 729 489
5	866	445 137	893 479 326	4 938 627 635 669
6	2 143	3 302 506	29 521 615 942	962 962 451 049 968
7	4 611	17 409 443	568 860 072 916	91 812 561 254 804 105

$\max(\alpha(D))$ (Without Recombination)

	L			
n	2	3	4	5
2	5	9	17	33
3	9	17	33	65
4	11	19	35	67
5	16	28	52	100
6	19	31	55	103
7	25	41	73	137

a tree puts tight restriction on which sequences can appear together in an AC. As in the above case, for every site i, let $n_0^i = a_0$ and $n_1^i = a_1$, where $a_0 = a_1 = n/2$ if n is even, and $a_0 = \lfloor n/2 \rfloor + 1, a_1 = \lfloor n/2 \rfloor$ if n is odd. Under these conditions, the number $\alpha(D)$ of ACs is maximum if the input data contains a_0 all-0 sequences and a_1 all-1 sequences. One can use the method described in Section 2.2 to show that $\max(\alpha(D)) = 1 + a_0(a_1 + 2^L - 1)$, where L is the number of sites. Some numerical values of this function are shown on the right hand side of Table 2. These numbers are negligibly small, compared to the value of $\beta(D)$ shown on the left hand side of Table 2.

For L sites, let $B_L(k)$ denote the number of ACs with exactly k strings, and let $S_L(k)$ denote the cumulative count $\sum_{i=1}^{k} B_L(i)$, which gives the number of ACs with at most k strings. Table 3 shows $B_L(k)$ and $S_L(k)$ for data sets with $n = 5$ and $(n_0^i, n_1^i) = (3, 2)$ for every site i. The largest value of $S_L(k)$ corresponds to $\beta(D)$ for $n = 5$ in Table 2.

4 Ancestral Configurations in Constrained ARGs

In this section, we enumerate ACs in ARGs with at most R recombinations. This study is motivated by the problem of approximating the likelihood $P(D)$ of the input data D when the recombination rate is low, by summing over the probability of ARGs with small number of recombinations. Further, the work described here can be used to enumerate all ARGs that can generate D using at most R recombinations, under the infinite-sites model of mutation. The constraint on the number of recombinations further complicates the problem of enumerating ACs, and combinatorial analysis is more difficult to carry out. We provide a dynamic programming algorithm that enumerates ACs by explicitly tracking all possible evolutionary histories backwards in time.

(Cautionary Remark: Given a data set D, an AC ψ that is R recombinations away from D may not appear in any ARG for D with at most R recombinations; further recombinations may be required for ψ to reach the grand common ancestor. Hence, the number of ACs in ARGs for D with at most R recombinations is in general smaller than the number of ACs that can be reached from D using at most R recombinations. In what follows, we are interested in counting the former.)

4.1 Examples revisited

As mentioned in Section 3.4, when arbitrary number of recombinations are allowed, the set of ACs does not depend on the order of 0s and 1s within a column. For example, $D = [(00, 11), (1, 1)]$ and $D' = [(01, 10), (1, 1)]$ have the same set of ACs, shown in Figure 6, when arbitrary number of recombinations are allowed. That no longer holds true if there is a restriction on the number of allowed recombinations. This fact is easiest to see in the absence of recombination, in which case the set of ACs for D is $\{[(00, 11), (1, 1)]$, $[(00, 01), (1, 1)]$, $[(00, 10), (1, 1)]$, $[(00), (2)]$, $[(00), (1)]\}$, whereas that for D' is $\{[(01, 10), (1, 1)]$, $[(01, 00), (1, 1)]$, $[(00, 10), (1, 1)]$, $[(00), (2)], [(00), (1)]\}$. Similarly, when only a single recombination is allowed, it is straightforward to show that D and D' have different sets of ACs, although the two sets are both of size 19.

When there is a restriction on the number of recombinations, even the number of ACs may depend on the order of 0s and 1s within a column. Consider the two data sets $D = [(00, 11), (1, 2)]$ and $D' = [(01, 10, 11), (1, 1, 1)]$. In the absence of recombination, D has 6 ACs, while there exists no coalescent tree for D' (c.f. the three gamete test mentioned in Section 2). The minimum number of recombinations for D' is one, and there are 21 inequivalent ACs in the set of ARGs for D' with exactly one recombination. In contrast, there are 26 ACs for D when one recombination is allowed.

In Section 3.4, we observed that, when arbitrary number of recombinations are allowed, two data sets differing by some permutation of their columns have the same number of ACs. This also is no longer true, in general, if the number of recombinations is constrained. For example, consider the two data sets $D = [(001, 110, 111), (1, 1, 1)]$ and $D' = [(010, 101, 111), (1, 1, 1)]$. While D has an ARG with only one recombination, no valid ARG with less than two recombinations exists for D'.

4.2 The search algorithm

For the case with restricted number R of recombinations, the approach we take is to trace explicitly all possible evolutionary histories backwards in time. For any particular AC, we try all possible events—that is, recombination, coalescent or mutation events—that can happen to that AC. Possible recombination and coalescent events are as illustrated in Figure 4. We do not

TABLE 3
Counting of ACs with a restricted number of strings. For L sites, $B_L(k)$ denotes the number of ACs with exactly k strings, while $S_L(k)$ denotes the number of ACs with at most k strings. This table is for data with $n = 5$ and $(n_0^i, n_1^i) = (3, 2)$ for every site i.

k	$B_2(k)$	$S_2(k)$	$B_3(k)$	$S_3(k)$	$B_4(k)$	$S_4(k)$	$B_5(k)$	$S_5(k)$
1	1	1	1	1	1	1	1	1
2	12	13	62	63	312	313	1562	1563
3	60	73	1143	1206	21720	22033	412683	414246
4	163	236	9517	10723	575921	597954	35132963	35547209
5	244	480	40006	50729	6811757	7409711	1168994740	1204541949
6	210	690	89716	140445	38651220	46060931	16622300644	17826842593
7	120	810	118047	258492	115829376	161890307	112796639915	130623482508
8	45	855	99013	357505	201958486	363848793	408212643773	538836126281
9	10	865	56395	413900	222978515	586827308	868467110362	1407303236643
10	1	866	22818	436718	167052986	753880294	1177704929588	2585008166231
11			6732	443450	89613906	843494200	1087103087864	3672111254095
12			1447	444897	35821104	879315304	719916684420	4392027938515
13			218	445115	10971625	890286929	356517388653	4748545327168
14			21	445136	2620233	892907162	136353481775	4884898808943
15			1	445137	491380	893398542	41276692230	4926175501173
16					72009	893470551	10070266029	4936245767202
17					8076	893478627	2004602973	4938250370175
18					662	893479289	327842330	4938578212505
19					36	893479325	44105460	4938622317965
20					1	893479326	4854891	4938627172856
21							431132	4938627603988
22							30060	4938627634048
23							1565	4938627635613
24							55	4938627635668
25							1	4938627635669

consider recombination events that produce an all-$*$ sequence. Recall that, because we assume the infinite-sites model, a mutation of 1 to 0 at a site is possible if and only if exactly one sequence carries a 1 at that site. Going backwards in time in this manner, event by event, we can exhaustively find all sequences of events with less than or equal to the allowed number of recombinations, while keeping track of all distinct ACs encountered. To accelerate the search, we propose two ideas, one based on branch and bound, and the other on dynamic programming.

We maintain a hash table H that stores ACs already encountered, together with the number of recombinations further allowed for each AC. Initially, H contains the pair (D, R), where D is the given data set and R is the maximum number of allowed recombinations. Given an AC ψ and its associated limit c on the number of recombinations, we first compute the minimum number k of recombinations needed in the evolution of ψ; our method of computing k is described in [14]. If $k > c$, we backtrack the search. Otherwise, we check whether ψ is present in H. Suppose that (ψ, c') with $c' \geq c$ is present in H. Then, continuing along the current search path will not lead to any AC that has not been encountered previously, so again it is safe to backtrack. If $c' < c$ or if ψ is not present in H, then we need to add the entry (ψ, c) to H and iteratively try all possible events backwards in time starting from ψ. When either a mutation or a coalescent event is tried, the number of allowed recombinations remains c; when a recombination event is tried, the number of allowed recombinations decreases to $c - 1$. The hash table serves a dual purpose. First, as just described, it allows us to avoid repeating a search already carried out. Second, when all branches have been explored, the number of entries in the hash table equals $\gamma_R(D)$, the number of ACs that can be encountered in ARGs deriving D with at most R recombinations. It is straightforward to produce a list of all $\gamma_R(D)$ inequivalent ACs encountered during the search.

We have written a C program, called `greven`, that implements the method described above. For small data sets, we have checked that this program agrees with `aceTrees` when no recombination is allowed, and with `aceARGs` when an arbitrary number of recombinations are allowed.

4.3 Further explicit enumeration of ACs

For small values of n, we applied our program to all data sets with the same values of n_0^i and n_1^i as in Table 2. Let $R_{\min}(D)$ denote the minimum number of recombinations needed in the evolution of a given set D of sequences. Shown in Table 4 is $\max_D[\gamma_{R_{\min}(D)}(D)]$, i.e. the maximum number, over all D, of ACs that can be encountered in ARGs for D with $R_{\min}(D)$ recombinations. The numbers in parentheses denote the minimum number $R_{\min}(D)$ of recombinations for D with the maximum value of $\gamma_{R_{\min}(D)}(D)$. Table 4 should be compared with Table 2. As this enumeration shows, although the number of ACs encountered in minimal ARGs is noticeable larger than the number of ACs possible in the absence of recombination, it is still negligibly small compared to the total number of ACs when an arbitrary number of recombinations are allowed.

TABLE 4
The maximum number, over all data sets D satisfying $n_0^i = \lceil n/2 \rceil$ and $n_1^i = \lfloor n/2 \rfloor$ for every site i, of ACs that can be encountered in ARGs for D with the minimum number $R_{\min}(D)$ of recombinations. The numbers in parentheses indicate the minimum number $R_{\min}(D)$ of recombinations for D with the maximum value of $\gamma_{R_{\min}(D)}(D)$.

$\max_D \left[\gamma_{R_{\min}(D)}(D)\right]$				
	L			
n	2	3	4	5
2	5(0)	9(0)	17(0)	33(0)
3	9(0)	17(0)	33(0)	65(0)
4	34(1)	216(2)	1 524(3)	12 085(4)
5	60(1)	408(2)	2 986(3)	23 958(4)
6	78(1)	2 709(3)	17 053(4)	1 210 197(6)

5 Discussion

In this paper, we addressed the problem of enumerating all configurations (or Markov states) ancestral to a given data set D, consisting of binary sequences of finite length. Our main motivation was to understand in detail the difficulties involved in computing the likelihood $P(D)$ of observing D. We constructed a general method of obtaining closed-form formulas (as functions of the multiplicity of distinct sequences in D) for the total number of ACs when recombination is absent. For the case in which recombination is allowed, we devised a method of enumeration based on counting contingency tables and constructed a dynamic programming algorithm. In terms of the number of ACs, one can use our methods to study rigorously how the case with recombination compares with the case without recombination. In our work, we assumed that recombination breakpoints occur at the midpoints of consecutive sites in D. If this assumption is relaxed, then the number of ACs will increase.

We also discussed a method of counting the number of ACs that can appear in ARGs with less than or equal to a given number R of recombinations. Of particular interest is the case in which R is (or is close to) the minimum number recombinations for D. When recombination rate is low, it is widely believed that ARGs with the minimum or "near minimum" number of recombinations should make dominant contribution to the likelihood $P(D)$. However, we are not aware of any work that actually tried to sum the probability over all minimal or near minimal ARGs. That problem is the main motivation for our present work. Ultimately, we would like to see how the total probability carried by ARGs with at most R recombinations changes as R increases. The goal of this paper was to understand some basic properties of the Markov state space itself and to gauge the size of data that we may be able to handle.

As discussed in Section 3.8, the total number of ACs can be inordinately large when an arbitrary number of recombinations are allowed. Not all ACs play an equally important role in computing the likelihood, however. We mentioned in Section 3.4 that if D and D' are related by a series of operations involving permutations of characters within a column and/or permutations of entire columns, then they have the same total number of ACs. But, D and D' may look radically different to anybody's eye and carry quite different information. For example, consider the following two contrived data sets:

$$D = \begin{pmatrix} 1010101 \\ 1111111 \\ 0101010 \\ 0000000 \end{pmatrix} \text{ and } D' = \begin{pmatrix} 1111111 \\ 1111111 \\ 0000000 \\ 0000000 \end{pmatrix}.$$

When an arbitrary number of recombinations are allowed, D and D' have the same number of ACs, but they are qualitatively very different; clear signals of past recombinations are present in D, while D' is compatible with a tree. The configuration space is very large, containing about 2.26×10^{17} ACs, but, for low recombination rates, a large portion of those ACs should be insignificant for computing the likelihood $P(D')$. In general, if we are to devise a deterministic method for estimating the likelihood of observing a given data set, we would need to investigate how we can systematically take advantage of our understanding of the configuration space and of the structure of Markov chain transition graph.

Acknowledgments

This work is supported by grants EIA-0220154 and IIS-0513910 (Song) from NSF, by grant HAMJW (Hein, Lyngsø) from EPSRC, and by grant HAMKA (Hein, Lyngsø) from MRC.

References

[1] S. N. Ethier and R. C. Griffiths. The infinitely-many-sites model as a measure valued diffusion. *Ann. Probab.*, 15:515–545, 1987.
[2] S. N. Ethier and R. C. Griffiths. On the two-locus sampling distribution. *J. Math. Biol.*, 29:131–159, 1990.
[3] P. Fearnhead and P. Donnelly. Estimating recombination rates from population genetic data. *Genetics*, 159:1299–1318, 2001.
[4] R. C. Griffiths. Genealogical-tree probabilities in the infinitely-many-site mode. *J. Math. Biol.*, 27:667–680, 1989.
[5] R. C. Griffiths and P. Marjoram. Ancestral inference from samples of DNA sequences with recombination. *J. Comput. Biol.*, 3:479–502, 1996.
[6] R. C. Griffiths and S. Tavaré. Ancestral inference in population genetics. *Stat. Sci.*, 9:307–319, 1994.
[7] R. C. Griffiths and S. Tavaré. Simulating probability distributions in the coalescent. *Theoret. Pop. Biol.*, 46:131–159, 1994.
[8] D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21:19–28, 1991.
[9] J. F. C. Kingman. The coalescent. *Stoch. Process. Appl.*, 13:235–248, 1982.
[10] J. F. C. Kingman. On the genealogy of large populations. *J. Appl. Prob.*, 19A:27–43, 1982.
[11] M. K. Kuhner, J. Yamato, and J. Felsenstein. Estimating effective population size and mutation rate from sequence data using metropolis-hastings sampling. *Genetics*, 140:1421–1430, 1995.
[12] M. K. Kuhner, J. Yamato, and J. Felsenstein. Maximum likelihood estimation of recombination rates from population data. *Genetics*, 156:1393–1401, 2000.
[13] J. De Loera, R. Hemmecke, J. Tauzer, and R. Yoshida. Effective lattice point counting in rational convex polytopes. *J of Symbolic Computation*, 38:1273–1302, 2004.
[14] R. Lyngsø, Y. S. Song, and J. Hein. Minimum recombination histories by branch and bound. In *Proc. of 2005 Workshop on Algorithms in Bioinformatics*, pages 239–250, Berlin, Germany, 2005. Springer-Verlag LNCS.
[15] K. L. Simonsen and G. A. Churchill. A markov chain model of coalescence with recombination. *Theoret. Pop. Biol.*, 52:43–59, 1997.
[16] M. Stephens and P. Donnelly. Inference in molecular population genetics. *J.R. Stat. Sco. Ser. B*, 62:605–655, 2000.
[17] R. H. Ward, B. L. Frazier, K. Dew, and S. Pääbo. Extensive mitochondria diversity within a single amerindian tribe. *Proc. Nat. Acad. Sci.*, 88:8720–8724, 1991.

APPENDIX

Counting lattice points bounded by polytopes

For fixed L, let $\{h_1, \ldots, h_{3^L-1}\}$ be the set of all distinct length-L strings over $\{0, 1, *\}$ except for the all-$*$ string, and define

$$\begin{aligned} \mathcal{H}_0^i &:= \{j \,|\, h_j \text{ has 0 at site } i\}, \\ \mathcal{H}_1^i &:= \{j \,|\, h_j \text{ has 1 at site } i\}. \end{aligned}$$

Recall that the set $I(n_0^i, n_1^i)$, defined in (3.2)–(3.4), gives the possible range of 0s and 1s at site i that can appear in an ancestral configuration. Let $m_j \geq 0$ denote the number of h_j in an AC. Then, the discussion in Section 3.3 implies the following condition: The sum $p_i = \sum_{j \in \mathcal{H}_0^i} m_j$ of the number of strings with 0s at site i and the sum $q_i = \sum_{j \in \mathcal{H}_1^i} m_j$ of the number of strings with 1s at site i satisfy $(p_i, q_i) \in I(n_0^i, n_1^i)$. The total number of ACs can be obtained by counting the number of non-negative integer valued solutions $m_1, \ldots, m_{3^L-1}$ to the following L simultaneous constraints:

$$(p_i, q_i) \in I(n_0^i, n_1^i), \quad i = 1, \ldots, L.$$

A similar approach was taken in [15] for studying the special case where $(n_0^i, n_1^i) = (n, 0)$ for all i.

Let us demonstrate how this method works through a concrete example. Consider a data set D containing n sequences, in which case $n_0^i + n_1^i = n$ for all i. Assume that $L = 2$ for simplicity, and suppose that $n_0^1 \neq 0, n_1^1 \neq 0$ for site 1 and that $n_0^2 = 0, n_1^2 = n$ for site 2. For $L = 2$, the $3^L - 1$ distinct strings we need to consider are as follows:

$$\begin{aligned} h_1 &= 00, & h_5 &= 1*, \\ h_2 &= 01, & h_6 &= *1, \\ h_3 &= 10, & h_7 &= 0*, \\ h_4 &= 11, & h_8 &= *0. \end{aligned}$$

Using (3.2), the site-1 constraint $(p_1, q_1) \in I(n_0^1, n_1^1)$ can be translated to

$$\begin{aligned} &\text{either} \quad (1i) \quad 1 \leq m_1 + m_2 + m_7 \leq n_0^1 \text{ and} \\ &\qquad\qquad\quad 1 \leq m_3 + m_4 + m_5 \leq n_1^1, \\ &\text{or} \quad\;\; (1ii) \quad 1 \leq m_1 + m_2 + m_7 \leq n_0^1 + 1 \text{ and} \\ &\qquad\qquad\quad m_3 = m_4 = m_5 = 0. \end{aligned}$$

Similarly, using (3.3), the site-2 constraint $(p_2, q_2) \in I(n_0^2, n_1^2)$ can be translated to

$$\begin{aligned} &\text{either} \quad (2i) \quad m_1 = m_3 = m_8 = 0 \text{ and} \\ &\qquad\qquad\quad 1 \leq m_2 + m_4 + m_6 \leq n, \\ &\text{or} \quad\;\; (2ii) \quad m_1 + m_3 + m_8 = 1 \text{ and} \\ &\qquad\qquad\quad m_2 = m_4 = m_6 = 0. \end{aligned}$$

The total number of ACs for D is given by the sum of the number of non-negative integer valued solutions $m_1, \ldots, m_8$ to each of the following simultaneous constraints: $(1i)$ and $(2i)$; $(1i)$ and $(2ii)$; $(1ii)$ and $(2i)$; $(1ii)$ and $(2ii)$. Note that one way to solve such a problem is via integer linear programming.

Together with the constraint that m_j need be non-negative integer valued, any pair of simultaneous constraints discussed above defines a polytope in 8-dimensional Euclidean lattice, and the corresponding number of ACs can be computed by counting the number of lattice points bounded by the polytope; i.e. by enumerating the number of lattice points inside or on the boundary of the polytope. More generally, for $L > 2$, the total number of ACs can be determined by summing the number of lattice points bounded by the polytopes in $(3^L - 1)$-dimensional Euclidean lattice defined by the simultaneous constraints $(p_1, q_1) \in I(n_0^1, n_1^1), \ldots, (p_L, q_L) \in I(n_0^L, n_1^L)$.